

PYTHON FOR KIDS

A PLAYFUL INTRODUCTION TO PROGRAMMING

JASON R. BRIGGS



Python za otroke

Igriva predstavitev programiranja

Python for Kids

Jason R. Briggs

Ilustrator: Miran Lipovača

No Starch Press, San Francisco, 2013

Prevod: Aleš Drinovec, 2018

Mnenja o knjigi

Python za otroke

"Odlična povezovalna izkušnja za družino."

-Patrice Gans, Education Week

"Prav tako dober uvod za odrasle, ki bi radi programirali."

-Matthew Humphries, Geek.com

"Jason Briggs uspešno opiše programiranje za otroke brez odvečnega 'nakladanja'. Lekcije so dobro zgrajene in pustijo bralcu občutek, da v vsakem poglavju doseže cilj. "

-Marziah Karch, Wired.com's GeekMom blog

"Dobro napisana in privlačna. Nič ni lepšega kot videti, da se v nekaj vrsticah nekaj zgodi in vedeti, zakaj se je zgodilo. "

-Copil Yáñez, revija Full Circle

"Odličen uvod v programiranje za vsakogar, ki ga zanima učenju programiranja, ne glede na njegovo starost. Odlično vir za dom ali šolo."

-Roy Wood, GeekDad

"Prosila sem mojo osemletnico naj pove, kaj misli o Pythonu za otroke. Izbrala je pet zvezdic... Všeč mi je, da moja hčerka piše prave programe v Pythonu s to knjigo."

-Richard Bejtlich, CSO pri Mandiant

"Odlična knjiga za vsakogar, ki želi prodreti v programiranje brez, da bi trpel zaradi neustreznosti."

-Sandra Henry-Stocker, ITworld

Vsebina

O avtorju.....	13
O ilustratorju	13
O tehničnih pregledovalcih	13
Zahvale	14
Uvod	15
Zakaj Python?	15
Kako se naučiti kodiranja.....	15
Kdo naj prebere to knjigo	16
Kaj je v tej knjigi.....	16
Spletna stran v podporo	17
Zabavajte se!	17
1. poglavje: Vse kače ne pičijo.....	18
Namestitev Pythona v operacijskem sistemu Windows 7 in višje	18
Ko ste namestili Python.....	19
Shranjevanje Python programov.....	20
Kaj ste se naučili	21
2. poglavje: Računanje in spremenljivke	22
Računanje s Pythonom.....	22
Operatorji v Pythonu	23
Vrstni red operacij.....	23
Spremenljivke so kot oznake	24
Uporaba spremenljivk	25
Kaj ste se naučili	27
3. poglavje: String, list, tuple, map	28
Nizi (Strings)	28
Ustvarjanje nizov	28
Reševanje težav z nizi	29
Vsebovane vrednosti v nizih.....	30
Množenje nizov	31
Seznami so močnejši od nizov	32
Dodajanje elementov v seznam	34
Odstranjevanje elementov iz seznama	34
Računanje s seznamami	35

Tuple (terka).....	36
S Python map (zemljevid) ne boste našli svoje poti.....	37
Kaj ste se naučili	38
4. poglavje: Risanje z želvami	39
Uporaba Python turtle modula	39
Ustvarjanje platna	39
Premikanje želve	41
Kaj ste se naučili	44
Vaje.....	44
# 1: pravokotnik.....	44
# 2: Trikotnik.....	44
# 3: Škatla brez vogalov	44
5. poglavje: Postavljanje vprašanj z IF in ELSE.....	46
Pogojni stavki.....	46
Blok je skupina programskih ukazov	46
Pogoji nam pomagajo primerjati stvari	48
If-then-else stavki	49
If in elif stavki.....	50
Združevanje pogojev	50
Spremenljivke brez vrednosti – None	51
Razlika med nizi in števili.....	51
Kaj ste se naučili	53
Vaje.....	53
# 1: Ali ste bogati?	53
# 2: Kolački!	53
# 3: Samo prava številka.....	53
# 4: Lahko se borim proti tistim ninjam	54
6. poglavje: Gremo se zanke	55
Uporaba for zank.....	55
Medtem, ko govorimo o zankanju... ..	59
Kaj ste se naučili	61
Vaje.....	61
# 1: Zanka pozdravljen.....	61
# 2: Parne in neparne številke	62
# 3: Mojih pet najljubših sestavin.....	62
# 4: Vaša teža na Luni	62

7. poglavje: Predelava kode s funkcijami in moduli	63
Uporaba funkcij	63
Deli funkcije	63
Spremenljivke in področje uporabe	64
Uporaba modulov	66
Kaj ste se naučili	67
Vaje	68
# 1: Osnovna funkcija za težo na Luni	68
# 2: funkcija teže na Luni in leta	68
# 3: Program za težo na Luni	68
8. poglavje: Kako uporabljati razrede in objekte.....	69
Razbijanje stvari v razrede.....	69
Otroci in starši	70
Dodajanje objektov v razrede	70
Definiranje funkcij v razredih	71
Dodajanje značilnosti razreda kot funkcij	71
Zakaj uporabljati razrede in objekte?.....	72
Objekti in razredi v slikah	73
Druge uporabne lastnosti objektov in razredov.....	75
Podedovane funkcije	75
Funkcije, ki kličejo druge funkcije.....	76
Inicializacija objekta	76
Kaj ste se naučili	77
Vaje.....	77
# 1: Žirafji ples	78
# 2: Želvji štirizob.....	78
9. poglavje: Pythonove vgrajene funkcije	79
Uporaba vgrajenih funkcij	79
Funkcija abs	79
Funkcija bool.....	80
Funkcija dir	81
Funkcija eval	82
Funkcija exec	83
Funkcija float	83
Funkcija int	83
Funkcija len.....	84

Funkciji max in min	85
Funkcija range	85
Funkcija sum	86
Delo z datotekami	86
Ustvarjanje poskusne datoteke	87
Odpiranje datoteke v Pythonu	87
Odpiranje datoteke v Windows.....	87
Pisanje v datoteke	87
Kaj ste se naučili	88
Vaje.....	88
# 1: Skrivnostna koda	88
# 2: Skrito sporočilo	88
# 3: Kopiranje datoteke	88
10. poglavje: Uporabni moduli	89
Izdelava kopij z modulom copy	89
Na sledi ključnim besedam z modulom keywords	91
Z modulom random do naključnih števil.....	91
Uporaba randint vrne naključno celo število	91
Uporaba choice za izbiro naključnega predmeta iz seznama	93
Uporaba shuffle za premešanje seznama	93
Nadzor lupine z modulom sys	93
Branje z objektom stdin.....	93
Pisanje z objektom stdout	94
Katere verzijo Pythona uporabljam?	94
Obdelovanje časa z modulom time	94
Pretvarjanje datuma z asctime.....	95
Pridobivanje datuma in časa z lokalnim časom.....	95
Pavza s funkcijo sleep	96
Uporaba modula pickle za shranjevanje informacije	96
Kaj ste se naučili	97
Vaje.....	97
# 1: Kopirani avtomobili	97
# 2: Shranjene priljubljene vrednosti	98
11. poglavje: Več želvje grafike	99
Začnimo z osnovnim kvadratom	99
Risanje zvezd	99

Risanje avtomobila	102
Kako pobarvamo stvari.....	103
Funkcija za risanje pobarvanega kroga	104
Ustvarjanje čiste črne in bele	105
Funkcija za risanje kvadrata	105
Risanje zapolnjenih kvadratov.....	106
Risanje zapolnjenih zvezd.....	107
Kaj ste se naučili	108
Vaje.....	108
# 1: Risanje osemkotnika.....	109
# 2: Risanje pobarvanega osemkotnika.....	109
# 3: Druga funkcija za risanje zvezd.....	109
12. poglavje: Uporaba tkinter za boljšo grafiko	110
Ustvarjanje gumba za klikanje.....	110
Uporaba imenovanih parametrov.....	111
Ustvarjanje platna za risanje	112
Risanje črt	112
Risanje likov.....	113
Risanje množice pravokotnikov.....	115
Nastavljanje barv	116
Risanje lokov.....	119
Risanje večkotnikov	120
Prikazovanje besedila	121
Prikazovanje slik	122
Ustvarjanje osnovne animacije	123
Odziv objekta na dogodek.....	125
Več načinov uporabe identifikatorja	127
Kaj ste se naučili	128
Vaje.....	128
# 1: Izpolnite zaslon s trikotniki	128
# 2: Premikajoči se trikotnik	128
# 3: Premikajoča se fotografija	128
13. poglavje: Začetek vaše prve igre: Odbij!	129
Udari žogico.....	129
Ustvarjanje igralnega platna	129
Ustvarjanje razreda Ball	130

Dodajanje nekaj akcije.....	131
Premaknimo žogo.....	132
Žogica naj se odbija	133
Nastavitev začetne smeri žogice	134
Kaj ste se naučili	135
14. poglavje: Dokončanje vaše prve igre: Odbij!.....	136
Dodajanje loparja	136
Premikanje loparja	137
Kdaj žoga zadene lopar.....	138
Dodajanje elementa naključnosti.....	140
Kaj ste se naučili	143
Vaje.....	143
# 1: Zakasnitev začetka igre.....	143
# 2: Pravilen "Konec igre"	143
# 3: Pospešite žogico	144
# 4: Zapišite rezultat igralca.....	144
15. poglavje: Izdelava grafike za igrico Mr. Stickman.....	145
Načrt igre Mr. Stickman.....	145
Namestimo GIMP	145
Ustvarjanje elementov igre	147
Priprava prosojnega ozadja slike	147
Risba Mr. Stickmana	148
Mr. Stickman, ki teče desno	148
Mr. Stickman teče proti levi	149
Risanje platform	149
Risanje vrat	150
Risanje ozadja	150
Prosojnost.....	151
Kaj ste se naučili	152
16. poglavje: Razvoj igre Mr. Stickman	153
Izdelava razreda Game	153
Nastavitev naslova okna in izdelava platna.....	153
Končanje funkcije <code>_init_</code>	154
Izdelava glavne zanke - funkcija <code>mainloop</code>	155
Izdelava razreda <code>Coords</code>	156
Preverjanje trkov	157

Trk sličic vodoravno	157
Trk sličic navpično	158
Sestavimo skupaj: naša končna koda za preverjanje trka	158
Funkcija collided_left	158
Funkcija collided_right	159
Funkcija collided_top	159
Funkcija collided_bottom	160
Izdelava razreda Sprite	160
Dodajanje platform	161
Dodajanje objekta platform	161
Dodajanje skupine platform	162
Kaj ste se naučili	163
Vaje	163
# 1: Šahovnica	163
# 2: Dvobarvna šahovnica	164
# 3: Knjižna polica in svetilka	164
17. poglavje: Izdelava Mr. Stickmana	165
Inicializacija glavnega lika	165
Nalaganje slik Mr. Stickmana	165
Nastavitev spremenljivk	166
Povezovanje s tipkami	167
Obračanje glavne figure levo in desno	167
Ustvarjanje skokov	167
Kaj imamo do sedaj	168
Kaj ste se naučili	169
18. poglavje: Dokončanje igre Mr. Stickman	170
Animiranje glavnega igralca	170
Ustvarjanje funkcije Animate	170
Pozicija glavnega lika	173
Premikanje glavnega lika	173
Testiranje našega glavnega lika	178
Vrata!	179
Ustvarjanje razreda DoorSprite	179
Ugotavljanje stanja vrat	180
Dodajanje objekta door	180
Končana igra	181

Kaj ste se naučili	186
Vaje.....	186
# 1: "Zmagal si!".....	186
# 2: Animacija vrat.....	186
# 3: Premične platforme.....	187
Zaključna beseda - kako naprej.....	188
Igre in grafično programiranje.....	188
PyGame	188
Programski jeziki.....	189
Java	189
C/C++	189
C#.....	190
PHP	190
Objective-C.....	191
Perl.....	191
Ruby.....	191
JavaScript.....	191
Zaključna beseda	192
Dodatek - rezervirane besede v Pythonu	193
and.....	193
as	193
assert	193
break.....	194
class	194
continue.....	195
def.....	195
del.....	195
elif.....	196
else	196
except.....	196
finally	196
for	196
from	196
global	197
if.....	198
import.....	198

in	198
is	198
lambda.....	199
not	199
or	199
pass.....	199
raise	200
return.....	201
try	201
while.....	201
with.....	201
yield	201
Slovar	202

O avtorju

Jason R. Briggs je programer od osmega leta, ko se je prvič učil BASIC na Radio Shack TRS-80. Profesionalno je programiral kot razvijalec in sistemski arhitekt ter bil zunanji sodelavec za Java Developer's Journal. Njegovi članki so bili objavljeni v JavaWorld, ONJava in ONLamp. Python for Kids je njegova prva knjiga.

Jasona je mogoče doseči na <http://jasonrbriggs.com/> ali po elektronski pošti na mail@jasonrbriggs.com.

O Ilustratorju

Miran Lipovača je avtor Learn You a Haskell for Great Good! Uživa v boksu, igranju bas kitare in seveda, risanju. Navdušen je nad plešočimi okostnjaki in nad številko 71. Ko gre skozi drsna vrata, se pretvarja, da jih dejansko odpira s svojim razumom.

O tehničnih pregledovalcih

Nedavni diplomant Nueva School, 15-letni Josh Pollock je novinec na srednji šoli Lick-Wilmerding v San Franciscu. Najprej je začel programirati v Scratchu, ko je bil star 9 let, začel uporabljati TI-BASIC, ko je bil v 6. razredu, in nadaljeval z Java in Python v sedmem in UnityScript v 8 razredu. Poleg programiranja rad igra trobento, razvija računalniške igre in uči ljudi o zanimivih temah STEM.

Maria Fernandez ima magisterij iz uporabnega jezikoslovja in jo zanimajo računalniki in tehnologija že več kot 20 let. Angleščino je učila mlade begunke v Global Village Project v Georgiji in je trenutno na severu Kalifornija na projektu ETS (Educational Testing Service).

Zahvale

To je tako kot, ko pridete na oder sprejeti nagrado in ko se želite zahvaliti, ugotovite, da ste spisek oseb pozabili v žepu drugih hlač: zagotovo boste nekoga pozabili in glasba vas bo kmalu začela priganjati, da se umaknete z odra.

Torej, kot rečeno, tu je (nedvomno) nepopoln seznam ljudi, ki jim dolgujem veliko zahvalo za pomoč pri izdelavi te knjige tako dobre, kot mislim, da je sedaj.

Hvala ekipi No Starch, zlasti Billu Pollocku, za uporabo liberalnega odmerka »kaj bi si mislil otrok«, pri urejanju. Ko že dolgo programirate, se je težko spomniti, kako težko je nekaj za učence, Bill pa je bil neprecenljiv pri izpostavljanju pogosto prezrtih, preveč zapletenih delov. In zahvaljujoč Sereni Yang, izredni izvajalki; tukaj upam, da vam ni izpadlo preveč las ob več kot 300 straneh pravilno obarvane kode.

Velika zahvala gre Miranu Lipovači za briljantne ilustracije. Več kot briljantne. Res! Če bi sam moral narediti umetniški del, bi bili veseli, če bi bila tu ali tam kakšna slika, ki ne spominja na ničesar posebnega. Ali je to medved...? Je pes...? Ne, počakaj ... naj bi bilo to drevo?

Hvala pregledovalcema. Opravičujem se, če kakšen predlog na koncu ni bil upoštevan. Verjetno ste imeli prav in krivim lahko le moj osebni značaj za morebitne neumnosti. Posebna zahvala Joshu za nekaj zelo dobrih predlogov in odpravo nekaj resnih napak. In opravičujem se Mariji, ki je morala občasno pregledovati neumno kodo.

Hvala moji ženi in hčerki, ki sta prenašali moža in očeta, ko je bil z nosom v računalniškem zaslonu še več kot običajno.

Mami, za neskončne količine spodbud v preteklih letih.

Nazadnje, se zahvaljujem še mojemu očetu, za nakup računalnika daleč nazaj v sedemdesetih letih prejšnjega stoletja in ki ga je postavil pred nekoga, ki ga je želel uporabljati toliko, kot ga je. Nič od tega ne bi bilo mogoče brez njega.

Uvod

Zakaj se naučiti računalniškega programiranja?

Programiranje spodbuja ustvarjalnost, razmišljanje in reševanje problemov. Programer dobi priložnost, da iz nič nekaj ustvari, uporabi logiko, da poveže programske gradnike v obliko, ki jo razume računalnik in ko stvari ne delujejo tako, kot so bile pričakovane, uporabi načine reševanja problemov za odkrivanje napak. Programiranje je zabavna, včasih zahtevna (in občasno frustrirajoča) aktivnost, in spretnosti, ki se jih pri tem naučite so lahko uporabne tako v šoli kot tudi pri delu... tudi, če vaša kariera nima nič skupnega z računalniki.

In, če nič drugega, je programiranje odličen način preživljanja popoldnevov, ko je zunaj slabo vreme.

Zakaj Python?

Python je programski jezik, ki se ga je mogoče enostavno naučiti in ima nekaj res uporabnih lastnosti za programerja začetnika. Kodo je precej enostavno brati v primerjavi z drugimi programskimi jeziki in ima interaktivno lupino (shell), v katero lahko vnašate svoje programe in vidite, kako tečejo. Poleg enostavne jezikovne strukture in interaktivne lupine za ekperimentiranje, ima Python nekaj lastnosti, ki močno povečujejo učni proces in vam omogočajo sestaviti preproste animacije za ustvarjanje lastnih iger. Prvi je modul turtle (želva), ki ga navdihuje grafika Turtle (uporabljal ga je programski jezik Logo v šestdesetih letih prejšnjega stoletja), in je zasnovan v izobraževalne namene. Drugi je modul tkinter, vmesnik za Tk grafični uporabniški vmesnik (GUI - Graphic User Interface), ki omogoča preprost način ustvarjanja programov z malo bolj napredno grafiko in animacijo.

Kako se naučiti kodiranja

Kot karkoli, kar prvič poskusite, je vedno najbolje, da začnete z osnovami, torej začnite s prvimi poglavji in ne podležite željam po preskakovanju poglavij. Nihče ne more igrati v simfoničnem orkestru, ko prvič vzame v roke instrument. Pri učenju letenja se je treba najprej naučiti osnovnih kontrol. Gimnastiki ne (ponavadi) znajo narediti salte ali vijaka že ob prvem poskusu. Če prehitro skočite naprej, ne samo, da se osnovne ideje ne obdržijo v vaši glavi, ampak boste tudi vsebino kasnejših težje dojeli in se vam bo zdela bolj zapletena, kot je v resnici.

Ko greste skozi to knjigo, poskusite vsak primer, tako da lahko vidite, kako delujejo. Obstajajo tudi vaje na koncu večine poglavij, ki vam bodo pomagale izboljšati programsko znanje. Ne pozabite, da bolje, ko boste razumeli osnove, lažje bo razumeti bolj zapletene ideje kasneje.

Ko boste naleteli na težjo snov, je tule nekaj koristnih nasvetov:

1. Razbijte problem na manjše koščke. Poskušajte razumeti, kaj počne majhen del kode, ali razmišljajte o manjšem delu težje ideje (osredotočite se na majhen del kode, namesto da poskušate takoj razumeti celotno stvar).
2. Če to še vedno ne pomaga, je včasih najbolje, da problem za nekaj časa odložite. Prespite in se vrnite k njemu kakšen drug dan. To je dober način za reševanje številnih težav, še posebej je v pomoč računalniškim programerjem.

Kdo naj prebere to knjigo

Ta knjiga je za vse, ki jih zanima računalniško programiranje, bodisi otrok ali odrasla oseba, ki prihaja v stik s programiranjem prvič. Če želite izvedeti, kako napisati lasten program, namesto da bi samo uporabljali programe, ki so jih razvili drugi, potem je Python za otroke odličen način za začetek.

V naslednjih poglavjih boste našli informacije, ki vam bodo pomagale namestiti Python, zagnati lupino Python in opraviti osnovne izračune, izpisati besedilo na zaslonu in ustvariti sezname ter izvesti preprosto krmilne operacije z uporabo if stavkov in for zank (in izvedeti, kaj so if stavki in for zanke!). Naučili se boste ponovne uporabe kode s funkcijami, osnov razredov in objektov ter opisov nekaterih od številnih vgrajenih Python funkcij in modulov.

Našli boste poglavja tako za enostavno kot naprednejšo grafiko, kot tudi za uporabo modula tkinter za risanje na računalniški zaslon. Obstajajo vaje z različno zahtevnostjo na koncu številnih poglavij, ki bodo bralcem pomagale pri utrjevanju novega znanja z možnostjo pisanja lastnih programčkov.

Ko boste zgradili svoje temeljno programsko znanje, se boste naučili napisati lastne igre. Razvili boste dve grafični igri in se seznanili z odkrivanjem trkov, dogodki in različnimi animacijskimi tehnikami.

Večina primerov v tej knjigi uporablja Pythonovo lupino IDLE (Integrated DeveLopment Environment). IDLE ponuja sintaksno poudarjanje, kopiranje in lepljenje (podobno kot v drugih aplikacijah), urejevalno okno, kjer lahko shranite kodo za kasnejšo uporabo, kar pomeni, da IDLE deluje kot oboje: interaktivno okolje za eksperimentiranje in nekakšen urejevalnik besedil. Primeri delujejo prav tako s standardno konzola in drugimi urejevalniki besedil, toda IDLE sintaksno poudarjanje in nekoliko bolj uporabniku prijazno okolje lahko pomagajo pri razumevanju, zato vam prvo poglavje pokaže, kako ga nastavite.

Kaj je v tej knjigi

Tukaj je kratek pregled vsega, kar boste našli v vsakem poglavju.

Poglavje 1 je uvod v programiranje z navodili za prvo namestitev Pythona.

Poglavje 2 uvaja osnovne izračune in spremenljivke ter

Poglavje 3 opisuje nekatere osnovne tipe v Pythonu, na primer: niz, seznam in tuple.

Poglavje 4 je prviokus modula želva. Od osnovnega programiranja bomo skočili h gibanju želve (v obliki puščice) po zaslonu.

Poglavje 5 zajema spremembe pogojev in »če« izjave, in **poglavje 6** se premakne naprej na »for« zanke in »while« zanke.

Poglavje 7 je trenutek, ko začnemo uporabljati in ustvarjati funkcije in potem v **8. poglavju** spoznamo razrede in objekte. Pokrivamo dovolj osnovnih idej za podporo nekaterih programskih tehnik, ki jih bomo morali uporabiti v poglavju o razvoju iger pozneje v knjigi. Na tej točki se začnejo stvari nekoliko bolj zapletati.

Poglavje 9 gre skozi večino vgrajenih funkcij v Pythonu in **poglavje 10** se nadaljuje z nekaj moduli (v bistvu zaloga uporabnih funkcij), ki so privzeto nameščene v Pythonu.

Poglavje 11 se vrne na modul želva, ko poskušamo z bolj zapletenimi oblikami. **Poglavje 12** nadaljuje z uporabo modula tkinter za naprednejše ustvarjanje grafike.

V **poglavjih 13 in 14** ustvarimo našo prvo igro, "Odbij!", ki temelji na znanju, pridobljenem iz prejšnjih poglavij in v **poglavjih 15-18** ustvarimo še eno igro, "Mr. Stickman drvi k izhodu." V poglavjih z razvojem igre so stvari, kjer se lahko začnejo resne težave. Če vse odpove, prenesite kodo s podporne spletne strani (<https://nostarch.com/pythonforkids>), in primerjate svojo kodo s temi delujočimi primeri.

Spletna stran v podporo

Če med branjem ugotovite, da potrebujete pomoč, poskusite s spletnim mestom za podporo, <https://nostarch.com/pythonforkids>, kjer boste našli prenose za vse primere v tej knjigi in več vaj. Prav tako lahko najdete rešitve za vse vaje v knjigi, če pride do blokade ali če želite preveriti svoje delo.

Zabavajte se!

Ne pozabite, da je programiranje zabavno, ko se prebijate skozi to knjigo. Ne razmišljajte o tem kot o delu. Pomislite o programiranju kot o načinu ustvarjanja zabavnih iger ali aplikacij, ki jih lahko delite s prijatelji ali drugimi.

Učenje programiranja je čudovita miselna telovadba in rezultati so lahko zelo koristni. Najpomembneje pa je, da karkoli naredite, se pri tem zabavajte!

1. poglavje: Vse kače ne pičijo

Računalniški program je nabor navodil, ki jih izvaja računalnik, da izvede neko dejanje. To niso fizični deli računalnika, kot so žice, mikročipi, kartice, trdi disk in podobno, ampak skrita stvar, ki se izvaja na tej strojni opremi. Računalniški program, ki ga bomo običajno imenovali kar »program«, je sestavljen iz ukazov, ki tej neumni opremi povedo, kaj storiti. Programska oprema je zbirka računalniških programov.

Brez računalniških programov, bi skoraj vsaka naprava, ki jo dnevno uporabljate, prenehala delovati ali pa bi bila veliko manj uporabna, kot je sedaj. Računalniški programi, v taki ali drugačni obliki, nadzirajo ne samo vaš osebni računalnik ampak tudi video igre, mobilne telefone in enote GPS v avtomobilih. Programska oprema prav tako nadzoruje manj očitne predmete, kot so LCD televizorji in njihovi daljinski upravljalniki, pa tudi nekatere najnovejše radijske sprejemnike, DVD predvajalnike, pečice in nekatere hladilnike. Tudi avtomobilske motorje, semaforje, ulične svetilke, signalizacijo za vlake, elektronske reklamne panoje in dvigala nadzorujejo programi. Programi so podobno kot misli. Če bi bili brez misli, bi verjetno samo sedeli na tleh in strmeli predse. Vaša misel "vstani s tal" je navodilo ali ukaz, ki vašemu telesu pove, naj vstane. Na enak način računalniški programi računalnikom povejo, kaj storiti. Če veste, kako napisati računalniške programe, lahko naredite mnogo uporabnih stvari. Seveda morda ne boste mogli pisati programov za nadzor avtomobilov, semaforjev ali hladilnika (no, vsaj ne takoj), vendar lahko ustvarite spletne strani, napišete svoje igre ali celo naredite program za pomoč pri vaši domači nalogi.



Kot ljudje, tudi računalniki uporabljajo več jezikov za komuniciranje -v tem primeru programske jezike. Programiranje je preprosto povedano poseben način za pogovor z računalnikom - način za podajanje navodil, ki jih ljudje in računalniki dobro razumejo. Obstajajo programski jeziki poimenovani po ljudeh (kot Ada in Pascal), z uporabo preprostih akronimov (npr. BASIC in FORTRAN), pa tudi nekaj imen po televizijskih oddajah, kot je Python. Da, programski jezik Python je bil poimenovan po Letečem cirkusu Montyja Pythona.

Številne stvari v programskem jeziku Python so zelo primerne za začetnike. Najpomembnejše je, da ga zelo hitro lahko uporabite za pisanje preprostih in učinkovitih programov. Python ne uporablja zelo zapletenih simbolov kot drugi programski jeziki, zaradi česar ga je lažje brati in je veliko prijaznejši za začetnike.

Namestitev Pythona v operacijskem sistemu Windows 7 in višje

Če želite namestiti Python za Microsoft Windows 7 in novejše, prenesite različico Python za Windows, ki je 3.5 ali novejša: <https://www.python.org/downloads/windows/>.

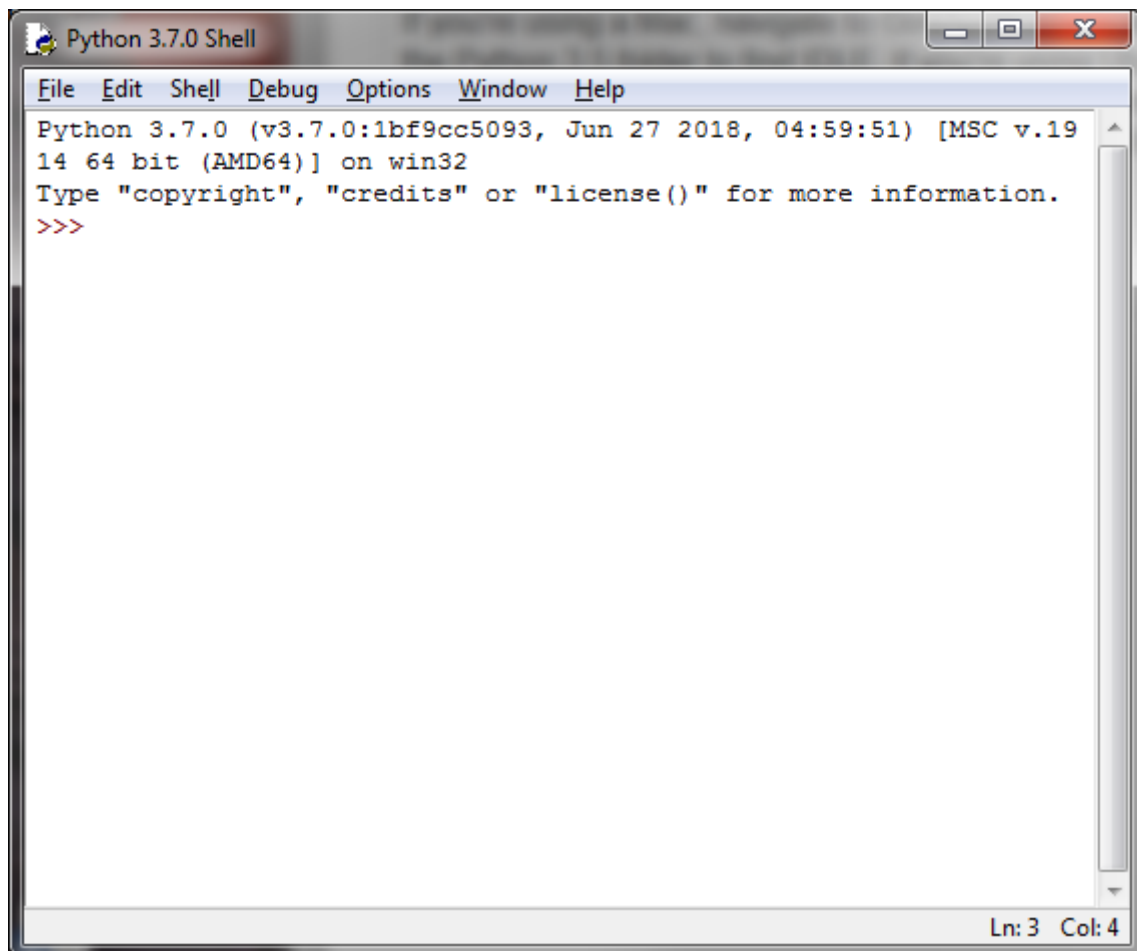
Ko ste prenesli datoteko za namestitev Pythona za Windows, jo poženite. Če vam ne ponudi zagona programa, odprite mapo Prenosi in dvokliknite datoteko. Sledite navodilom za namestitev na zaslonu. Python namestite na privzeto lokacijo, in sicer:

1. Kliknite Namesti zdaj.

2. Na vprašanje, ali naj program dopušča spremembe v računalniku izberite Next/Naprej.
3. Ko se konča namestitvev, kliknite Zapri, nato pa Python 3 mapo že lahko vidite v vašem Windows meniju Start.

Ko ste namestili Python

Zdaj, ko ste namestili Python, napišite prvi program v IDLE. Če uporabljate Windows, kliknite Windows Start in v iskalno polje vnesite idle ter izberite IDLE (Python 3.5, Python 3.7). Ko odprete IDLE, bi morali videti naslednjo sliko:



To je lupina Python, ki je del Pythonovega integriranega razvojnega okolja. Trije znaki večji od (>>>) se imenuje ukazni poziv. Vpišite nekaj ukazov na pozivno vrstico, začenši z naslednjim:

```
>>> print("Pozdravljen svet")
```

Poskrbite, da boste vključili dvojne narekovaje. Pritisnite enter na tipkovnici, ko končate s tipkanjem. Če ste vnesli ukaz pravilno, bi morali videti nekaj takega:

```
>>> print("Pozdravljen svet")
```

```
Pozdravljen svet
```

```
>>>
```

Poziv bi se moral znova prikazati, da je Pythonova lupina ponovno pripravljena sprejeti nove ukaze.

Čestitke! Pravkar ste ustvarili svoj prvi program v Pythonu. Beseda `print` je vrsta Python ukaza in se imenuje funkcija. Na zaslon izpiše vse, kar je v oklepajih. V bistvu ste dali računalniku navodilo za prikaz besed "Pozdravljen svet", ki ga vi in računalnik razumeta.



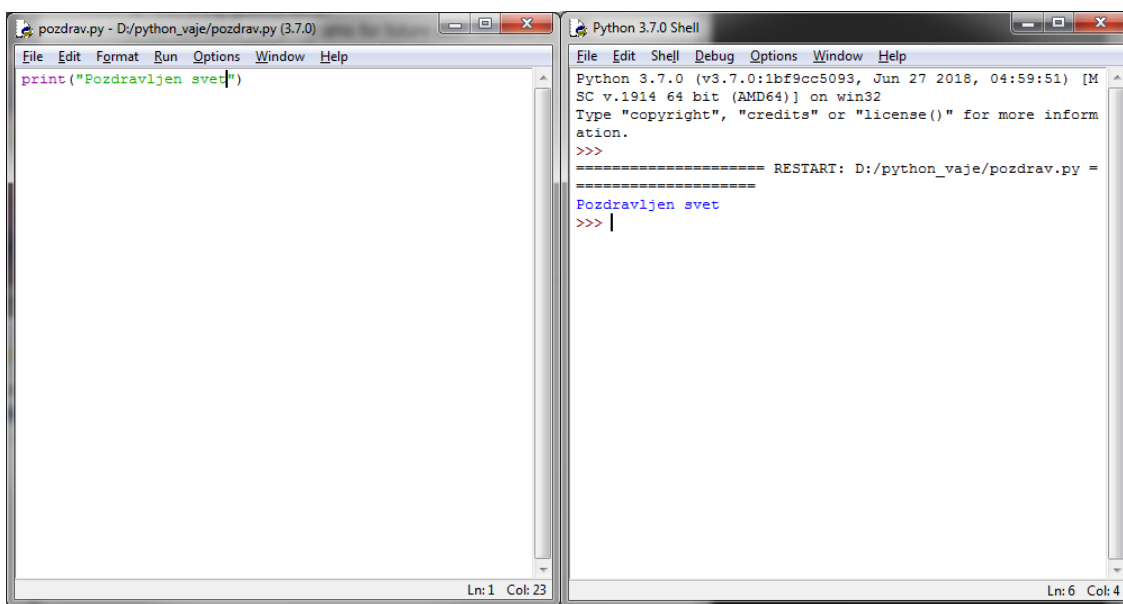
Shranjevanje Python programov

Programi Python ne bi bili zelo uporabni, če bi jih morali ponovno napisati vsakič, ko jih želite uporabiti. Morda kakšen kratek program, nikakor pa ne velik program, kot je urejevalnik besedil, ki lahko vsebuje milijone vrstic kode. Če bi vse natisnili, bi lahko imeli več kot 100.000 strani papirja. Zamislite si, da poskušate nesti tak ogromen kup papirja domov. Na srečo lahko shranimo naše programe za prihodnjo uporabo. Če želite shraniti nov program, odprite IDLE in izberite File, New Window. Pojavi se prazno okno z * Untitled * v menijski vrstici. V novo okno lupine vnesite naslednjo kodo:

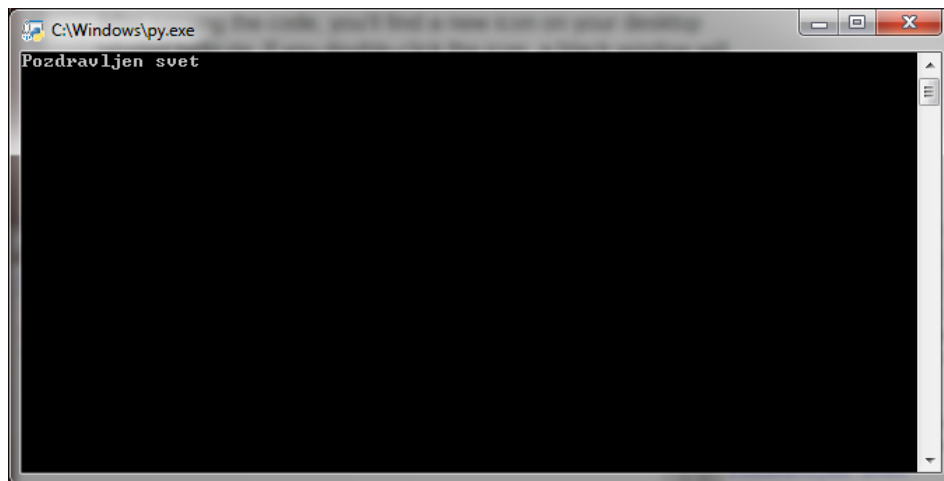
```
print("Pozdravljen svet")
```

Sedaj izberite File, Save.

Ko zahteva ime datoteke, vnesite `pozdrav.py` in shranite datoteko na namizje ali v svojo delovno mapo (npr: `python_vaje`). Potem izberite Run, Run Module.



Po zagonu kode boste na namizju ali v svoji mapi našli novo ikono označeno `pozdrav.py`. Če dvokliknete ikono, se bo na kratko pojavilo in izginilo črno okno. Kaj se je zgodilo? To je bilo okno ukazne vrstice Python (podobno kot lupina), izpisalo se je "Pozdravljen svet" in nato zaprlo. Tule lahko vidite, kaj bi videli, če bi imeli super hiter vid:



Kaj ste se naučili

V tem poglavju smo preprosto začeli z aplikacijo Pozdravljen svet – programom, ki je prvi za skoraj vse, ki se začnejo učiti računalniškega programiranja. V naslednjem poglavju bomo naredili še nekaj koristnih stvari z lupino Python.

2. poglavje: Računanje in spremenljivke

Zdaj, ko ste namestili Python in znate začeti delati, ste pripravljeni na naslednji korak. Začeli bomo z nekaj preprostimi izračuni in nato gremo na spremenljivke. Spremenljivke so način shranjevanja stvari v računalniškem programu in pomagajo vam pisati uporabne programe.

Računanje s Pythonom

Običajno, ko vas vprašajo za produkt dveh števil, kot je 8×3.57 , bi uporabili kalkulator ali svinčnik in papir. No, kaj pa, če bi za izračun uporabili Python? Poskusimo. Zaženite lupino Python tako, da dvokliknete ikono IDLE namizje. V pozivu vnesite to enačbo:

```
>>> 8 * 3.57
28.56
```

Upoštevajte, da pri vnosu izračuna množenja v Pythonu, namesto znaka za množenje (\times) uporabite zvezdico (*). Kaj pa, če poskusimo enačbo, ki je malo bolj uporabna? Recimo, da kopljete na vrtu in odkrijete vrečko 20 zlatih kovancev. Naslednji dan, se izmuznete v klet in vstavite kovance v parno napravo, ki jo je izumil vaš dedek, za kopiranje raznih stvari (na srečo lahko vstavite ravno 20 zlatnikov). Zaslišite čarobni pisk in ropotanje in nekaj ur kasneje se vsuje še 10 bleščočih kovancev.

Koliko kovancev bi imeli v vaši skrinji za zaklad, če bi to naredil vsak dan v letu? Na papirju so lahko enačbe videti takole:

$$10 \times 365 = 3650$$

$$20 + 3650 = 3670$$

Seveda, enostavno je narediti te izračune na kalkulator ali na papirju, vendar lahko vse te izračune opravimo tudi s Pythonom. Najprej pomnožimo 10 kovancev s 365 dni v letu, da dobimo 3650. Nato dodamo prvotnih 20 kovancev, da dobimo 3670.

```
>>> 10 * 365
3650
>>> 20 + 3650
3670
```

Kaj pa, če bi sraka opazila bleščeče kovance v vaši spalnici in bi vsak teden priletela ter ukradla tri kovance?

Koliko kovancev bi ostalo konec leta? Tukaj je izračun:

```
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Najprej pomnožimo 3 kovance z 52 tedni v letu. Rezultat je 156. To število odštejemo od naših skupnih kovancev (3670), kar nam pove, da bomo ob koncu leta imeli 3514 kovancev.

To je zelo preprost program. Naučili se bomo, kako razširiti te ideje, da napišete programe, ki so še bolj uporabni.

Operatorji v Pythonu

V Pythonovi lupini lahko med drugimi matematičnimi operacijami tudi množite, seštevate, odštevate in delite. Osnovni simboli, ki jih Python uporablja za izvajanje matematičnih operacij imenujemo operatorji. Nekaj osnovnih je navedenih v tabeli.

simbol	operacija
+	Seštevanje
-	Odštevanje
*	Množenje
/	Deljenje



Naprej obrnjena poševnica (/) se uporablja za deljenje saj je podobna ulomkovi črti, ki se uporablja za ulomke. Na primer, če imate 100 piratov in 20 velikih sodov in bi želeli izračunati koliko piratov lahko skrijete v vsakem sodu, bi lahko razdelili 100 piratov na 20 sodov (100 : 20). V lupino bi vnesli 100/20. Samo zapomnite si, da je naprej obrnjena poševnica tista, katere vrh pade na desno.

Vrstni red operacij

Za vrstni red operacij v programskem jeziku uporabljamo oklepaje. Operacija je vse, kjer se uporablja operator. Množenje in deljenje imata prednost pred seštevanjem in odštevanjem, kar pomeni, da se izvedeta prej. Z drugimi besedami, če vnesete enačbo v Python, se množenje ali deljenje izvaja pred seštevanjem ali odštevanjem. Na primer, v naslednji enačbi števili 30 in 20 najprej pomnožimo, številka 5 pa se doda njenemu produktu.

```
>>> 5 + 30 * 20
605
```

To enačbo lahko povemo na naslednji način: "pomnožite 30 z 20 in nato rezultatu dodajte 5. "Rezultat je 605. Vrstni red operacij lahko spremenimo z dodajanjem oklepajev, na primer:

```
>>> (5 + 30) * 20
700
```

Rezultat te enačbe je 700 (ne 605), ker oklepaj povej Pythonu, da najprej izračuna vrednost v oklepaju in potem izvede še operacijo izven oklepajev. Tu lahko rečemo: "seštejte 5 in 30, nato pa rezultat pomnožite z 20."

Oklepaji so lahko vgnjezdeni, kar pomeni, da so lahko oklepaji znotraj oklepajev, na primer:

```
>>> ((5 + 30) * 20) / 10
70.0
```

V tem primeru Python izračuna najprej najgloblji oklepaj, potem zunanje oklepaje in nato še končni operator deljenja.

Z drugimi besedami, ta enačba pravi: "seštejte 5 in 30, nato pomnožite rezultat z 20 in delite ta rezultat z 10."

Tukaj se bo zgodilo:

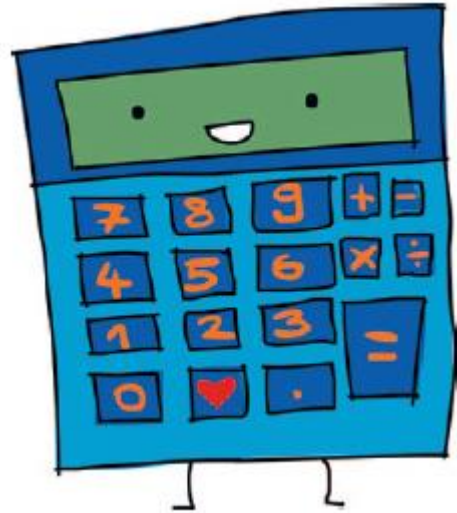
- Seštevanje 5 s 30 je 35.
- Množenje 35 z 20 je 700.
- Deljenje 700 z 10 je končni odgovor 70.

Če nismo uporabili oklepajev, bi bil rezultat rahlo drugačen:

```
>>> 5 + 30 * 20/10
65.0
```

V tem primeru se 30 najprej pomnoži z 20 (je 600), potem se 600 deli z 10 (je 60). Končno, dodamo še 5 in rezultat je 65.

Ne pozabite, da imata množenje in deljenje vedno prednost pred seštevanjem in odštevanjem, razen če se za nadzor vrstnega reda operacij uporabljajo oklepaji.



Spremenljivke so kot oznake

Beseda spremenljivka v programiranju opisuje prostor za shranjevanje informacij kot so številke, besedilo, sezname števil in besedila itd. Drug način gledanja na spremenljivko je, da je kot oznaka za nekaj. Na primer, za ustvarjanje spremenljivke, imenovane fred, uporabljamo enačaj (=) in nato Pythonu povemo, da bo to oznaka za nekaj. Tu ustvarimo spremenljivko fred in povemo Pythonu, da označuje številko 100 (to še ne pomeni, da kakšna druga spremenljivka ne more imeti enake vrednosti):

```
>>> fred = 100
```

Če želite izvedeti, kakšno vrednost označuje spremenljivka, vnesite v lupino print in v oklepajih navedite ime spremenljivke, kot je ta:

```
>>> print(fred)
100
```

Pythonu lahko tudi povemo, da spremenimo spremenljivko fred, tako da označuje nekaj drugega. Primer: na primer, kako spremeniti fred v številko 200:

```
>>> fred = 200
>>> print(fred)
200
```

V prvi vrstici rečemo, da fred označuje številko 200. V drugi vrstici vprašamo, kaj označuje fred, le da potrdimo spremembo. Python izpiše rezultat v zadnji vrstici. Uporabljamo lahko tudi več kot eno oznako (več kot eno spremenljivko) za isto vrednost:

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```


V tem primeru povemo Pythonu, da želimo, da ime (ali spremenljivka) john označuje isto stvar kot fred z uporabo enačaja med john in fred. Seveda, fred verjetno ni zelo koristno ime za spremenljivko, ker najverjetneje ne pove ničesar o tem, za kaj se spremenljivka uporablja. Pokličimo našo spremenljivko `stevilo_kovancev` namesto `fred`, takole:

```
>>> stevilo_kovancev = 200
>>> print(stevilo_kovancev)
200
```

To pojasnjuje, da govorimo o 200 kovancih. Imena spremenljivk lahko sestavljajo črke, številke in podčrtaj (`_`), vendar se ne morejo začeti s številko. Lahko uporabite vse od posameznih črk (npr. `a`) do dolgih stavkov za imena spremenljivk. (Spremenljivka ne sme vsebovati presledka, zato uporabite podčrtaj, da ločite besede.) Včasih, če delate nekaj hitro, je kratko ime spremenljivke najboljše. Ime, ki ga izberete za spremenljivko je odvisno od tega, kako pomembno je, da ime spremenljivke nekaj označuje (pomeni). Zdaj, ko veste, kako ustvariti spremenljivke, pogledjmo, kako jih uporabite.

Uporaba spremenljivk

Se spomnite naše enačbo za ugotovitev, koliko kovancev bi imeli ob koncu leta, če bi lahko čarobno ustvarili nove kovance z norim izumom vašega dedka v kleti? Imamo to enačbo:

```
>>> 20 + 10 * 365
3670
>>> 3 * 52
156
>>> 3670 - 156
3514
```



To lahko spremenimo v eno vrstico kode:

```
>>> 20 + 10 * 365 - 3 * 52
3514
```

Kaj pa, če bi številke spremenili v spremenljivke?

Poskusite vnesti naslednje:

```
>>> najdeni_kovanci = 20
>>> carobni_kovanci = 10
>>> ukradeni_kovanci = 3
```

Ti vnosi so ustvarili spremenljivke: `najdeni_kovanci`, `carobni_kovanci` in `ukradeni_kovanci`.

Sedaj lahko enačbo ponovno vnesemo tako:

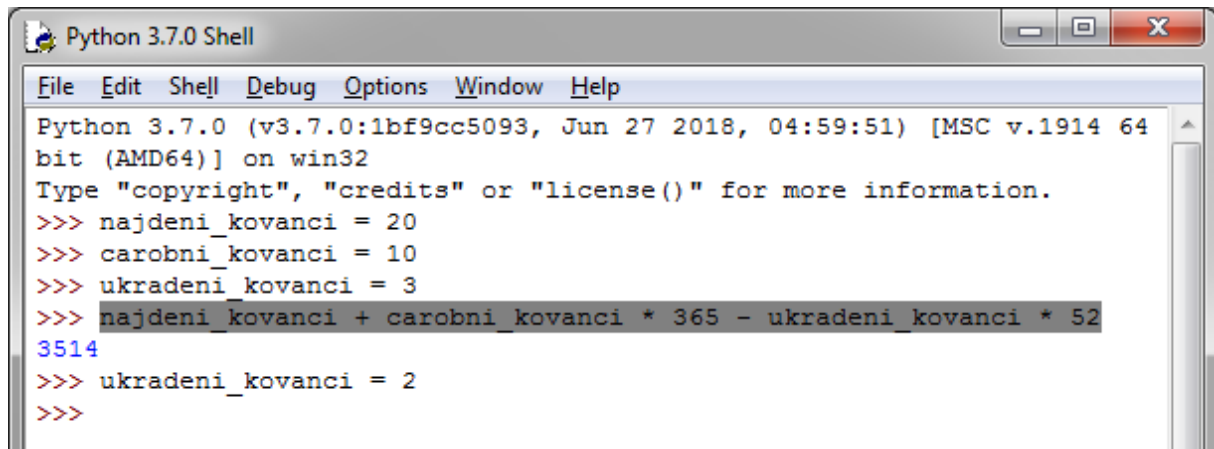
```
>>> najdeni_kovanci + carobni_kovanci * 365 - ukradeni_kovanci * 52
3514
```

Vidite, da nam daje enak odgovor. Pa kaj, boste rekli. Aha, ampak tukaj je še čarobnost spremenljivk. Kaj, če bi na okno postavili strašilo in bi sraka uspela ukrasti le dva kovanca namesto treh? Ko uporabimo spremenljivko, jo lahko enostavno spremenimo, da vsebuje novo vrednost, ki se bo uporabila povsod v enačbi. Lahko spremenite spremenljivko `ukradeni_kovanci` na vrednost 2, tako da vnesete to:

```
>>> ukradeni_kovanci = 2
```

Nato lahko kopiramo in prilepimo enačbo za ponovni izračun:

1. Izberite besedilo, ki ga želite kopirati, tako da kliknete z miško in povlečete od začetka do konca vrstice, kot je prikazano tukaj:



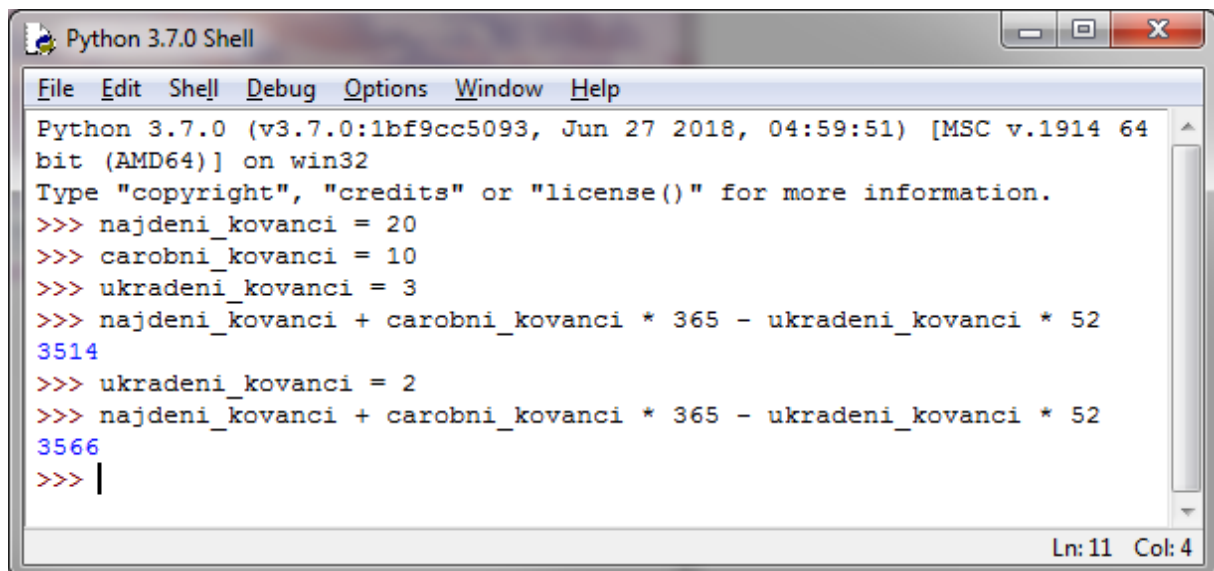
```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> najdeni_kovanci = 20
>>> carobni_kovanci = 10
>>> ukradeni_kovanci = 3
>>> najdeni_kovanci + carobni_kovanci * 365 - ukradeni_kovanci * 52
3514
>>> ukradeni_kovanci = 2
>>>
```

2. Držite tipko Ctrl in pritisnite C, da kopirate izbrano besedilo. (To boste videli kot ctrl-C od zdaj naprej.)

3. Kliknite zadnjo pozivno vrstico (po ukradeni_kovanci = 2).

4. Držite tipko Ctrl in pritisnite V, da prilepite izbrano besedilo. (Od zdaj naprej boste to videli kot ctrl-V.)

5. Pritisnite enter, da vidite nov rezultat:



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> najdeni_kovanci = 20
>>> carobni_kovanci = 10
>>> ukradeni_kovanci = 3
>>> najdeni_kovanci + carobni_kovanci * 365 - ukradeni_kovanci * 52
3514
>>> ukradeni_kovanci = 2
>>> najdeni_kovanci + carobni_kovanci * 365 - ukradeni_kovanci * 52
3566
>>> |
Ln: 11 Col: 4
```

Ali ni to veliko lažje kot ponovitev celotne enačbe? Seveda je.

Poskusite spremeniti druge spremenljivke in nato kopirati (ctrl-C) in prilepite (ctrl-V) izračun, da vidite učinek svoje spremembe. Na primer, če stresete napravo vašega dedka v pravem trenutku in vsakič izdela dodatne 3 kovancev, boste ugotovili, da bo na koncu 4661 kovancev letno:

```
>>> carobni_kovanci = 13
>>> najdeni_kovanci + carobni_kovanci * 365 - ukradeni_kovanci * 52
4661
```

Seveda, uporaba spremenljivk za tako preprosto enačbo, kot je ta, je le delno koristna. Še vedno nismo izkusili resnične koristnosti. Za zdaj, si samo zapomnite, da so spremenljivke način označevanja stvari tako, da jih lahko kasneje uporabite.

Kaj ste se naučili

V tem poglavju ste se naučili uporabljati enostavne enačbe, Python operatorje in kako uporabljati oklepaje za nadzor vrstnega reda operacij (vrstni red, v katerem Python vrednoti dele enačbe). Nato smo ustvarili spremenljivke za označevanje vrednosti in jih uporabili v naših izračunih.

3. poglavje: String, list, tuple, map

V 2. poglavju smo naredili nekaj osnovnih izračunov s Pythonom in izvedeli ste nekaj o spremenljivkah. V tem poglavju bomo delali z nekaterimi drugimi elementi: nizi (string), seznamami (list), fiksnimi seznamami (tuple) in slovarji (map). Uporabili boste nize za prikaz sporočil v svojih programih (npr "Pripravite se" in "Game Over" v igri). Odkrili boste tudi, kako se sezname in slovarji uporabljajo za shranjevanje zbirk stvari.

Nizi (Strings)

Pri programiranju besedilo običajno imenujemo niz (string). Pomislite na niz kot zbirko črk in izraz bo smiselni. Vse črke, številke in simboli v tej knjigi bi lahko bili niz, prav tako vaše ime in naslov. Pravzaprav, prvi Python program, ki smo ga ustvarili v Poglavju 1 je uporabil niz: "Pozdravljen svet."



Ustvarjanje nizov

V Pythonu ustvarjamo niz z dodajanjem narekovajev okoli besedila, ker morajo programski jeziki razlikovati med različnimi vrstami vrednosti. (Mi moramo računalniku povedati ali je vrednost številka, niz ali kaj drugega.) Na primer, lahko bi vzeli našo spremenljivko fred iz poglavja 2 in jo uporabili za označevanje niza:

```
fred = "Zakaj imajo gorile velike nosnice? Ker imajo velike prste!"
```

Da bi videli, kaj je v njej, jo izpišemo `print(fred)`:

```
>>> print(fred)
Zakaj imajo gorile velike nosnice? Ker imajo velike prste!
```

Za ustvarjanje niza lahko uporabite tudi enojne narekovaje:

```
>>> fred = 'Kaj je rožnato in puhasto? Rožnati puh!'
>>> print(fred)
Kaj je rožnato in puhasto? Rožnati puh!
```

Če poskusite vnesti več kot eno vrstico besedila v vaš niz, ki uporablja samo enojni (') ali dvojni narekovaj (") ali če začnete z eno vrsto narekovaja in končate z drugo, boste prejeli sporočilo o napaki. Vnesite na primer naslednjo vrstico:

```
>>> fred = "Kako dinozavri plačujejo svoje račune?"
```

Izpiše se vam:

```
SyntaxError: EOL while scanning string literal
```

To je sporočilo o napaki, ki sporoča napako v sintaksi (nekaj je narobe napisano), ker niste upoštevali pravil za konec niza z enim samim ali dvojnimi narekovajem. Sintaksa pomeni ureditev in vrstni red besed v stavku ali v tem primeru ureditev in vrstni red besed in simbolov v programu. Torej

SyntaxError pomeni, da ste nekaj naredili, česar Python ni pričakoval ali pa je Python pričakoval nekaj, kar ste izpustili. EOL (End Of Line) pomeni konec vrstice, preostali del sporočila o napaki pove, da je Python prišel do konca vrstice in ni našel dvojnega narekovaja za zaključek niza. Če želite v nizu uporabiti več vrstic besedila (večvrstični niz), uporabite tri enojne narekovaje (""") in nato pritisnite enter med vrsticami, kot je tu:

```
>>> fred = '''Kako dinozavri plačujejo svoje račune?
S tiranozavrskimi čeki! '''
```

Sedaj pa izpišimo vsebino fred-a, da vidimo, ali je to delovalo:

```
>>> print(fred)
Kako dinozavri plačujejo svoje račune?
S tiranozavrskimi čeki!
```

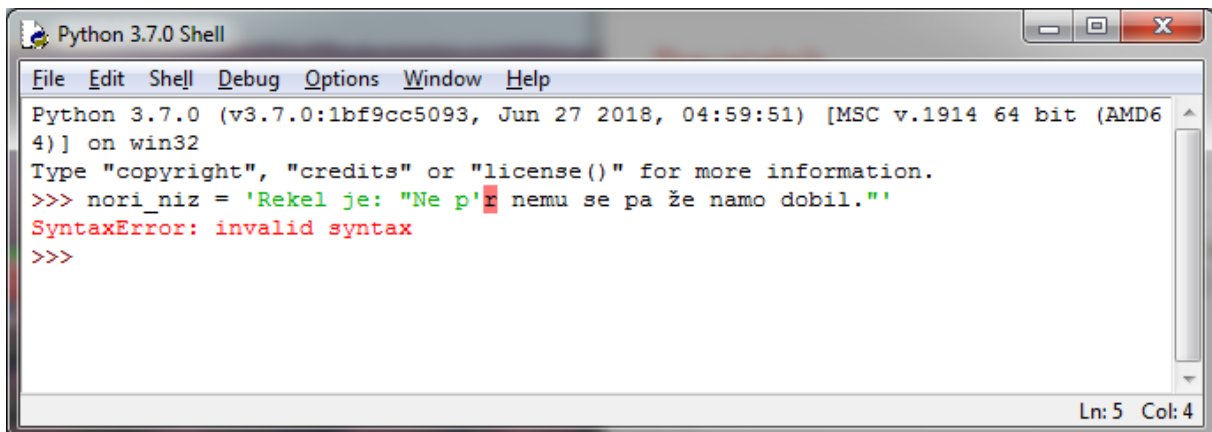
Reševanje težav z nizi

Poglejmo sedaj en nori primer niza, ki povzroča sporočilo o napaki:

```
>>> nori_niz = 'Rekel je: "Ne p'r nemu se pa že namo dobil.''
```

SyntaxError: invalid syntax

V prvi vrstici skušamo ustvariti niz (definiran kot spremenljivka nori_niz), ki ga obdajajo enojni narekovaji, vsebuje pa tudi mešanica premega govora in narečnega govora. Kakšen nered! Ne pozabite, da Python ni tako pameten kot človek, da je vse, kar vidi, niz, ki vsebuje. Rekel je: "Ne p'r nemu se pa že namo dobil". Ko Python vidi narekovaj (enojni ali dvojni), pričakuje, da se bo niz končal za naslednjim enakim narekovajem (bodisi enojnim ali dvojnimi). V tem primeru je začetek niza enojni narekovaj in niz se konča po Ne p'. IDLE izpostavlja točko, kjer so stvari napačne:

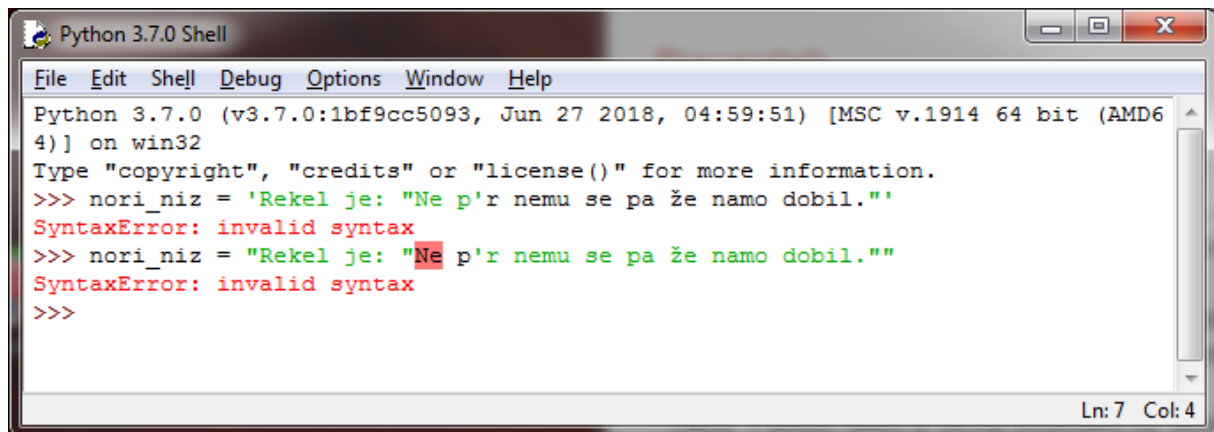


```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> nori_niz = 'Rekel je: "Ne p'r nemu se pa že namo dobil.'"
SyntaxError: invalid syntax
>>>
```

Zadnja vrstica IDLE nam pove, kakšna je bila napaka – v tem primeru je sintaktična napaka. Uporaba dvojnih, namesto enojnih narekovajev, še vedno povzroča napako:

```
>>> nori_niz = "Rekel je: "Ne p'r nemu se pa že namo dobil""
SyntaxError: invalid syntax
```

Tukaj Python vidi niz, omejen z dvojnimi narekovaji, ki vsebujejo črke, ki jih je rekel (in presledke). Vse, kar sledi temu, povzroči napako:



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> nori_niz = 'Rekel je: "Ne p'r nemu se pa že namo dobil."'
SyntaxError: invalid syntax
>>> nori_niz = "Rekel je: "Ne p'r nemu se pa že namo dobil.""
SyntaxError: invalid syntax
>>>
```

To je zato, ker s Pythonove perspektive vse te dodatne stvari ne bi smele biti tam. Python poišče naslednje ujemanje narekovajev in ne ve, kaj želite narediti s tem kar sledi temu narekovaju v isti vrstici.

Rešitev tega problema je večvrstični niz, ki smo se ga naučili prej, z uporabo treh enojnih narekovajev (""), kar nam omogoča, da združimo dvojne in enojne narekovaje v našem nizu brez povzročanja napake. Dejansko, če uporabimo tri enojne narekovaje lahko damo katero koli kombinacijo enojnih in dvojnih narekovajev znotraj niza (dokler ne poskušamo postavite tri enojne narekovaje). Takole izgleda naš niz brez napak:

```
nori_niz = '''Rekel je: "Ne p'r nemu se pa že namo dobil."'''
```

Toda počakajte, obstaja še nekaj. Če res želite uporabljati enojno ali dvojno narekovaje, da obkrožajo niz v Pythonu, namesto treh enojnih narekovajev, lahko dodate poševnico (\) pred vsakim narekovajem v nizu. To se imenuje escape znak. Na ta način povemo Pythonu, "Ja, vem, da imam citate v mojem nizu in hočem, da jih ignoriraš, dokler ne najdeš končnega narekovaja." Escape znaki lahko otežijo branje, zato je verjetno bolje uporabiti večvrstične nize. Kljub temu se boste morda srečali z deli kode, ki uporablja escape znake, zato je dobro vedeti, zakaj so poševnice tam. Tukaj je nekaj primerov z escape znaki:

```
(1) >>> niz_z_enojnimi_narekovaji = 'Rekel je: "Ne p\'r nemu se pa že namo dobil."'
(2) >>> niz_z_dvojnimi_narekovaji = "Rekel je: \"Ne p'r nemu se pa že namo dobil."
>>> print(niz_z_enojnimi_narekovaji)
Rekel je: "Ne p'r nemu se pa že namo dobil."
>>> print(niz_z_dvojnimi_narekovaji)
Rekel je: "Ne p'r nemu se pa že namo dobil."
```

Prvič, pri (1) ustvarimo niz z enojnimi narekovaji z uporabo obrnjene poševnice (backslash) pred enojnimi narekovaji v tem nizu. Pri (2), ustvarimo niz z dvojnimi narekovaji in uporabimo obrnjeno poševnico pred temi narekovaji v nizu. V vrsticah, ki sledijo, izpišemo spremenljivke, ki smo jih pravkar ustvarili. Kot vidite se obrnjena poševnica ne izpisuje.

Vsebovane vrednosti v nizih

Če želite prikazati sporočilo z vsebino spremenljivke, lahko vrednosti v nizu vnesete z uporabo %, kar označuje vrednost, ki jo želite dodati pozneje. (Vdelava vrednosti, tudi imenovan kot zamenjava nizov, je v programerskem jeziku "vstavljanje" vrednosti. ") Na primer, če želite v Pythonu izračunati

ali shraniti število točk, ki ste jih zabeležili v igri, in nato dodati v stavku kot "Osvojil si točk", uporabite %s v stavku namesto vrednost in nato te vrednosti navedite:

```
>>> rezultat = 1000
>>> sporocilo = 'Osvojil si %s točk.'
>>> print(sporocilo % rezultat)
Osvojil si 1000 točk.
```

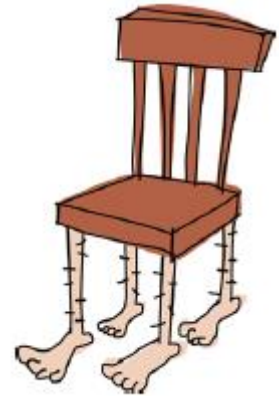
Tu ustvarimo spremenljivko rezultat z vrednostjo 1000 in spremenljivko sporocilo z nizom, ki vsebuje besede "Osvojil si %s točk.", kjer je %s prostor za število točk. V naslednji vrstici, pokličemo izpis (sporocilo) s simbolom %, da povemo Pythonu naj zamenja %s z vrednostjo, shranjeno v spremenljivki rezultat. Izpiše se »Osvojil si 1000 točk.« Ni nam treba uporabiti spremenljivke za vrednost. Lahko bi isti primer naredili drugače in le uporabili print(sporocilo % 1000). Prav tako lahko uporabimo različne vrednosti za %s, kot v tem primeru:

```
>>> besedilko_vica = '%s: naprava za iskanje pohištva v temi'
>>> deltelesa1 = 'Koleno'
>>> deltelesa2 = 'Golenica'
>>> print(besedilko_vica % deltelesa1)
Koleno: naprava za iskanje pohištva v temi
>>> natisni(besedilko_vica % deltelesa2)
Golenica: naprava za iskanje pohištva v temi
```

Tu ustvarimo tri spremenljivke. Prva, besedilo_vica, vključuje niz z oznako %s. Ostali spremenljivki sta deltelesa1 in deltelesa2. Spremenljivko besedilo_vica lahko izpišemo in enkrat uporabimo operator %, da ga zamenjamo z vsebina spremenljivke deltelesa1 in drugič deltelesa2 ter ustvarimo dve različni sporočili.

V nizu lahko uporabite tudi več mest za nadomeščanje niza, na primer:

```
>>> stevila = 'Kaj je številka %s rekla številki %s?
Lep pas!'
>>> print(stevila %(0, 8))
Kaj je številka 0 rekla številki 8? Lep pas!
```



Če uporabljate več kot eno mesto nadomeščanja znakov, obvezno postavite nadomestne vrednosti v oklepaju, kot je prikazano v primeru. Vrstni red vrednosti je vrstni red, v katerem bodo uporabljeni v nizu.

Množenje nizov

Kaj je 10 pomnoženo s 5? Odgovor je seveda 50. Kaj pa je 10 pomnoženo z a? Tukaj je odgovor Pythona:

```
>>> print(10 * 'a')
aaaaaaaaaa
```

Python programerji bi lahko to uporabili za tvorjenje nizov z določenim številom presledkov pri prikazovanju sporočil. Kaj pa, če bi v lupini izpisali pismo (izberite File, New Window in vnesite naslednjo kodo):

```
presledki = ' ' * 25
print('%s Ljubljana, 12.6.2018' % presledki)
print()
print()
```

```

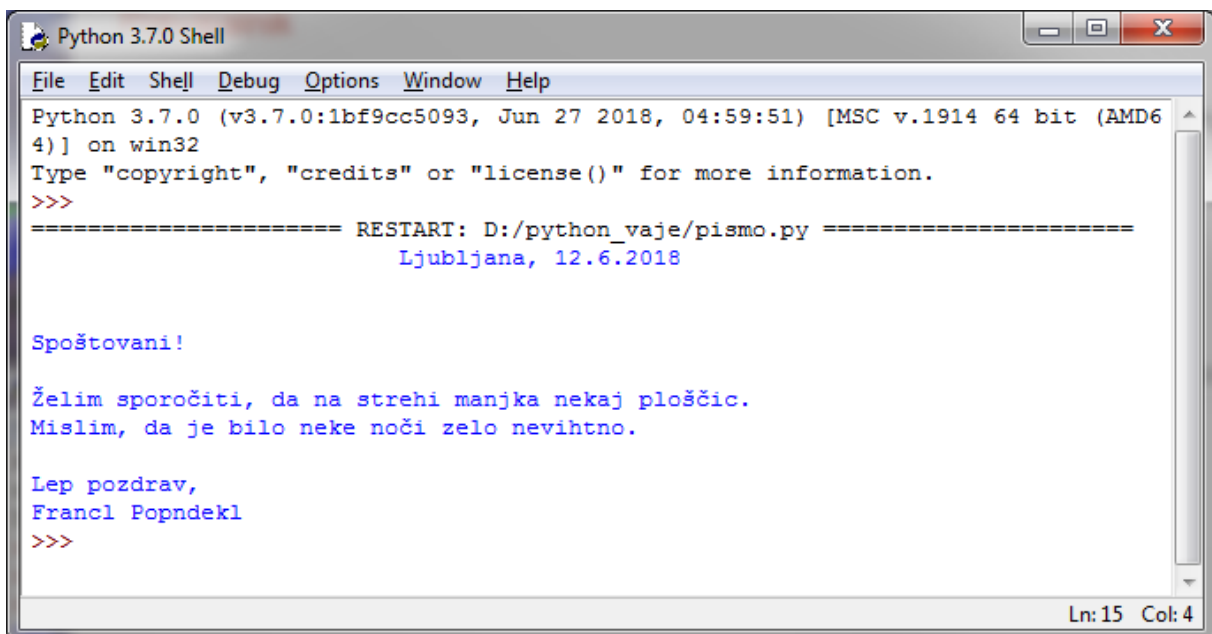
print("Spoštovani!")
print()
print('Želim sporočiti, da na strehi manjka nekaj ploščic.')
print('Mislim, da je bilo neke noči zelo nevihtno.')
print()
print('Lep pozdrav,')
print('Francl Popndekl')

```

Ko vnesete kodo v okno ukazne lupine, izberite File, Save As. Poimenujte svojo datoteko pismo.py. Nato lahko zaženete kodo (kot smo že storili prej) z izbiro Run, Run Module.

Od sedaj naprej, ko vidite Save As: imedatoteke.py nad neko kodo, boste vedeli, da morate izbrati File, New Window, vnesti kodo v okno, ki se pojavi, in jo shraniti, kot smo to storili v tem primeru.

V prvi vrstici tega primera ustvarjamo spremenljive presledke s pomnožitvijo znaka presledek s 25. Nato uporabimo to spremenljivko v naslednji vrstici, da se besedilo izpiše desno. Rezultat tega izpisa lahko vidite spodaj:



```

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/python_vaje/pismo.py =====
                Ljubljana, 12.6.2018

Spoštovani!

Želim sporočiti, da na strehi manjka nekaj ploščic.
Mislim, da je bilo neke noči zelo nevihtno.

Lep pozdrav,
Francl Popndekl
>>>
Ln:15 Col:4

```

Poleg tega, da lahko uporabimo množenje za poravnavo, ga lahko uporabite tudi za zapolnitev zaslona z nadležnimi sporočili. Poskusite ta primer tole:

```
>>> print(1000 * 'car')
```

Seznami so močnejši od nizov

"Pajkove noge, žabji prst, kuščarjevo oko, krilo netopirja, polžja slina in kačji prhljaj" niso ravno običajen seznam nakupov (razen, če ste čarovnik), vendar jo bomo uporabili kot prvi primer razlike med nizi in seznamami.

Ta seznam elementov lahko shranimo v spremenljivko carovnik_seznam z uporabo niza na takle način:

```

>>> carovnik_seznam = 'pajkove noge, žabji prst, kuščarjevo oko, krilo
netopirja, polžja slina, kačji prhljaj'
>>> print(carovnik_seznam)
pajkove noge, žabji prst, kuščarjevo oko, krilo netopirja, polžja slina,
kačji prhljaj

```


Lahko pa tudi ustvarimo seznam, nekoliko čaroben Pythonov objekt, ki ga lahko upravljamo. Na takle način jih lahko zapišemo kot seznam:

```
>>> carovnik_seznam = ['pajkove noge', 'žabji prst',
                        'kuščarjevo oko', 'krilo netopirja', 'polžja slina',
                        'kačji prhljaj']
>>> print(carovnik_seznam)
['pajkove noge', 'žabji prst', 'kuščarjevo oko',
 'krilo netopirja', 'polžja slina', 'kačji prhljaj']
```



Ustvarjanje seznama zahteva več tipkanja kot ustvarjanje niza, vendar je seznam bolj uporaben kot niz, ker lahko z njim upravljamo - manipuliramo. Na primer, natisnemo lahko tretji element iz seznama `carovnik_seznam` (kuščarjevo oko) z vnosom njegovega položaja v seznamu (imenovan indeks) med oglatima oklepajema (`[]`), tako:

```
>>> print(carovnik_seznam[2])
kuščarjevo oko
```

Kaj? Ali ni tretji v seznamu? Ja, ampak sezname se začnejo indeksnim položajem 0, tako da je prvi element na seznamu 0, drugi 1 in tretji je 2. To morda ne bo imelo veliko smisla za ljudi, ima ga pa za računalnike.

Prav tako lahko spremenimo predmet v seznamu veliko lažje kot v nizu. Morda namesto kuščarjevega očesa rabimo polžev jezik. Takole bi to storili z našim seznamom:

```
>>> carovnik_seznam[2] = 'polžji jezik'
>>> print(carovnik_seznam)
['pajkove noge', 'žabji prst', 'polžji jezik', 'krilo netopirja', 'polžja
slina', 'kačji prhljaj']
```

To postavlja 2. element seznama na novo vrednost: polžji jezik.

Druga možnost je, da pokažemo le del seznama. To počnemo z dvopičjem (`:`) v oglatih oklepajih. Na primer, vnesite naslednje za ogled tretjega do petega elementa v seznamu (sijajni spisec sestavin za ljubki sendvič):

```
>>> print(carovnik_seznam[2: 5])
['polžji jezik', 'krilo netopirja', 'polžja slina']
```

Če napišemo `[2: 5]` je enako kot bi rekli: "prikaži elemente od indeksa 2 do (vendar ne vključno) indeksa 5" ali z drugimi besedami, elemente 2, 3 in 4. Sezname se lahko uporabljajo za shranjevanje vseh vrst podatkov, kot so številke:

```
>>> nekaj_stevil = [1, 2, 5, 10, 20]
```

Lahko imamo tudi nize:

```
>>> nekaj_nizov = ["which", "witch", "is", "which"]
```

Morda mešanica števil in nizov:

```
>>> stevila_in_nizi = ['Why', 'was', 6, 'affraid', 'of', 7, "because", 7,
8, 9]
>>> print(stevila_in_nizi)
```

```
['Why', 'was', 6, 'affraid', 'of', 7, "because", 7, 8, 9]
(Zakaj, se je, 6, bala, 7, ker je, 7, pojedla, 9)
```

Seznami lahko celo shranjujejo druge sezname:

```
>>> stevilke = [1, 2, 3, 4]
>>> nizi = ['Brcnil', 'sem', 'se', 'v', 'prst', 'in', 'sedaj', 'me',
'boli']
>>> mojseznam = [stevilke, nizi]
>>> print(mojseznam)
[[1, 2, 3, 4], ['Brcnil', 'sem', 'se', 'v', 'prst', 'in', 'sedaj', 'me',
'boli']]
```

Pri primeru seznama znotraj seznama gre za tri spremenljivke: stevilke s štirimi številkami, nizi z devetimi nizi in mojseznam z le dvema elementoma: stevilke in nizi. Tretji seznam (mojseznam) ima le dva elementa, ker je to seznam imen spremenljivk, ne pa vsebin spremenljivk.

Dodajanje elementov v seznam

Če želite dodati elemente na seznam, uporabimo funkcijo `append`. Funkcija je skupina ukazov, ki Pythonu pove, da naj nekaj naredi. V tem primeru `append` doda element na konec seznama.

Na primer, če želite na seznam dodati medvedje bruhanje (prepričan sem, da takšen stvar obstaja) za nakupovanje čarovnika, naredite to:

```
>>> carovnik_seznam.append('medvedje bruhanje')
>>> print(carovnik_seznam)
['pajkove noge', 'žabji prst', 'polžji jezik', 'krilo netopirja', 'polžja
slina', 'kačji prhljaj', 'medvedje bruhanje']
```

V čarovnikov seznam lahko še naprej dodajate več čarobnih predmetov na enak način:

```
>>> carovnik_seznam.append('mandragora')
>>> carovnik_seznam.append('ricinus')
>>> carovnik_seznam.append('močvirski plin')
```

Čarovnikov seznam je zdaj videti takole:

```
>>> print(wizard_list)
['pajkove noge', 'žabji prst', 'polžji jezik', 'krilo netopirja', 'polžja
slina', 'kačji prhljaj', 'medvedje bruhanje', 'mandragora', 'ricinus', 'močvirski plin']
```



Čarovnik je nedvomno pripravljen delati resne čarvnije!

Odstranjevanje elementov iz seznama

Če želite odstraniti elemente iz seznama, uporabite ukaz `del` (kratko za brisanje). Na primer, če želite odstraniti šesti element s čarovnikovega seznama, kačji prhljaj, naredite to:

```
>>> del carovnik_seznam[5]
>>> print(carovnik_seznam)
['pajkove noge', 'žabji prst', 'polžji jezik', 'krilo netopirja', 'polžja
slina', 'medvedje bruhanje', 'mandragora', 'ricinus', 'močvirski plin']
```

OPOMBA: Upoštevajte, da se položaji začnejo z nič, tako da se `carovnik_seznam[5]` dejansko nanaša na šesto točko na seznamu.

In tukaj pokažimo še, kako odstraniti elemente, ki smo jih pravkar dodali (mandragora, ricinus in močvirski plin):

```
>>> del carovnik_seznam [8]
>>> del carovnik_seznam [7]
>>> del carovnik_seznam [6]
>>> print(carovnik_seznam)
['pajkove noge', 'žabji prst', 'polžji jezik', 'krilo netopirja', 'polžja slina', 'medvedje bruhanje']
```

Računanje s sezname

Seznami se lahko sestavljajo tako, da jih seštejemo, kot seštevanje števil. Uporabljamo znak plus (+). Recimo, da imamo dva seznama: list1, ki vsebujejo številke od 1 do 4 in seznam2, ki vsebujejo nekaj besed. Lahko ju seštejemo z ukazom print in znakom +, takole:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['Spodtaknil', 'sem', 'se', 'in', 'padel', 'na', 'tla']
>>> print(list1 + list2)
[1, 2, 3, 4, 'Spodtaknil', 'sem', 'se', 'in', 'padel', 'na', 'tla']
```

Sezname lahko tudi seštevamo v nove spremenljivke.

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['Pojedel', 'sem', 'čokolado', 'in', 'je', 'hočem', 'še', 'več']
>>> list3 = list1 + list2
>>> print(seznam3)
[1, 2, 3, 4, 'Pojedel', 'sem', 'čokolado', 'in', 'je', 'hočem', 'še', 'več']
```

Seznam lahko tudi pomnožimo s številko. Na primer, če želite pomnožiti list1 s 5, napišete seznam1 * 5:

```
>>> list1 = [1, 2]
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

To dejansko ukazuje Pythonu, da seznam1 petkrat ponovi v 1, 2, 1, 2, 1, 2, 1, 2, 1, 2.

Po drugi strani pa deljenje (/) in odštevanje (-) privede le do napak, kot v teh primerih:

```
>>> seznam1 / 20
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
list1 / 20
TypeError: unsupported operand type(s) for /: 'list' and 'int'

>>> seznam1 - 20
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
list1 - 20
TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

Toda zakaj? Združevanje seznamov s + in ponavljanje seznamov z * so enostavne operacije. Ti načini prav tako delujejo v resničnem svetu. Na primer, če bi vam dal dva nakupovalna seznama na papirju in rekel: »Združi ta dva seznama«, bi lahko napisali vse predmete na nov list od prvega do zadnjega. Enako bi bilo možno, če bi rekel: »Naredite mi tri take sezname.« Lahko si zamislite, da boste trikrat napisali seznam vseh elementov na nov list.

Toda kako bi delil seznam? Na primer, razmislite, kako bi razdelili seznam šestih števil (od 1 do 6) na dva dela. Tukaj so le trije načini od mnogih:

```
[1, 2, 3] [4, 5, 6]
```

```
[1] [2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4] [5, 6]
```

Ali bi razdelili seznam na sredini, ga razdelili po prvem elementu ali pa nekje naključno? Ne obstaja preprost odgovor na to vprašanje. Ravno tako tega ne zna Python, ne ve, kaj storiti. Zato se odziva z napako.

Podobno velja za dodajanje česar koli drugega kot seznama seznamu. Tega ne morete storiti. Za primer pogledimo, kaj se zgodi, ko poskušamo dodati število 50 na seznam1:

```
>>> seznam1 + 50
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
list1 + 50
TypeError: can only concatenate list (not "int") to list
```

Zakaj imamo tukaj napako? No, kaj pa pomeni dodati 50 na seznam? Ali to pomeni, da vsakemu elementu dodate 50? Kaj pa, če predmeti niso številke? Ali to pomeni dodati številko 50 na koncu ali začetku seznama? V računalniškem programiranju morajo ukazi delovati natančno na enak način vsakič, ko jih vnesete. Ta neumen računalnik vidi stvari samo v črno-beli barvi. Prosite ga za zapleteno odločitev in že dvigne roke z obilico napak.

Tuple (terka)

Tuple je seznam, ki uporablja običajne oklepaje, kot v tem primeru:

```
>>> fibs = (0, 1, 1, 2, 3)
>>> print(fibs [3])
2
```

Tukaj definiramo spremenljivko fibs kot številke 0, 1, 1, 2 in 3. Potem, kot pri seznamu, izpišemo element z indeksom 3 v tuple z uporabo print(fibs [3]). Glavna razlika med tuple in seznamom je, da se tuple ne morete spremeniti, ko je enkrat ustvarjen. Če na primer poskusimo zamenjati prvo vrednost v tuple fibs s številko 4 (prav tako kot smo zamenjali vrednosti v našem carovnik_seznam), dobimo sporočilo o napaki:

```
>>> fibs [0] = 4
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
fibs[0] = 4
TypeError: 'tuple' object does not support item assignment
```

Zakaj bi namesto seznama uporabljali tuple? V bistvu zato, ker je včasih koristno uporabiti nekaj, za kar veste, da se nikoli ne bo spremenilo. Če ustvarite tuple z dvema elementoma, bosta v njem vedno le ta dva elementa.



S Python map (zemljevid) ne boste našli svoje poti

V Pythonu je map (imenovan tudi dict, krajše za slovar) zbirka stvari, podobno kot sezname in tuple. Razlika med slovarji in seznamami ali tuple je, da ima vsak element v slovarju ključ in ustrežno vrednost.

Recimo, da imamo seznam ljudi in njihovih najljubših športov. Te informacije bi lahko dali v seznam z imenom osebe, ki ji sledi njen šport, tako:

```
>>> najljubsi_sporti = ['Ralph Williams, nogomet',
'Michael Tippett, košarka',
'Edward Elgar, baseball',
'Rebecca Clarke, netball',
'Ethel Smyth, badminton',
'Frank Bridge, rugby']
```

Če bi vas vprašali, kaj je najljubši šport Rebecce Clarke, bi lahko preiskal seznam in ugotovili, da je to netball. Kaj pa, če je v seznam vključenih 100 (ali še veliko več) ljudi?

Če bomo te iste podatke shranili v slovar, z imenom osebe kot ključ in najljubši športom kot vrednostjo, bi Python koda izgledala tako:

```
>>> najljubsi_sporti = {'Ralph Williams': 'nogomet',
'Michael Tippett': 'košarka',
'Edward Elgar': 'baseball',
'Rebecca Clarke': 'netball',
'Ethel Smyth': 'badminton',
'Frank Bridge': 'rugby'}
```



Uporabljamo dvopičje, da ločimo vsak ključ od njegove vrednosti in vsak ključ in vrednost sta obdana z enojnimi narekovaji. Verjetno ste opazili, da so vsi elementi obdani z zavrtimi oklepajih ({}), ne z običajnimi oklepaji ali oglatimi oklepaji.

Rezultat je slovar (vsak ključ kaže na določeno vrednost), kot je prikazano v tabeli.

Tabela: Ključi, ki kažejo na vrednosti v slovarju priljubljenih športov

ključ	vrednost
Ralph Williams	Nogomet
Michael Tippett	košarka
Edward Elgar	Baseball
Rebecca Clarke	Netball
Ethel Smyth	Badminton
Frank Bridge	Rugby

Da bi dobili najljubši šport Rebecce Clarke, v našem slovarju najljubsi_sporti le uporabimo ustrezen ključ, tako:

```
>>> print(najljubsi_sporti['Rebecca Clarke'])
Netball
```

In odgovor je netball. Če želite izbrisati vrednost v slovarju, uporabite njen ključ. Na primer, tukaj je kako odstraniti Ethel Smyth:

```
>>> del najljubsi_sporti['Ethel Smyth']
>>> print(najljubsi_sporti)
```

```
{'Rebecca Clarke': 'netball', 'Michael Tippett': 'košarka', 'Ralph Williams': 'nogomet', 'Edward Elgar': 'baseball', 'Frank Bridge': 'rugby'}
```

Za spreminjanje vrednosti v slovarju uporabljamo ključ:

```
>>> najljubsi_sporti['Ralph Williams'] = 'hokej na ledu'
>>> print(najljubsi_sporti)
{'Rebecca Clarke': 'netball', 'Michael Tippett': 'košarka', 'Ralph Williams': 'hokej na ledu', 'Edward Elgar': 'baseball', 'Frank Bridge': 'rugby'}
```

Zamenjamo najljubši šport nogomet z ledenim hokejem z uporabo ključa Ralph Williams.

Kot lahko vidite, delo s slovarjem je podobno kot s seznamom in tuple, razen, da ne morete seštevati slovarjev z operaterjem plus (+). Če poskusite to narediti, boste prejeli sporočilo o napaki:

```
>>> najljubsi_sporti = {'Rebecca Clarke': 'netball',
'Michael Tippett': 'košarka',
'Ralph Williams': 'hokej na ledu',
'Edward Elgar': 'baseball',
'Frank Bridge': 'rugby'}
>>> najljubse_barve = {'Malcolm Warner': 'rožnate pike',
'James Baxter': 'oranžne črte',
'Sue Lee': 'vijolična'}
>>> najljubsi_sporti + najljubse_barve
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

Združevanje slovarjev ni smiselno za Python, zato samo javi napako.

Kaj ste se naučili

V tem poglavju ste se naučili, kako se v Pythonu uporablja nize za shranjevanje besedila, kako uporabljati sezname in tuple za obdelavo več elementov. Videli ste da se elementi v seznamih lahko spremenijo in da se lahko seštevajo sezname z drugimi sezname, v tuple se pa vrednosti ne da spreminjati. Prav tako ste se naučili, kako uporabljati slovarje za shranjevanje vrednosti s ključi, ki jih identificirajo.

4. poglavje: Risanje z želvami

Želva v Pythonu je podobna želvi v resničnem svet. Poznamo jo kot plazilca, ki se premika zelo počasi in živi v oklepu. V svetu Pythona, je želva majhna črna puščica, ki se premika po zaslonu. Pravzaprav je Pythonova želva manj želva in bolj podobna polžu, ob upoštevanju, da pušča sled, ko se premika naokrog po zaslonu.

Želva je lep način, kako se naučiti nekaj osnov računalniške grafike, tako da bomo v tem poglavju uporabljali Pythonovo želvo za risanje nekaj preprostih oblik in linij.

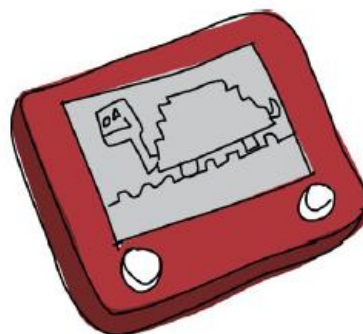
Uporaba Python turtle modula

Modul v Pythonu je način za zagotavljanje uporabne kode, da je na voljo drugim programom (med drugimi stvarmi modul lahko vsebuje funkcije, ki jih lahko uporabimo). O modulih se bomo več naučili v poglavju 7. Python ima poseben modul, imenovan turtle, ki ga uporabljamo za učenje računalniškega risanja na zaslon. Modul turtle je način programiranja vektorske grafike, ki je v bistvu risanje s samo enostavnimi črtami, pikami in krivuljami.

Poglejmo, kako deluje želva. Najprej zaženite lupino Python. Nato povejte Pythonu, da bi radi uporabili želvo tako, da uvozite modul turtle, kot sledi:

```
>>> import turtle
```

Uvoz modula pove Pythonu, da ga želite uporabiti.

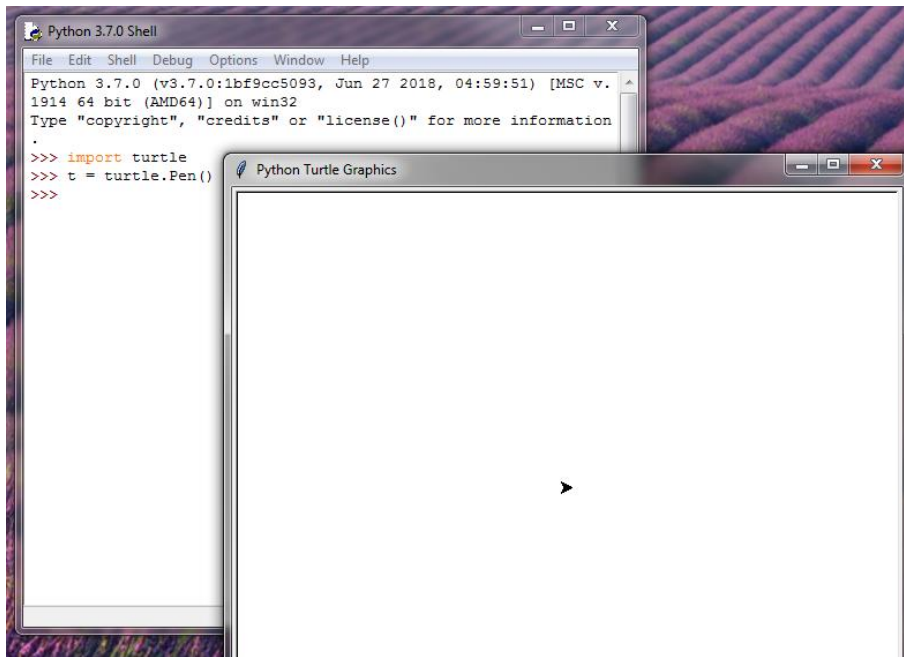


Ustvarjanje platna

Sedaj, ko smo uvozili modul turtle, moramo ustvariti platno - prazen prostor za risanje, podobno kot umetnikovo platno. Pokličemo funkcijo Pen iz modula turtle, ki samodejno ustvari platno (več o funkcijah bomo izvedeli v naslednjih poglavjih). Vnesite to v Python lupino:

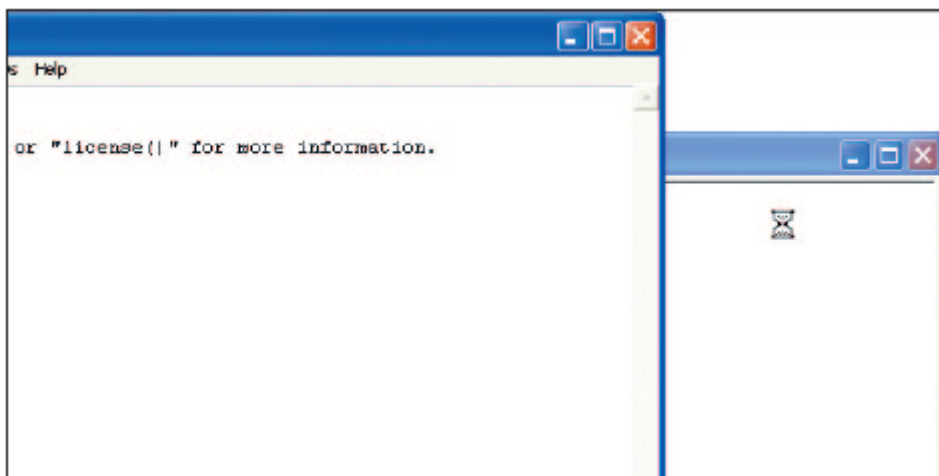
```
>>> t = turtle.Pen()
```

Videti bi morali prazno polje (platno) s puščico v središču, nekaj takega:



Puščica na sredini zaslona je želva in prav imaš - ni ravno želva.

Če se okno Turtle pojavi za oknom lupine Python, boste ugotovili, da ne deluje pravilno. Ko premaknete miško preko okna Turtle, se kazalec spremeni v peščena uro, takole:



To se lahko zgodi iz več razlogov: niste pognali lupine iz ikone na namizju (če uporabljate Windows ali Mac), v meniju Windows Start ste kliknili IDLE (Python GUI), ali pa IDLE ni pravilno nameščen. Poskusite zapreti program in znova zagnati lupino iz ikone na namizju. Če to ne uspe, poskusite uporabiti Python konzolo namesto lupine, kot sledi:

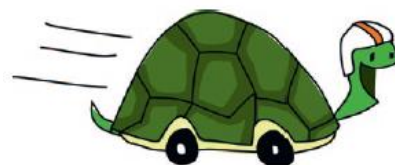
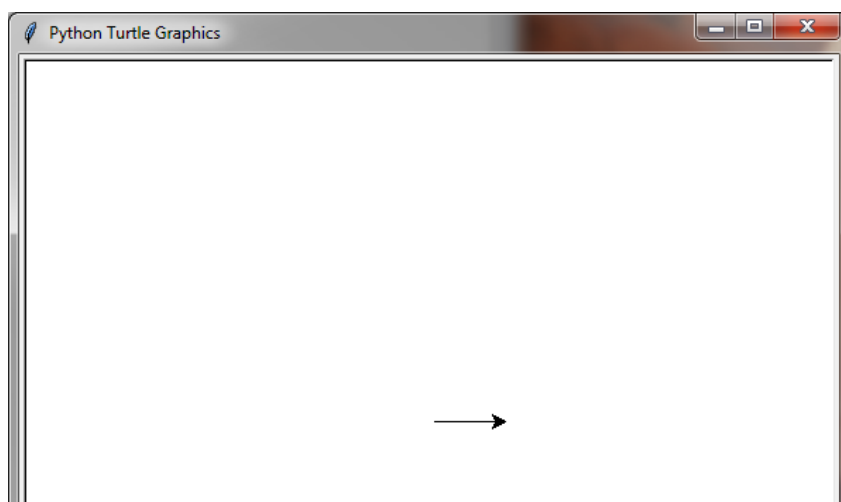
- V operacijskem sistemu Windows izberite Start, All Programs, nato pa v Python 3.7 skupini, kliknite Python (ukazna vrstica).
- V Mac OS X v zgornjem desnem kotu kliknite ikono Spotlight na zaslonu in vnesite Terminal v vnosno polje. Potem vnesite python, ko se terminal odpre.
- V Ubuntu odprite terminal iz menija Aplikacije in vpišite python.

Premikanje želve

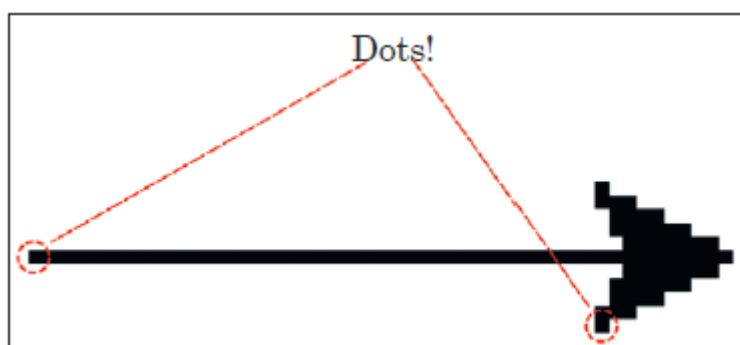
Navodila za želvo pošiljate z uporabo funkcij, ki so na voljo na spremenljivki `t`, ki smo jo pravkar ustvarili, podobno kot z uporabo funkcije `Pen` v modulu `turtle`. Na primer, `forward` navodilo pove želvi naj gre naprej. Za napredovanje za 50 slikovnih pik, vnesite naslednji ukaz:

```
>>> t.forward(50)
```

Moral bi videti nekaj takega:



Želva se je pomaknila naprej za 50 slikovnih pik. Slikovna pika je ena točka na zaslonu - najmanjši element, ki ga je mogoče predstaviti. Vse, kar vidite v računalniškem monitorju, je sestavljeno iz slikovnih pik, ki so majhne, kvadratne pike. Če bi lahko povečali platno in črto, ki jo je pripravila želva, bi lahko videli puščico, ki predstavlja pot želve, le skupina pik. To je preprosta računalniška grafika.



Sedaj bomo rekli želvi naj zavijemo levo za 90 stopinj z naslednjim ukazom:

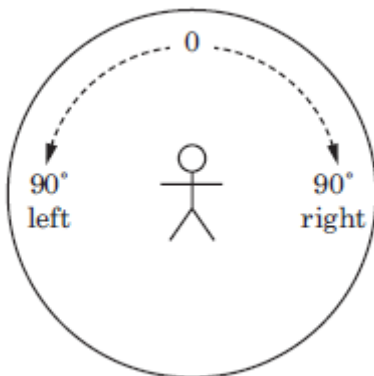
```
>>> t.left(90)
```

Če se še niste učili o stopinjah, pogledjmo, kako razmišljati o njih. Predstavljajte si, da stojite v središču kroga.

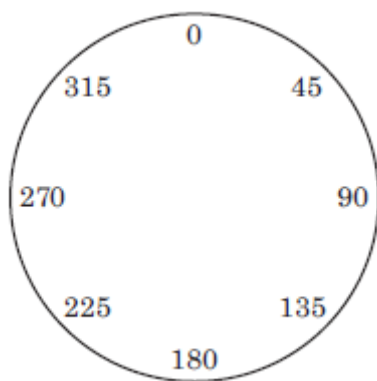
- Smer, v katero gledate, je 0 stopinj.
- Če iztegete levo roko, je to levo za 90 stopinj.

- Če iztegnete desno roko, je to 90 stopinj v desno.

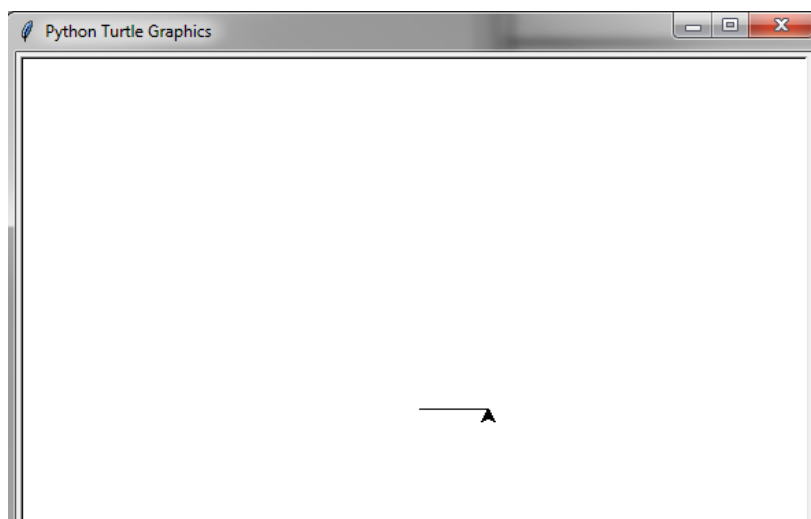
To 90-stopinjsko zavijanje lahko vidite spodaj:



Če nadaljujete po krogu naprej v desno, v smeri vaše desne roke, 180 stopinj je neposredno za vami, 270 stopinj je smer, v katero kaže leva roka, in 360 stopinj je tam, kjer ste začeli; stopinje segajo od 0 do 360. Stopinje lahko vidite v polnem krogu pri obračanju v desno. Tukaj jih vidimo v korakih po 45 stopinj:



Ko se Pythonova želva obrne v levo, se obrne in kaže v novo smer (tako kot bi se obrnili v smer, kamor je kazala vaša leva roka). Ukaz `t.left(90)` usmeri puščico navzgor (če je začela s kazanjem v desni smeri):

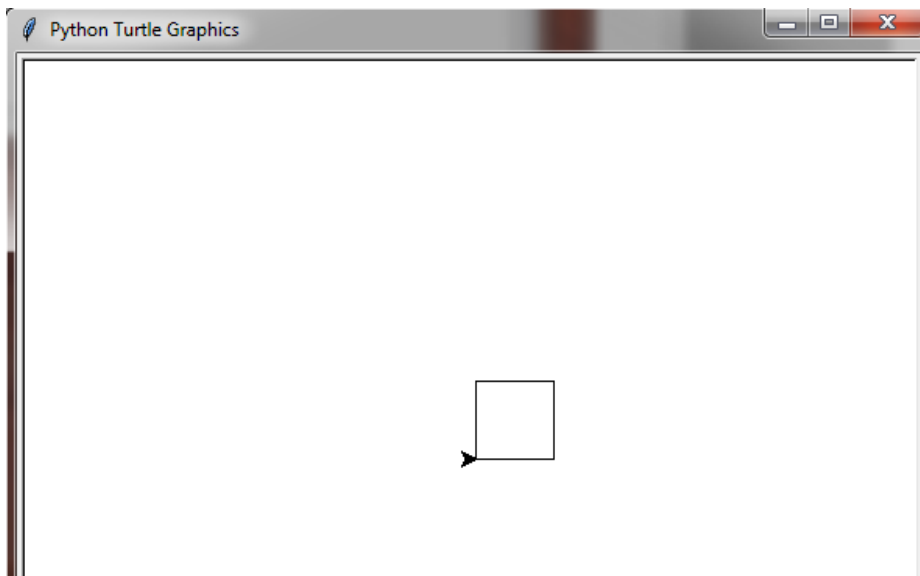


OPOMBA Ko ukažete `t.left (90)`, je enako kot bi ukazali `t.right (270)`. To velja tudi za ukaz `t.right (90)`, ki je enak `t.left(270)`. Zamislite si, da se kroži in sledi stopinjam.

Sedaj bomo naredili kvadrat. Vrsticam, ki ste jih že vnesli dodajte naslednjo kodo:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

Vaša želva bi morala narisati kvadrat in biti obrnjena v isti smeri, kot je bila na začetku:



Če želite izbrisati platno, vnesite `reset`. To počisti platno in postavi želvo nazaj na začetni položaj.

```
>>> t.reset()
```

Uporabite lahko tudi `clear`, kar samo očisti zaslon in pusti želvo, kjer je.

```
>>> t.clear()
```

Prav tako lahko zavrtimo želvo v desno (`right`) ali premaknemo nazaj (`backward`). Uporabite `up` za dvig svinčnika s platna (z drugimi besedami: rečete želvi naj neha risati) in `down`, da začnete risati. Te funkcije se uporabljajo tako kot že opisane.

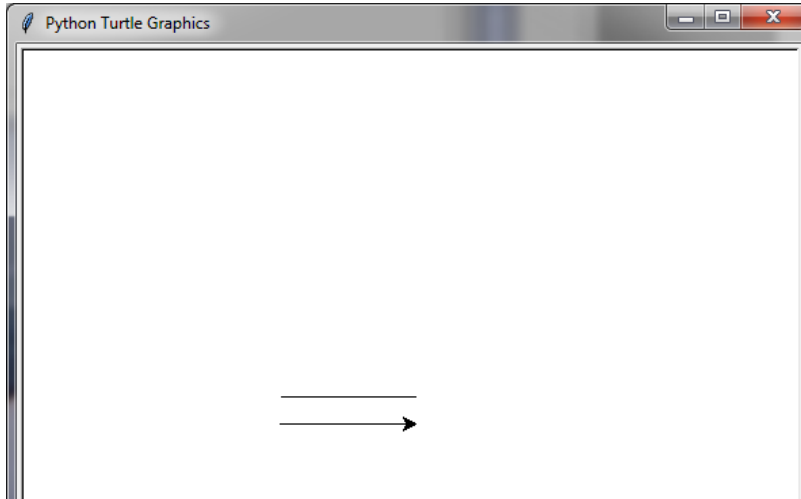
Poskusimo s še eno risbo z uporabo nekaterih od teh ukazov. Sedaj bomo narisali dve črti. Vnesite naslednjo kodo:

```
>>> t.reset()
>>> t.backward(100)
>>> t.up()
>>> t.right(90)
>>> t.forward(20)
>>> t.left(90)
>>> t.down()
>>> t.forward(100)
```



Najprej ponastavimo platno in premaknemo želvo nazaj na začetni položaj s `t.reset ()`. Nato premaknemo želvo nazaj 100 slikovnih pik s `t.backward (100)` in nato s `t.up ()` dvignemo svinčnik, da zaustavimo risanje.

Nato z ukazom `t.right (90)`, obrnemo želvo desno za 90 stopinj navzdol, proti dnu zaslona, in s `t.forward (20)`, se premaknemo naprej 20 slikovnih pik. Zaradi uporabe ukaza `up` se črta ne nariše. Želvo zavrtimo levo za 90 stopinj s `t.left (90)`, nato pa z ukazom `down`, povemo želvi naj se pero spusti in znova začne risati. Končno, potegnemo črto naprej, vzporedno s prvo črto, ki smo jo narisali, s `t.forward (100)`. Dve vzporedni črti, ki smo ju pripravili, sta na koncu videti takole:



Kaj ste se naučili

V tem poglavju ste se naučili uporabljati Pythonov modul `turtle`. Povlekli smo nekaj preprostih črt z uporabo levega in desnega zasuka ter ukazov za naprej in nazaj. Ugotovili ste, kako ustaviti želvje risanje in kako znova začeti risati. Prav tako ste odkrili da se želva obrača v stopinjah.

Vaje

Poskusite z želvo narediti nekaj naslednjih oblik. Odgovore najdete na <https://nostarch.com/pythonforkids>.

1: pravokotnik

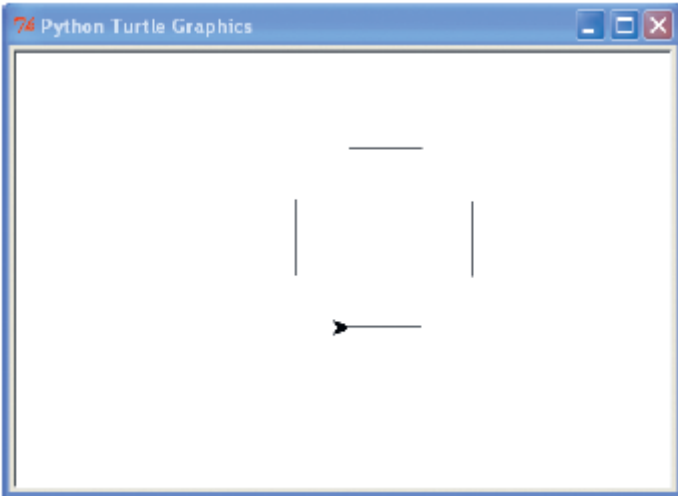
Ustvarite novo platno s funkcijo `Pen` in potem narišite pravokotnik.

2: Trikotnik

Ustvarite drugo platno, in tokrat narišite trikotnik. Poglej nazaj na diagram kroga s stopnjami ("Premikanje želve"), da osvežite v katero smer želite zavrteti želvo z uporabo stopinj.

3: Škatla brez vogalov

Napišite program za risanje štirih črt, prikazanih tukaj (velikost ni pomembna, le oblika):



5. poglavje: Postavljanje vprašanj z IF in ELSE

Pri programiranju pogosto postavljamo da in ne vprašanja in se potem odločimo, kaj bomo naredili v enem primeru in kaj v drugem. Na primer, vprašamo lahko: "Ali ste starejši od 20 let?" in če je odgovor da, izpišete »Preveč ste stari!«

Ta vrsta vprašanj se imenuje pogoji. Pogoje in odzive združujemo v pogojne stavke (if stavki). Pogoji so lahko bolj zapleteni od enega samega vprašanja in if stavke je mogoče kombinirati tudi z več pogoji in različnimi odgovori.

V tem poglavju se boste naučili, kako uporabljati pogojne stavke pri izdelavi programa.

Pogojni stavki

Pogojni stavek je v Pythonu lahko napisan takole:

```
>>> starost = 13
>>> if starost > 20:
    print('Preveč si star!')
```

Pogojni stavek je sestavljen iz ključne besede if, ki ji sledi pogoj in dvopičje (:), tako kot pri if starost > 20:. Vrstice, ki sledijo dvopičju morajo biti v bloku in če je odgovor na vprašanje da (ali True, kot pravimo v Python programiranju), bodo ukazi v bloku izvedeni. Sedaj pa raziščimo, kako napisati bloke in pogoje.



Blok je skupina programskih ukazov

Blok kode je združen niz programskih stavkov. Na primer, ko je izjava if starost > 20: resnična, lahko izvedete več ukazov, kot le izpišete "Preveč si star!" Morda želite izpisati še nekaj dodatnih stavkov, kot so ti:

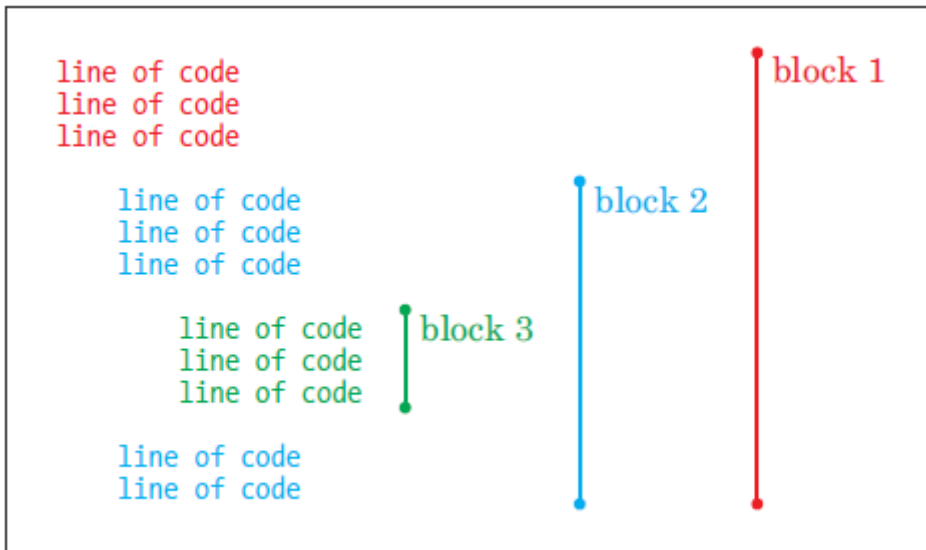
```
>>> starost = 25
>>> if starost > 20:
    print('Preveč ste stari!')
    print('Zakaj ste tu?')
    print('Zakaj ne kosite travnika ali urejate papirjev?')
```

Ta blok kode je sestavljen iz treh ukazov za izpisovanje, ki se izvajajo le, če je trditev starost > 20 resnična. Vsaka vrstica v bloku ima štiri presledke na začetku, ko jo primerjate s pogojnim stavkom zgoraj. Poglejmo to kodo znova, z vidnimi presledki:

```
>>> starost = 25
>>> if starost > 20:
----print('Preveč ste stari!')
----print('Zakaj ste tu?')
----print('Zakaj ne kosite travnika ali urejate papirjev?')
```

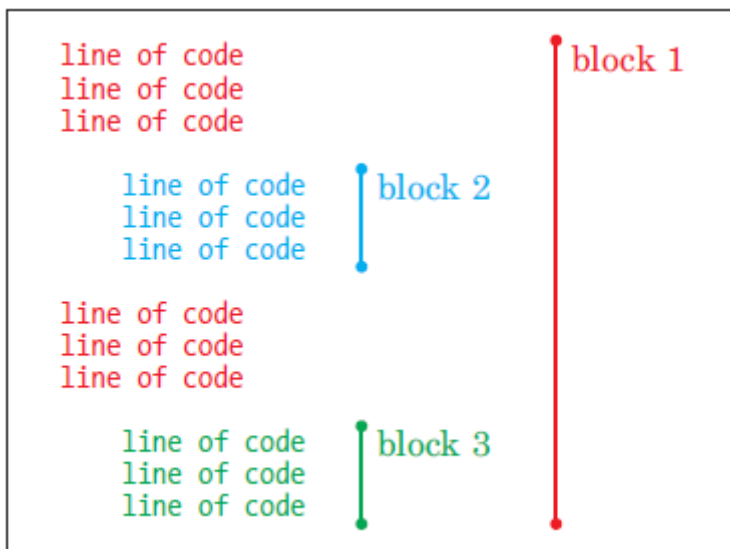
V Pythonu so prazni prostori, kot je tab (vstavljen, ko pritisnete TAB tipko) ali presledek (vstavljen, ko pritisnete preslednico), pomembni. Koda, ki je na istem zamiku (zamaknjena enaka število presledkov

od levega roba) je združena v blok, in ko začnete novo vrstico z več presledki, kot v prejšnji vrstici, začnete nov blok, ki je del prejšnjega, kot to:



Ukaze združujemo v blokade, ker so povezani. Ukaze je treba izvajati skupaj.

Ko spremenite zamik, ustvarite nov blok. Naslednji primer prikazuje tri ločene bloke, ki so narejeni le z zamikom.



Tukaj, čeprav imata bloka 2 in 3 enak zamik, sta to različna bloka, ker je med njima blok z manj zamika (manj presledki).

Zaradi tega bo v primeru bloka s štirimi presledki v eni vrstici in šest presledki v naslednji prišlo do napake v zamiku (indentation error), ko ga poženete. Python pričakuje, da boste uporabili enako število presledkov za vse vrstice v bloku. Torej, če začnete blok s štirimi presledki, morate za ta blok dosledno uporabljati štiri presledke. Tukaj je primer:

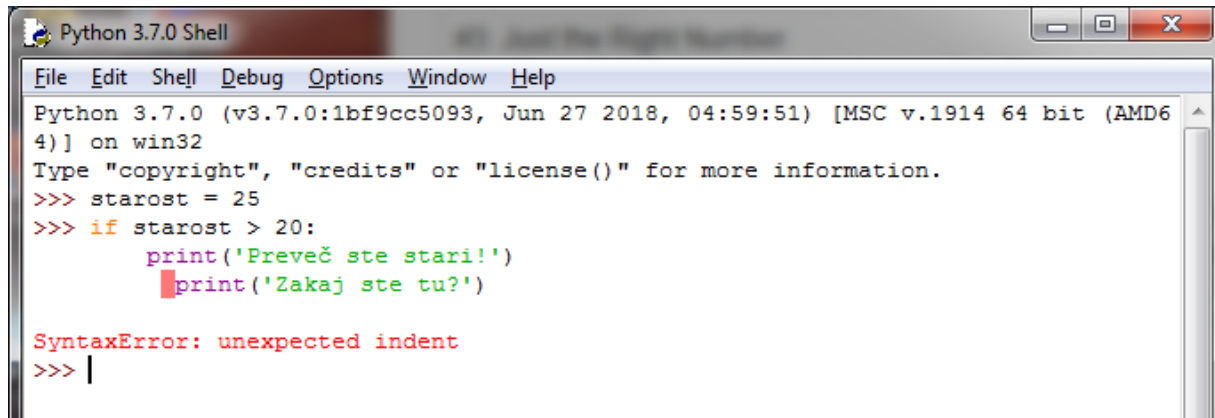
```
>>> if starost > 20:
----print('Preveč ste stari!')
-----print('Zakaj ste tu?')
```

Presledki so vidni, da vidite razliko. Upoštevajte, da ima tretja vrstica šest presledkov, namesto štiri.

Ko poskušamo zagnati to kodo, IDLE osvetli vrstico z rdečim blokom kjer vidi problem in prikaže `SyntaxError` sporočilo:

```
>>> starost = 25
>>> if starost > 20:
    natisni('Preveč si star!')
    natisni('Zakaj ste tu?')
```

`SyntaxError: unexpected indent`



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> starost = 25
>>> if starost > 20:
    print('Preveč ste stari!')
    print('Zakaj ste tu?')

SyntaxError: unexpected indent
>>> |
```

Python ni pričakoval, da bo videl dva dodatna presledka na začetku druge vrstice pri izpisu.

Opomba: Dosledno uporabljajte zamikanje, da bo vaša koda lažje čitljiva. Če začnete pisanje programa in uporabljate štiri presledke na začetku bloka, še naprej uporabljajte štiri presledke na začetku drugih blokov v vašem programu. Prav tako vsako vrstico v istem bloku zapišite z enakim številom presledkov.

Pogoji nam pomagajo primerjati stvari

Pogoj je programska izjava, ki primerja stvari in nam pove ali je izjava glede na kriterij pravilna - True (da) ali napačna - False (ne). Na primer, `starost > 10` je pogoj in je druga oblika izjave: "Ali je vrednost spremenljivke `starost` večja od 10?" To je tudi pogoj: `barva_las == 'rjava'`, kar je drug način za izjavo: "Ali je vrednost spremenljivke `barva_las` rjava?"

V Pythonu uporabljamo simbole (imenovane operatorji) za oblikovanje naših pogojev, kot so: enak, večji in manjši kot. Tabela navaja nekaj simbolov za pogoje.

Tabela: Simboli za pogoje

Simbol	Definicija
<code>==</code>	Enako kot
<code>!=</code>	Ni enako
<code>></code>	Več kot
<code><</code>	Manj kot
<code>>=</code>	Večje ali enako
<code><=</code>	Manj ali enako

Na primer, če ste stari 10 let, bo izjava `tvoja_starost == 10` vrnila `True`; sicer bi vrnila `False`. Če ste stari 12 let, bi za izjavo `tvoja_starost > 10` vrnilo `True`.

Opozorilo: Pri primerjavi enakosti uporabite dvojni znak (==).

Poskusimo še nekaj primerov. Tukaj bomo starost nastavili na 10 in nato napisali pogojno izjavo, ki bo izpisala "Vi ste prestari za moje šale!", če bo starost večja od 10:

```
>>> starost = 10
>>> if starost > 10:
    print('Prestar si za moje šale!')
```

Kaj se zgodi, ko vnesemo to v IDLE in pritisnemo enter? Nič. Ker vrednost v spremenljivki `starost` ni večja od 10, Python ne izvrši (požene) `print` blok. Če pa vrednost spremenljivke `starosti` nastavimo na 20, se sporočilo izpiše. Sedaj pa spremenimo prejšnji primer z uporabo več-kot-ali-enak (`>=`):

```
>>> starost = 10
>>> if starost >= 10:
    print('Prestar si za moje šale!')
```

Morali bi videti izpis »Prestar si za moje šale!«, ker je vrednost spremenljivke `starost` enaka 10. Poskusimo sedaj uporabiti operator enakosti (`==`):

```
>>> starost = 10
>>> if starost == 10:
    print('Kaj je rjavo in paličasto? Palica!')
```

Morali bi videti sporočilo "Kaj je rjavo in paličasto? Palica!" izpisano na zaslonu.



If-then-else stavki

Pogojne stavke lahko poleg resničnih izjav uporabimo tudi za neresnične (če pogoj ni izpolnjen). Na primer, eno sporočilo izpišemo na zaslon, če je vaša starost 12 in drugo, če ni 12 (`False`).

Trik tukaj je uporaba stavka `if-then-else`, kar v bistvu pravi: "Če je nekaj res, potem naredite to; sicer pa storite to."

Ustvarimo en stavek `if-then-else`. V lupino vnesite naslednje:

```
>>> print("Želite slišati umazano šalo?")
Želite slišati umazano šalo?
>>> starost = 12
>>> if starost == 12:
    print("Svinja je padla v blato!")
else:
    print("Pst, to je skrivnost")
Svinja je padla v blato!
```



Ker smo nastavili spremenljivko `starost` na 12 in pogoj sprašuje, če je starost enaka 12, bi morali na zaslonu videti prvi izpis. Poskusite spreminjati vrednosti za starost, na primer:

```
>>> print("Želite slišati umazano šalo?")
Želite slišati umazano šalo?
>>> starost = 8
>>> if starost == 12:
```

```

    print("Svinja je padla v blato!")
else:
    print("Pst, to je skrivnost")
Pst. To je skrivnost.

```

Tokrat bi morali videti izpisano drugo sporočilo.

If in elif stavki

V pogojnem stavku lahko dodajamo dodatne pogoje z elif (skrajšano za sicer-če). Na primer, preverimo lahko, če je starost osebe 10, 11 ali 12 (in tako naprej) in naš program naredi vsakič nekaj drugega. Te izjave se razlikujejo od if-then-else izjave, saj je v njih lahko več elif izjav:

```

>>> starost = 12
(1)>>> if starost == 10:
(2)     print("What do you call an unhappy cranberry?")
        print("A blueberry! ")
(3) elif starost == 11:
        print("What did the green grape say to the blue grape? ")
        print("Breathe! Breathe! ")
(4) elif starost == 12:
(5)     print("What did 0 say to 8? ")
        print("Hi guys! ")
        elif starost == 13:
            print("Why wasn't 10 afraid of 7? ")
            print("Because rather than eating 9, 7 8 pi.")
        else:
            print("Uf?")
What did 0 say to 8?
Hi guys!

```

V tem primeru pogojni stavek v drugi vrstici želi ugotoviti, če je vrednost spremenljivke starost enaka 10 pri (1). Izpis izjave, ki sledi pri (2), se izvede, če je starost enaka 10. Ker smo starost nastavili na 12, se računalnik premakne na naslednjo izjavo pri (3) in preveri, ali je vrednost starost enaka 11. Ni, torej računalnik preide na naslednji elif pri (4), da vidi, če je starost enaka 12. To drži, zato računalnik tokrat izvede izpis ukaza pri (5).

Ko vnesete to kodo v IDLE, se bo samodejno zamikala, zato uporabite vračalko (backspace) ali brisanje znaka (delete), ko ste vnesli ukaz za izpis. Tako, da se vaši if, elif in else stavki začnejo na levem robu. To je isti položaj, kot ga ima if, če ne bi bilo pozivne vrstice (>>>).

Združevanje pogojev

Pogoje lahko združujete z uporabo ključnih besed and in or, ki omogočata krajšo in preprostejšo kodo. Tukaj je primer uporabe ali:

```

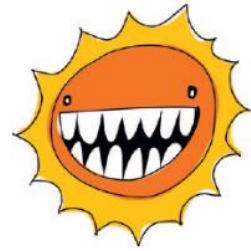
>>> if starost == 10 or starost == 11 or starost == 12 or starost == 13:
        print('Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!')
else:
        print("Uf?")

```

Če je katerikoli od pogojev v tej kodi v prvi vrstici resničen (z drugimi besedami, če je starost 10, 11, 12 ali 13), se bo izvedla koda v naslednji vrstici. Če vsi pogoji v prvi vrstici niso resnični (else), se Python premakne na blok v zadnji vrstici in na zaslону prikaže »Uf?« Da bi ta primer še bolj skrčili,

smo uporabili ključno besedo `and`, skupaj z operatorjem večje-ali-enako (`>=`) in manj-ali-enako (`<=`), kot sledi:

```
>>> if starost >= 10 and starost <= 13:
    print('Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!')
else:
    print("Uf?")
```



Tukaj, če je starost večja ali enaka 10 in manjša ali enaka 13, kot je definirano v prvi vrstici `if starost >= 10 and starost <= 13` :, se bo zagnal blok kode, ki se začne z izpisom v naslednji vrstici. Na primer, če je starost 12 let, bo na zaslon natisnjeno: Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!, ker je 12 več kot 10 in manj kot 13.

Spremenljivke brez vrednosti – None

Tako kot lahko spremenljivkam dodelimo števila, nize in sezname, lahko spremenljivki dodelimo prazno vrednost ali ji ničesar ne dodelimo. V Pythonu se prazna vrednost imenuje **None** in pomeni odsotnost vrednosti. In pomembno je omeniti, da je vrednost `None` drugačna od vrednosti `0`, ker je odsotnost vrednosti, ne pa številka z vrednostjo `0`. Edina vrednost, ki jo ima spremenljivka, ko ji dodelimo prazno vrednost je `None`. Tukaj je primer:

```
>>> myval = None
>>> print(myval)
None
```

Če spremenljivki dodelimo prazno vrednost `None`, lahko rečemo, da spremenljivka nima več nobene vrednosti (ali bolje, da ne označuje več vrednosti). Nastavitev spremenljivke na `None` je tudi način definiranja spremenljivka brez nastavitve vrednosti. To lahko storite, ko veste, da boste pozneje v programu potrebovali spremenljivko, vendar na začetku ne želite nastaviti vseh svojih spremenljivk. Programerji pogosto definirajo svoje spremenljivke na začetku programa, ker imajo potem vse spremenljivke zbrane na enem mestu.

Vrednost `None` lahko preverite v pogojnih stavkih:

```
>>> myval = None
>>> if myval == None:
    print("Spremenljivka myval nima vrednosti")
Spremenljivka myval nima vrednosti
```

To je uporabno, če želite nastaviti vrednost spremenljivke le v primeru, da še ni bila nastavljena.

Razlika med nizi in števili

Uporabniški vnos je tisto, kar uporabnik vnese preko tipkovnice – pa naj bo to črka, pritisnjena puščica, tipka `enter` ali kaj drugega. Uporabniški vnos pride v Python kot niz, kar pomeni, da ko vnesete številko `10` na tipkovnici, Python to številko `10` shrani v spremenljivko kot niz, ne števil. Kakšna je razlika med številko `10` in nizom `"10"`? Oba sta nam enaka, z edino razliko, da je to ena obdana z narekovaji. Toda za računalnik sta to povsem različni stvari.

Recimo, da primerjamo vrednost spremenljivke `starost` s številom v `if` stavku, na primer:

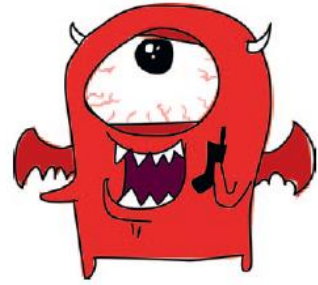
```
>>> if starost == 10:
    print("Kateri je najboljši način pogovora s pošastjo?")
```

```
print("Od kar najbolj daleč!")
```

Potem spremenljivko starost postavimo na število 10:

```
>>> starost = 10
>>> if starost == 10:
    print("Kateri je najboljši način pogovora s
pošastjo?")
    print("Od kar najbolj daleč!")
```

Kateri je najboljši način pogovora s pošastjo?
Od kar najbolj daleč!



Kot lahko vidite, se izjava izpiše. Nato starost nastavili na niz '10' (z narekovaji), na primer:

```
>>> starost = '10'
>>> if starost == 10:
    print("Kateri je najboljši način pogovora s pošastjo?")
    print("Od kar najbolj daleč!")
```

Tukaj izpis ne deluje, ker Python ne vidi znakov v narekovajih (niz) kot številko. Na srečo ima Python čarobno funkcije, ki lahko spremeni nize v številke in številke v nize. Na primer, niz '10' lahko spremenite v številko z int:

```
>>> starost = '10'
>>> spremenjena_starost = int(starost)
```

Spremenljivka spremenjena_starost ima sedaj številko 10.

Če želite pretvoriti številko v niz, uporabite str:

```
>>> starost = 10
>>> spremenjena_starost = str(starost)
```

V tem primeru bo spremenjena_starost vsebovala niz '10' namesto številke 10.

Se spomnite, da izjava if starost == 10, ni izpisala ničesar, ko je bila spremenljivka nastavljena na niz (starost = '10')? Če pretvorimo spremenljivko, dobimo povsem drugačen rezultat:

```
>>> starost = '10'
>>> spremenjena_starost = int(starost)
>>> if spremenjena_starost == 10:
    print("Kateri je najboljši način pogovora s pošastjo?")
    print("Od kar najbolj daleč!")
```

Kateri je najboljši način pogovora s pošastjo?
Od kar najbolj daleč!

Ampak pogledajte tole: če poskušate pretvoriti številko z decimalno piko, boste dobili napako, ker funkcija int pričakuje celo število.

```
>>> starost = '10.5 '
>>> spremenjena_starost = int(starost)
```

```
Traceback (most recent call last):
File "<pyshell#35>", line 1, in <module>
spremenjena_starost = int(starost)
ValueError: invalid literal for int() with base 10: '10.5'
```

Z ValueError Python pove, da poskušate uporabiti napačno vrednost. Če želite to popraviti, uporabite funkcijo float namesto int. Funkcija plavajoče vejice lahko pretvarja števila, ki niso cela števila.

```
>>> starost = '10.5 '  
>>> spremenjena_starost = float(starost)  
>>> print(spremenjena_starost)  
10.5
```

ValueError boste dobili tudi, če skušate pretvoriti niz številke, ki ni zapisana s števki:

```
>>> starost = 'deset'  
>>> spremenjena_starost = int(starost)  
Traceback (most recent call last):  
File "<pyshell#1>", line 1, in <module>  
spremenjena_starost = int(starost)  
ValueError: invalid literal for int() with base 10: 'deset'
```

Kaj ste se naučili

V tem poglavju ste se naučili, kako delati s pogojnimi stavki, ustvariti bloke kode, ki se izvajajo le, če so določeni pogoji izpolnjeni. Videli ste, kako razširiti pogojne stavke z uporabo elif, da so različni deli kode izvršeni ob različnih pogojih in kako uporabiti ključno besedo else za izvedbo kode, če ni izpolnjen noben pogoj. Naučili ste se tudi združiti pogoje, ki uporabljajo ključne besede and in or, da lahko vidite, če so števila v določenem razponu in kako spreminjati nize v števila in obratno z int, str in float. Ugotovili ste tudi, da ima prazna vrednost (None) svoj pomen v Pythonu in se lahko uporablja za ponastavitev spremenljivk na njihovo začetno prazno stanje.

Vaje

Poskusite naslednje vaje s pogojnimi stavki. Odgovore najdete na <https://nostarch.com/pythonforkids>.

1: Ali ste bogati?

Kaj menite, da bo naredila naslednja koda? Poskusite ugotoviti, ne da bi program vtiskali v lupino, in nato preverite odgovor.

```
>>> denar = 2000  
>>> if denar > 1000:  
    print("Bogat sem!")  
else:  
    print("Nisem bogat!")  
    print("Ampak bom morda kasneje ...")
```

2: Kolački!

Ustvarite pogojni stavek in preverite, če je število kolačkov (v spremenljivki kolacki) manj kot 10 ali večje kot 50. Vaš program naj izpiše sporočilo "Premalo ali preveč", če je pogoj resničen.

3: Samo prava številka

Ustvarite pogojni stavek, če je znesek denarja v spremenljivki denar med 100 in 500 ali med 1000 in 5000.

4: Lahko se borim proti tistim ninjam

Ustvarite pogojni stavek, ki izpiše niz "To je preveč", če spremenljivka ninje vsebuje število, ki je manjše kot 50, "To bo boj, ampak jih lahko premagam", če je manj kot 30, "Lahko se borim s temi ninjami!", če je manj kot 10. Morda preizkusite kodo z:

```
>>> ninje = 5
```

6. poglavje: Gremo se zanke

Nič ni hujšega, kot isto stvar delati znova in znova. Obstaja razlog, zakaj nekateri štejejo ovce, ko imajo težave s spanjem. To nima nič opraviti z neverjetno spalno-močjo prijetnih ovčic. To je zato, ker je neskončno ponavljanje nečesa, zelo dolgočasno. Če se ne dogaja kaj zanimivega, postanete zaspani.

Programerji se še posebno neradi ponavljajo, razen če se tudi trudijo zaspati. K sreči ima večina programskih jezikov tisto, kar se imenuje for zanka, ki avtomatsko ponavlja stvari, kot so skupine programskih stavkov in bloki kode. V tem poglavju bomo pregledali for zanke, kot tudi drugo vrsto zanke, ki jo ponuja Python: while zanke.

Uporaba for zank

Če želite v Python natisniti pet krat zdravo, lahko naredite naslednje:

```
>>> print("zdravo")
zdravo
>>> print("zdravo")
zdravo
>>> print("zdravo")
zdravo
>>> print("zdravo")
zdravo
>>> print("zdravo")
zdravo
```

Ampak to je precej dolgočasno. Namesto tega lahko uporabite for zanko in zmanjšate količino tipkanja in ponovitev, na primer:

```
(1)>>> for x in range(0, 5):
(2)     print("zdravo")
zdravo
zdravo
zdravo
zdravo
zdravo
```

Funkcijo range na (1) lahko uporabite za ustvarjanje seznama števil, ki se gibljejo od začetne številke do številke tik pred končno številko. To lahko zveni nekoliko zmedeno. Združimo range funkcijo s funkcijo list, da boste bolje videli, kako to deluje. Funkcija range dejansko ne ustvarja seznama števil; vrne iterator, ki je tip objekta v Pythonu, posebej namenjen delu z zankami. Če pa združimo range in list, dobimo seznam števil.

```
>>> print(list(range(10, 20)))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

V primeru for zanke, koda na (1) dejansko ukaže naj Python naredi naslednje:

- Začnete štetje od 0 in se ustavite, preden dosežete 5.
- Za vsako številko, ki jo štejemo, shranimo vrednost v spremenljivko x.

Potem Python izvrši blok kode pri (2). Upoštevajte, da so na začetku vrstice (2) štirje presledki (ko primerjate z vrstico (1)). IDLE bo to storil samodejno.

Ko pritisnemo enter po drugi vrstici, Python izpiše pet krat "zdravo".

V našem izpisu lahko uporabimo tudi x za štetje pozdravov:

```
>>> for x in range(0, 5):
    print('zdravo %s' % x)
zdravo 0
zdravo 1
zdravo 2
zdravo 3
zdravo 4
```

Če za trenutek pozabimo na for zanko, bi naša koda lahko izgleda nekaj podobnega:

```
>>> x = 0
>>> print('zdravo %s' % x)
zdravo 0
>>> x = 1
>>> print('zdravo %s' % x)
zdravo 1
>>> x = 2
>>> print('zdravo %s' % x)
zdravo 2
>>> x = 3
>>> print('zdravo %s' % x)
zdravo 3
>>> x = 4
>>> print('zdravo %s' % x)
zdravo 4
```



Torej, uporaba for zanke nam je dejansko prihranila pisanje osem dodatnih vrstic kode. Dobri programerji sovražijo delati isto stvar več kot enkrat, zato je for zanka eden izmed bolj priljubljenih stavkov v programskem jeziku.

Ni vam treba uporabljati funkcij range in list za pisanje for zank. Uporabite lahko tudi seznam, ki ste ga že imeli ustvarjenega, kot na primer nakupovalni seznam iz 3. poglavja, kot sledi:

```
>>> carovnikov_seznam = ['pajkove noge', 'žabji prst', 'polžji jezik',
                        'krilo netopirja', 'polžja slina', 'medvedje bruhanje']
>>> for i in carovnikov_seznam:
    print(i)
pajkove noge
žabji prst
polžji jezik
krilo netopirja
polžja slina
medvedje bruhanje
```

Ta koda je, kot bi rekel: "Za vsak element v carovnikov_seznam, shranite vrednost v spremenljivko i, nato izpišite vsebino te spremenljivke." Spet, če bi želeli napisati brez zanke, bi morali storiti nekaj podobnega:

```
>>> carovnikov_seznam = ['pajkove noge', 'žabji prst', 'polžji jezik',
                        'krilo netopirja', 'polžja slina', 'medvedje bruhanje']
>>> print(carovnikov_seznam [0])
pajkove noge
>>> print(carovnikov_seznam [1])
```



```

žabji prst
>>> print(carovnikov_seznam [2])
polžji jezik
>>> print(carovnikov_seznam [3])
krilo netopirja
>>> print(carovnikov_seznam [4])
polžja slina
>>> print(carovnikov_seznam [5])
medvedje bruhanje

```

Torej še enkrat, zanka nam prihrani veliko tipkanja.

Ustvarimo še eno zanko. Vnesite naslednjo kodo v lupino. Zamiki se naredijo samodejno.

```

(1) >>> ogromnekosmatehlace = ['ogromne', 'kosmate', 'hlače']
(2) >>> for i in ogromnekosmatehlace:
(3)     print(i)
(4)     print(i)
(5)
(6) ogromne
ogromne
kosmate
kosmate
hlače
hlače

```



V prvi vrstici (1) ustvarimo seznam, ki vsebuje "ogromne", "kosmate" in "hlače". V naslednji vrstici (2), zankamo po elementih v seznamu in vsak element se nato dodeli spremenljivki i. Vsebino spremenljivke izpišemo dva krat v naslednjih dveh vrsticah (3 in 4). S pritiskom na enter v naslednji prazni vrstici (5) povemo Pythonu naj konča blok in nato zažene kodo in izpiše vsak element seznama dva krat (6).

Ne pozabite, da boste pri vnosu napačnega števila presledkov na koncu dobili sporočilo o napaki. Če ste vnesli prejšnjo kodo z dodatnim presledkom v četrti vrstici (4), bi Python prikazal napaka pri vnosu:

```

(1) >>> ogromnekosmatehlace = ['ogromne', 'kosmate', 'hlače']
(2) >>> for i in ogromnekosmatehlace:
(3)     print(i)
(4)     print(i)

```

SyntaxError: unexpected indent

Kot ste se naučili v 5. poglavju, Python pričakuje dosledno število presledkov v bloku. Ni pomembno, koliko jih je, ko jih vstavite, vendar jih morate uporabljati isto število (poleg tega je lažje brati kodo).

Tukaj je bolj zapleten primer for zanke z dvema blokoma kode:

```

(1) ogromnekosmatehlace = ['ogromne', 'kosmate', 'hlače']
(2) for i in ogromnekosmatehlace:
(3)     print(i)
(4)     for j in ogromnekosmatehlace:
(5)         print(j)

```

Kje so bloki v tej kodi? Prvi blok je prva for zanka, to so vrstice (2-5). Drugi blok je ena vrstica izpisa v drugi zanki (5). Se pravi, vrstica (5) je tako prvi kot drugi blok.

Ali lahko ugotovite, kaj bo te nekaj kode storilo?

Ko se na (1) ustvari seznam, imenovan ogromnekosmatehlace, lahko povemo za naslednjih dve vrstici, da se bo zanka sprehodila po elementih seznama in izpisala vsakega. Vendar pa bo pri (4) ponovno zankala skozi seznam in tokrat dodelila vrednost spremenljivki j in nato vsakega zopet izpisala v (5). Koda pri (4 in 5) je še vedno del prve zanke, kar pomeni, da bodo vrstice izvedene za vsak element for zanke. Torej, ko ta koda teče, bi morali videti ogromne, ki jim sledi ogromne, kosmate, hlače, nato pa kosmate, ki ji sledijo ogromne, kosmate, hlače in tako naprej.

Vnesite kodo v lupino Python in si oglejte sami:

```
>>>ogromnekosmatehlace = ['ogromne', 'kosmate', 'hlače']
>>>for i in ogromnekosmatehlace:
(1) print(i)
    for j in ogromnekosmatehlace:
(2)     print(j)
(-) ogromne
ogromne
kosmate
hlače
(-) kosmate
ogromne
kosmate
hlače
(-) hlače
ogromne
kosmate
hlače
```

Python vstopi v prvo zanko in izpiše element s seznama na (1). Nato vstopi v drugo zanko in natisne vse elemente iz seznama (2). Nato nadaljujete s print(i), izpiše naslednji element s seznama, nato pa znova natisne celoten seznam s print(j). V izpisu so vrstice, označene z (-), izpisane s print(i) ukazom. Neoznačene vrstice se izpišejo s print(j).

Kaj pa kaj bolj praktičnega kot tiskanje neumnih besed? Ne pozabite, da smo v 2. poglavju za vajo izračunali koliko zlatih kovancev imate ob koncu leta, če uporabite nori izum svojega dedka za kopiranje kovancev? To izgledal takole:

```
>>> 20 + 10 * 365 - 3 * 52
```

To predstavlja 20 najdenih kovancev plus 10 magičnih kovancev, pomnoženih s 365 dnevi na leto, minus trije ukradeni kovanci na teden (sraka).

Morda bi bilo koristno videti, kako se bo vaš kupček zlatnikov povečeval vsak teden. To lahko storimo z novo for zanko, vendar moramo najprej spremeniti vrednost naše spremenljivke carobni_kovanci, tako da predstavlja število čarobnih kovancev na teden. To je 10 čarobnih kovancev na dan x 7 dni v tednu, zato carobni_kovanci dodelimo vrednost 70:

```
>>> najdeni_kovanci = 20
>>> carobni_kovanci = 70
>>> ukradeni_kovanci = 3
```

Da bomo videli, kako se količina kovancev spreminja, si ustvarimo še eno spremenljivko kovanci in uporabimo zanko:

```
>>> najdeni_kovanci = 20
```



```

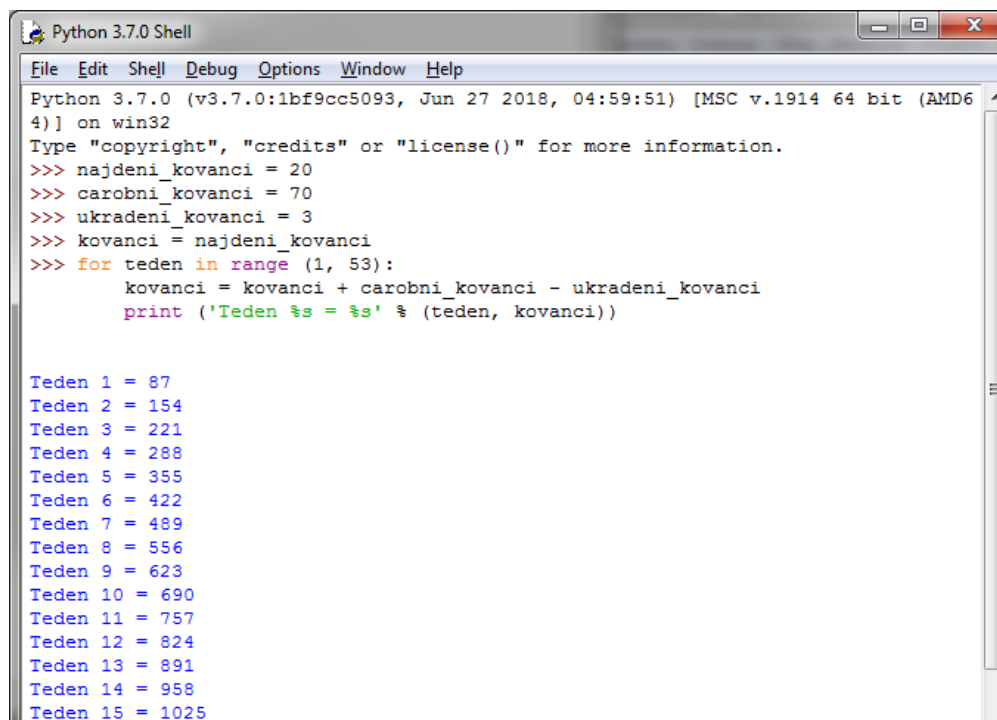
>>> carobni_kovanci = 70
>>> ukradeni_kovanci = 3
(1) >>> kovanci = najdeni_kovanci
(2) >>> for teden in range(1, 53):
(3)     kovanci = kovanci + carobni_kovanci - ukradeni_kovanci
(4)     print('Teden %s = %s' % (teden, kovanci))

```

Pri (1) se spremenljivka kovanci naloži z vrednostjo spremenljivke najdeni_kovanci; to je naše izhodiščno številko. Naslednja vrstica (2) nastavi for zanko, ki bo zagnala ukaze v bloku (blok je sestavljene iz vrstic 3 in 4). Vsakič, ko gremo skozi zanko, se spremenljivka teden naloži z naslednjo številko v obsegu od 1 do 52.

Vrstica (3) je malo bolj zapletena. V bistvu, vsak teden želimo dodati številu kovancev kovance, ki smo jih čarobno ustvarili in odštejemo število kovancev, ki so bili ukradeni. Zamislite si spremenljivko kovanci kot skrinjo. Vsak teden novi kovanci priletijo v skrinjo. Torej ta vrstica resnično pomeni, "Zamenjajte vsebino spremenljivke kovanci s številom mojih sedanjih kovancev, plus tisto, kar sem ustvaril ta teden." V bistvu je enačaja (=) glavni del ukaza, ki pravi: "Opravi najprej nekaj na desni strani in nato shranite za pozneje z uporabo imena na levi."

Vrstica (4) je izpis z uporabo sestavljanja nizov, ki izpiše številko tedna in skupno število kovancev (doslej) na zaslonu. (Če vam to ni smiselno, ponovno preberite »Vsebovane vrednosti v nizu«.) Torej, če zaženete ta program, boste videli nekaj takega:



```

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> najdeni_kovanci = 20
>>> carobni_kovanci = 70
>>> ukradeni_kovanci = 3
>>> kovanci = najdeni_kovanci
>>> for teden in range(1, 53):
    kovanci = kovanci + carobni_kovanci - ukradeni_kovanci
    print ('Teden %s = %s' % (teden, kovanci))

Teden 1 = 87
Teden 2 = 154
Teden 3 = 221
Teden 4 = 288
Teden 5 = 355
Teden 6 = 422
Teden 7 = 489
Teden 8 = 556
Teden 9 = 623
Teden 10 = 690
Teden 11 = 757
Teden 12 = 824
Teden 13 = 891
Teden 14 = 958
Teden 15 = 1025

```

Medtem, ko govorimo o zankanju...

For zanka ni edina vrsta zanke, ki jo lahko naredite v Pythonu.

Obstaja tudi while zanka. For zanka je zanka določene dolžine, medtem ko je while zanka, zanka ki se uporablja, če ne veste, kdaj jo je potrebno ustaviti.

Predstavljajte si stopnišče z 20 stopnicami. Stopnišče je v znotraj in veste, da se lahko enostavno povzpnete 20 korakov. For zanka deluje na tak način.

```
>>> for korak in range(0, 20):  
    print(korak)
```

Sedaj si predstavljajte stopnice, ki vodijo na goro. Gora je res visoka in morda ne boste zmogli do vrha, lahko se poslabša tudi vreme in vas prisili, da se ustavite. Taka je while zanka.

```
korak = 0  
while korak < 10000:  
    print(korak)  
    if utrujen == True:  
        break  
    elif slabovreme == True:  
        break  
    else:  
        korak = korak + 1
```



Če poskusite vnesti in zagnati to kodo, boste dobili napako. Zakaj? Napaka se zgodi, ker nismo ustvarili spremenljivke utrujen in slabovreme. Čeprav tukaj ni dovolj kode, da bi program dejansko tekel, lahko vidimo osnovno delovanje while zanke.

Začnemo z ustvarjanjem spremenljivke korak s korak = 0. Nato ustvarimo while zanko, ki preveri, če je vrednost spremenljivke korak manjša od 10.000 (korak < 10000), kar je skupno število korakov od dna gore do vrha. Dokler je korak manjši od 10.000, bo Python izvršil ostalo kodo.

S print(korak) izpišemo vrednost spremenljivke in nato preverimo, ali je vrednost spremenljivke odmor True z if utrujen == True:. (True se imenuje logična vrednost, ki jo bomo izvedeli približno v 8. poglavju.) Če je, uporabimo ključno besedo break, da končamo zankanje. Ključna beseda break je način takojšnjega izhoda iz zanke (z drugimi besedami, ustavi zanko), in deluje tako v while kot for zankah. Tu ima učinek izhoda iz bloka in bi se premaknili h katerim koli ukazom, ki bi se pojavili po vrstici korak = korak + 1.

Vrstica elif slabovreme == True: preveri, če je spremenljivka slabovreme nastavljena na True. Če je tako, z break končamo zanko. Če niti utrujen niti slabovreme nista resnična (else), dodamo 1 spremenljivki korak s korak = korak + 1 in zanka se nadaljuje.

Koraki while zanke so naslednji:

1. Preveri pogoj.
2. Izvedi kodo v bloku.
3. Ponovi.

Najpogosteje se uporablja z več pogoji kot le z enim:

```
(1) >>> x = 45  
(2) >>> y = 80  
(3) >>> while x < 50 and y < 100:  
    x = x + 1  
    y = y + 1  
    print(x, y)
```

Tu ustvarimo spremenljivko x z vrednostjo 45 (1) in spremenljivko y z vrednostjo 80 (2). Zanka preverja dva pogoja (3): ali je x manjše od 50 in ali je y manjše od 100.

Če sta oba pogoja resnična, se izvajajo vrstice, ki sledijo, dodajanje 1 obema spremenljivkama in nato izpis. Tukaj je rezultat te kode:

46 81
47 82
48 83
49 84
50 85

Ali lahko ugotovite, kako to deluje?

Začetek štetja pri 45 za spremenljivko x in pri 80 za spremenljivko y. Sledi povečanje (dodajte 1 vsaki spremenljivki) vsakič, ko se koda v zanki izvaja. Zanka bo trajala, dokler bo x manjši od 50 in y manjši od 100. Po peti izvedbi zanke (1 se vsakič doda vsaki spremenljivka), vrednost x doseže 50. Sedaj prvi pogoj ($x < 50$) ni več resničen, zato Python ve, da je treba zanko ustaviti.

While zanke pogosto uporabljamo za pol-neskončne zanke. To je vrsta zanke, ki bi lahko trajala večno, če v kodi ne poskrbimo za prekinitev.

Tukaj je primer:

```
while True:
    veliko kode tukaj
    veliko kode tukaj
    veliko kode tukaj
    if nekaj == True:
        break
```

Pogoj za zanko je samo True, kar je vedno res, zato se bo koda v bloku vedno izvajala (zato je zanka večna). Samo, če je spremenljivka nekaj resnična, bo Python zaključil zanko.

Kaj ste se naučili

V tem poglavju smo uporabili zanke za izvajanje ponavljajočih se opravil. Pythonu smo povedali, kaj želimo ponavljati z ukazi znotraj blokov kode, ki jih postavimo v zanke. Uporabili smo dve vrsti zank: for zanke in while zanke, ki so podobne, vendar jih je mogoče uporabiti na različne načine. Uporabili smo tudi ključno besedo break za ustavitev zanke - to je, da se prekine ponavljanje.

Vaje

Tukaj je nekaj primerov zank, ki jih lahko sami preizkusite. Odgovori so na voljo na <https://nostarch.com/pythonforkids>.

1: Zanka pozdravljen

Kaj menite, da bo naredila naslednja koda? Najprej, uganite kaj se bo zgodilo, nato pa zaženite kodo v Pythonu, da vidite, ali ste mislili pravilno.

```
>>> for x in range(0, 20):
    print('Pozdravljen %s' % x)
    if je x < 9:
        break
```

2: Parne in neparne številke

Ustvarite zanko, ki natisne parna število, dokler ne doseže leta vaše starosti ali, če je vaša starost neparno število, izpišete neparne številke dokler ne doseže vaših let. Na primer, lahko natisne nekaj podobnega:

```
2
4
6
8
10
12
14
```

3: Mojih pet najljubših sestavin

Ustvarite seznam, ki vsebuje pet različnih sestavin sendviča kot sledi:

```
>>> sestavine = ['polži', 'pijavke', 'gorilji popek', 'gusarske obrvi',
'prst stonoge']
```

Zdaj ustvarite zanko, ki natisne seznam (vključno s števili):

```
1 polži
2 pijavke
3 gorilji popek
4 gusarske obrvi
5 prst stonoge
```

4: Vaša teža na Luni

Če bi zdaj stali na luni, bi bila vaša teža 16.5 odstotka tega, kar je na Zemlji. To lahko izračunate tako, da pomnožite svojo težo na Zemlji z 0.165. Če bi v naslednjih 15 letih vsako leto pridobili kilogram teže, kakšna bi bila vaša teža na Luni vsako leto in po 15 letih? Napišite program z uporabo for zanke, ki natisne težo na Luni za vsako leto.

7. poglavje: Predelava kode s funkcijami in moduli

Pomislite, koliko stvari vržete vsak dan stran: steklenice, plastenke, pločevinke, vrečke za čips, folijo za sendviče, nakupovalne vrečke, časopise, revije in tako naprej. Sedaj pa si predstavljajte, da vso to goro smeti dobite na dvorišče, ne da bi ločili papir, plastiko in kovino.

Seveda verjetno reciklirate, kolikor je mogoče, kar je dobro, ker nihče ne želi hoditi čez goro smeti v šolo. Namesto ležanja v ogromnem kupu smeti, se steklenice stopijo v peči in preoblikujejo v nove kozarce in steklenice; papir se zmelje in naredi recikliran papir; plastika se spremeni v nove plastične izdelke. Na ta način lahko ponovno uporabimo stvari, ki bi jih sicer zavrgli.

V programskem svetu je ponovna uporaba prav tako pomembna. Očitno vaš program ne bo izginil pod kupom smeti. Vendar, če ne boste ponovno uporabili delov, ki ste jih že naredili, boste imeli presneto boleče prste od tipkanja. Ponovna uporaba krajša vašo kodo in jo dela lažje berljivo.

Kot boste izvedeli v tem poglavju, Python ponuja številne različne načine ponovne uporabe kode.



Uporaba funkcij

Enega od načinov recikliranja Python kode ste že videli. V prejšnjem poglavju smo uporabili funkciji `range` in `list` za štetje.

```
>>> list(range(0, 5))
[0, 1, 2, 3, 4]
```

Če veste, kako se šteje, ni težko ustvariti seznama zaporednih števil, tako da jih sami vtipkate, vendar večji kot je seznam, več tipkanja imate. Če pa uporabljate funkcije, lahko preprosto ustvarite seznam s tisoči števil.

Tukaj je primer, ki uporablja funkciji `list` in `range` za izdelavo seznama števil:

```
>>> list(range(0, 1000))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ..., 997, 998, 999]
```

Funkcije so delčki kode, ki Pythonu pove, da naj nekaj naredi. Na nek način so ponovna uporaba kode – funkcije lahko v programih uporabite znova in znova.

Ko pišete preproste programe, so funkcije priročne. Ko začnete pisati dolge, bolj zapletene programe, na primer igre, so funkcije neobhodne (ob predpostavki, da želite končati pisanje vašega programa še v tem stoletju).

Deli funkcije

Funkcija ima tri dele: ime, parametre in telo. Tukaj je primer preproste funkcije:

```
>>> def testfunc(moje_ime):
    print('Zdravo %s' % moje_ime)
```

Ime te funkcije je `testfunc`. Ima en sam parameter, `moje_ime`, in njegovo telo je blok kode, ki takoj sledi vrstici z `def` (krajše za definicija). Parameter je spremenljivka, ki obstaja samo med uporabo funkcije. Funkcijo lahko zaženete tako, da pokličete njeno ime z uporabo oklepajev okoli vrednosti parametra:

```
>>> testfunc('Micka')
Zdravo Micka
```

Funkcija ima lahko dva, tri ali poljubno število parametrov, namesto le enega:

```
>>> def testfunc(ime, priimek):
    print('Zdravo %s %s' % (ime, priimek))
```

Obe vrednosti za te parametre ločimo z vejico:

```
>>> testfunc('Micka', 'Kovačova')
Zdravo Micka Kovačova
```

Najprej bi lahko ustvarili nekaj spremenljivk in funkcijo poklicali z njimi:

```
>>> ime = 'Francl'
>>> priimek = 'Popndekl'
>>> testfunc(ime, priimek)
Zdravo Francl Popndekl
```

Funkcija se pogosto uporablja za izračun vrednosti z uporabo stavka `return`. Na primer, lahko napišete funkcijo za izračun količine denarja, ki ste ga prihranili:

```
>>> def prihranek(zepnina, dohodki, poraba):
    return zepnina + dohodki - poraba
```

Ta funkcija ima tri parametre. Sešteje prva dva (`zepnina` in `prihodki`) in odšteje zadnjo (`poraba`). Rezultat se vrne in se lahko dodeli spremenljivki (na enak način kot nastavljamo druge vrednosti spremenljivk) ali izpišemo:

```
>>> print(prihranek(10, 10, 5))
15
```

Spremenljivke in področje uporabe

Spremenljivke znotraj telesa funkcije ni mogoče ponovno uporabiti, ko se funkcija konča, ker obstaja samo znotraj funkcije. V svetu programiranja se to imenuje področje uporabe (`scope`). Poglejmo si preprosto funkcijo, ki uporablja nekaj spremenljivk, vendar nima nobenih parametrov:

```
(1) >>> def test_spremenljivk():
    prva_spremenljivka = 10
    druga_spremenljivka = 20
(2) return prva_spremenljivka * druga_spremenljivka
```

V tem primeru ustvarimo funkcijo, imenovano `test_spremenljivk`, pri (1), ki pomnoži dve spremenljivki (`prva_spremenljivka` in `druga_spremenljivka`) in vrne rezultat (2).

```
>>> print(test_spremenljivk())
200
```


Če to funkcijo kličemo z izpisom, dobimo rezultat: 200. Če poskusimo natisniti vsebino `prva_spremenljivka` (ali `druga_spremenljivka`) izven bloka kode v funkciji, dobimo sporočilo o napaki:

```
>>> print(prva_spremenljivka)
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    print(prva_spremenljivka)
NameError: name 'prva_spremenljivka' is not defined
```

Če je spremenljivka določena zunaj funkcije, ima drugačno področje uporabe. Na primer, določimo spremenljivko, preden jo ustvarimo s funkcijo in jo poskusimo uporabiti znotraj funkcije:

```
(1) >>> tretja_spremenljivka = 100
>>> def test_spremenljivk2():
    prva_spremenljivka = 10
    druga_spremenljivka = 20
(2) return prva_spremenljivka * druga_spremenljivka * tretja_spremenljivka
```



V tej kodi, čeprav spremenljivk `prva_spremenljivka` in `druga_spremenljivka` ni mogoče uporabiti zunaj funkcije, spremenljivko `tretja_spremenljivka` (ki je bila ustvarjena zunaj funkcije (1)) lahko uporabljamo znotraj funkcije (2). Tu je rezultat klicanja te funkcije:

```
>>> print(test_spremenljivk2())
20000
```

Zdaj pa si zamislite, da bi zgradili vesoljsko ladja iz nečesa varčnega, kot so odpadne pločevinke. Na teden bi lahko poravnali 2 pločevinki, da bi ustvarili ukrivljene stene vaše vesoljske ladje, ampak potrebovali boste približno 500 pločevink za dokončanje trupa. Z lahkoto napišite funkcijo, ki vam bo pomagala ugotoviti, koliko časa bo potrebno, da se izravna 500 pločevink pri dveh na teden.

Ustvarimo funkcijo za izračun števila pločevink v enem letu. Naša funkcija bo prevzela število pločevink kot parameter:

```
>>> def gradnja_rakete(plocevinke):
    plocevink = 0
    for teden in range(1, 53):
        plocevink = plocevink + plocevinke
        print('Teden %s = %s pločevink' %(teden, plocevink))
```

V prvi vrstici funkcije ustvarimo spremenljivko `plocevink` in jo nastavimo na vrednost 0. Nato ustvarimo zanko za tedne v letu in dodajamo število pločevink, ki jih je treba vsak dan izravnati. Ta blok kode predstavlja vsebino naše funkcije. Toda tu je tudi drugi blok kode v tej funkciji: zadnje dve vrstici, ki sestavljata blok `for` zanke. Poskusimo vnesti to funkcijo v lupino in jo poklicati z različno vrednostjo za število pločevink:

```
>>> gradnja_rakete(2)
Teden 1 = 2 pločevink
Teden 2 = 4 pločevink
Teden 3 = 6 pločevink
Teden 4 = 8 pločevink
Teden 5 = 10 pločevink
Teden 6 = 12 pločevink
Teden 7 = 14 pločevink
Teden 8 = 16 pločevink
Teden 9 = 18 pločevink
Teden 10 = 20 pločevink
(se nadaljuje ...)
```

```
>>> gradnja_rakete(13)
Teden 1 = 13 pločevink
Teden 2 = 26 pločevink
Teden 3 = 39 pločevink
Teden 4 = 52 pločevink
Teden 5 = 65 pločevink
(se nadaljuje ...)
```

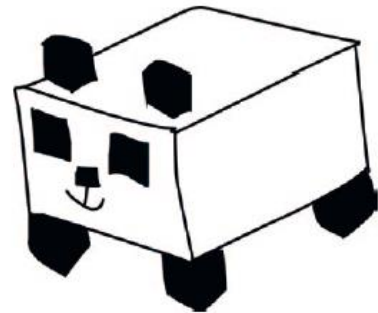
To funkcijo lahko ponovno uporabite z različnimi številkami pločevink na teden, kar je malo bolj učinkovito kot vsakič ponoviti zanko, ko jo želite preizkusiti z različnimi številkami.

Funkcije lahko združimo tudi v module, kar naredi Python resnično uporaben, v nasprotju z le blago uporaben.

Uporaba modulov

Moduli se uporabljajo za združevanje funkcij, spremenljivk in drugih stvari skupaj v večje, močnejše programe. Nekateri moduli so vgrajeni v Python, druge lahko prenesete ločeno. Našli boste module za pisanje iger (npr: tkinter, ki je vgrajen, in PyGame, ki ni), module za obdelavo slik (na primer PIL, Python Imaging Library) in module za risanje tridimenzionalnega grafike (na primer Panda3D).

Moduli se lahko uporabljajo za vse vrste uporabnih stvari. Na primer, če bi načrtovali simulacijsko igro in bi si želeli, da se svet v igri spreminja realno, bi lahko izračunavali aktualni datum in uro z uporabo vgrajenega modula, imenovanega time:



```
>>> import time
```

Tukaj je ukaz import uporabljen, da povemo Pythonu, da želimo uporabiti modul time.

Nato lahko kličemo funkcije, ki so na voljo v tem modulu, s simbolom pike. (Se spomnite, da smo uporabili takšne funkcije pri delu z želvjo grafiko v 4. poglavju, npr. t.forward (50).) Oglejmo si primer klicanja asctime funkcije v modulu time:

```
>>> print(time.asctime())
'Mon Nov 5 12:40:27 2012'
```

Funkcija asctime je del modula time, ki vrne trenutni datum in čas, kot niz.

Sedaj pa si zamislimo, da v programu želimo, da uporabnik vnese kakšno vrednost, morda njegov datum rojstva ali starost. To lahko naredimo z izpisom sporočila in modulom sys (kratko za sistem), ki vsebuje pripomočke za izmenjavo s Pythonom. Najprej uvozimo modul sys:

```
>>> import sys
```

V modulu sys je poseben objekt, imenovan stdin (za standardni vhod), ki zagotavlja precej uporabno funkcijo readline. Funkcija readline se uporablja za branje vrstice vnesenega besedila na tipkovnici, dokler ne pritisnete enter. (kako delujejo objekti bomo pogledali v 8. poglavju) Za preizkušanje readline vnesite naslednjo kodo v lupino:

```
>>> import sys
>>> print(sys.stdin.readline())
```

Če nato vnesete nekaj besed in pritisnete enter, se bodo te besede izpisale v lupini.

Spomnite se na kodo, ki smo jo zapisali v 5. poglavju, z uporabo if stavka:

```
>>> if starost >= 10 and starost <= 13:
    print('Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!')
else:
    print("Uf?")
```

Namesto ustvarjanja spremenljive starost in nastavitve določene vrednost pred if stavkom, lahko sedaj prosimo za vnos vrednosti. Ampak najprej kodo spremenimo v funkcijo:

```
>>> def neumen_vic_starosti(starost):
    if starost >= 10 and starost <= 13:
        print('Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!')
    else:
        print('Uf?')
```

Zdaj lahko pokličete funkcijo tako, da vnesete njeno ime in nato poveste, katero število uporabiti z vnosom števila v oklepaju. Ali deluje?

```
>>> neumen_vic_starosti(9)
Uf?
>>> neumen_vic_starosti(10)
Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!
```

Deluje! Sedaj pa v funkciji vprašajmo za starost osebe. (Funkcijo lahko spremenite kadarkoli želite.)

```
>>> def neumen_vic_starosti():
    print('Koliko si star?')
    (1) starost = int(sys.stdin.readline())
    (2) if starost >= 10 and starost <= 13:
        print('Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!')
    else:
        print('Uf?')
```

Ali ste prepoznali funkcijo int (1), ki pretvori niz v število? To funkcijo smo vključili, ker readline() vrne niz, mi pa želimo številko, ki jo lahko primerjamo s števili 10 in 13 (2).

Poskusiti sami, pokličite funkcijo brez kakršnih koli parametrov in nato po vprašanju Koliko si star? vpišete številko:

```
>>> neumen_vic_starosti()
Koliko si star?
10
Kaj je 13 + 49 + 84 + 155 + 97? Glavobol!
>>> neumen_vic_starosti()
Koliko si star?
15
Uf?
```

Kaj ste se naučili

V tem poglavju ste videli, kako s funkcijami narediti kodo za večkratno uporabo v Pythonu in kako uporabljati funkcije, ki jih ponujajo moduli. Spoznali ste področje uporabe spremenljivk znotraj ali zunaj funkcij in kako ustvariti funkcije z ukazom def. Naučili ste se tudi uvoziti module in uporabiti njihovo vsebino.

Vaje

Poskusite rešiti naslednje primere z ustvarjanjem lastnih funkcij. Odgovori so na voljo na <https://nostarch.com/pythonforkids>.

1: Osnovna funkcija za težo na Luni

V 6. poglavju je bila ena vaja z izdelavo zanke za določitev teže na Luni v obdobju 15 let. Ta for zanka se lahko enostavno pretvori v funkcijo. Poskusite ustvariti funkcijo, ki sprejme začetno težo in spremembo teže vsako leto. Novo funkcijo lahko pokličete z uporabo kode, kot je ta:

```
>>> teza_na_luni(30, 0.25)
```

2: funkcija teže na Luni in leta

Prejšnjo funkcijo izboljšajte tako, da boste s parametrom lahko nastavljali tudi število let, na primer 5 let ali 20 let:

```
>>> teza_na_luni(90, 0.25, 5)
```

3: Program za težo na Luni

Namesto preproste funkcije, kjer vnesete vrednosti kot parametre, lahko naredite programček, ki zahteva vrednosti z uporabo `sys.stdin.readline()`. V tem primeru pokličete funkcijo brez kakršnih koli parametrov:

```
>>> teza_na_luni()
```

Funkcija bo prikazala sporočilo za vnos začetne teže nato drugo sporočilo, ki zahteva vnos povečanja teže na leto in končno še sporočilo, ki zahteva število let. Nastalo naj bi nekaj podobnega:

```
Vnesite svojo trenutno težo na Zemlji
45
Prosimo, vnesite znesek, ki se lahko poveča vsako leto
0.4
Vnesite število let
12
(izpis rezultata)
```

Ne pozabite najprej uvoziti modula `sys`, preden ustvarite funkcijo:

```
>>> import sys
```

8. poglavje: Kako uporabljati razrede in objekte

Kaj imata skupnega žirafa in pločnik? Oba sta stvar. Iz slovnice znana tudi kot samostalnik, v Pythonu pa objekt.

Ideja o objektih je pomembna v svetu računalnikov. Objekti so način organiziranja kode v program in razbijanje stvari na manjše dele, da bi olajšali razmišljanje o zapletenih idejah. (Objekt smo uporabili v 4. poglavju, ko smo delali z želvo - Pen.)

Če želite resnično razumeti, kako objekti delujejo v Pythonu, moramo razmisliti o vrstah objektov. Začnimo z žirafami in pločniki.

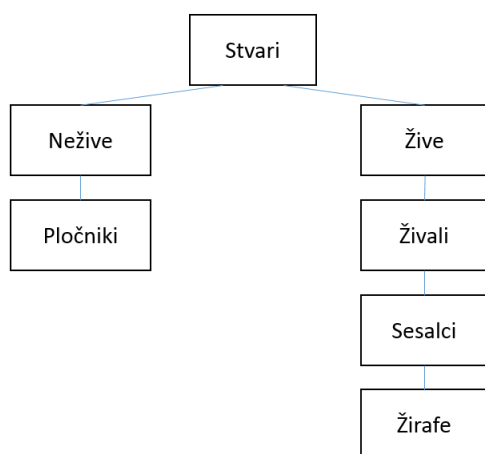
Žirafa je vrsta sesalca, ki je vrsta živali. Žirafa je tudi premikajoč se predmet – torej je živa.

Sedaj pa razmislimo o pločniku. Ni veliko povedati o pločniku, razen tega, da to ni živa stvar. Recimo, da je nepremičen objekt (z drugimi besedami, ni živ). Izrazi sesalec, žival, premikanje in nepremičnost so vsi načini razvrščanja stvari.



Razbijanje stvari v razrede

V Pythonu so objekti definirani po razredih, ki jih lahko zamislimo kot način razvrstitve objektov v skupine. Tukaj je diagram drevesa razredov, da bi žirafe in pločniki ustrezali našim predhodnim definicijam:



Glavni razred je Stvari. Pod razredom Stvari imamo Nežive in Žive. Te se nadalje delijo na Pločnike pri Neživih in Živali, Sesalce in Žirafe pri Živih.

Za organiziranje Python kode lahko uporabimo razrede. Za primer vzemimo modul turtle. Vse stvari, ki jih ima Pythonov modul turtle lahko nekaj naredijo – na primer premik naprej, premik nazaj, vrtenje v levo in vrtenje v desno – to so funkcije v razredu (class) Pen. Objekt se smatra kot pripadnik razreda in ga lahko ustvarimo poljubno število krat.

Ustvarimo sedaj isti nabor razredov, kot je prikazano na našem drevesnem diagramu, začeni z vrha. Razred ustvarimo z rezervirano besedo `class`, ki ji sledi ime z veliko začetnico (splošni dogovor). Ker so `Stvari` najvišji razred, najprej ustvarimo tega:

```
>>> class Stvari:
    pass
```

Imenujemo razred `Stvari` in uporabimo ukaz `pass`, da Python ve, da tu ne bomo še nič delali. `pass` se uporablja, če želimo definirati razred ali funkcijo, vendar trenutno še ne vemo podrobnosti.

Nato bomo dodali še druge razrede in jim dodali razmerja.

Otroci in starši

Če je razred del drugega razreda, potem je otrok tega razreda, drugi razred pa je starš. Razredi so lahko otroci in starši do drugih razredov. V našem drevesnem diagramu je razred nad razredom njegov starševski razred, razred pod njim pa je njegov otrok. Na primer: `Neživi` in `Živi` sta otroka razreda `Stvari`, kar pomeni da je `Stvari` njun starš.

Da povemo Pythonu, da je razred otrok drugega razreda, dodamo imenu razreda v oklepajih še ime starševskega razreda, takole:

```
>>> class Nežive(Stvari):
    pass
>>> class Žive(Stvari):
    pass
```

Tu ustvarimo razred `Nežive` in povemo Pythonu, da je `Stvari` njen starševski razred s kodo `class Nežive(Stvari)`. Naprej ustvarimo razred imenovan `Žive`, ki pove Pythonu, da je njen starš razred `Stvari`, s kodo `class Žive(Stvari)`.

Naredimo enako z razredom pločnikov. Ustvarimo razred `Pločniki` s starševskim razredom `Nežive`:

```
>>> class Pločniki(Nežive):
    pass
```

V razrede lahko organiziramo tudi živali, sesalce in žirafe z uporabo njihovih starševskih razredov:

```
>>> class Živali(Žive):
    pass
>>> class Sesalci(Živali):
    pass
>>> class Žirafe(Sesalci):
    pass
```

Dodajanje objektov v razrede

Sedaj imamo veliko razredov, kako pa kaj dodamo v te razrede? Recimo, da imamo žirafu po imenu `Reginald`. Vemo, da pripada razredu `Žirafe`, ampak kako pa uporabljamo v programskih izrazih posamezno žirafu, imenovano `Reginald`? `Reginald` imenujemo object (objekt) razreda `Žirafe` (morda boste videli tudi izraz `instance`). Da bi "predstavili" `Reginald` Pythonu, uporabimo ta delček kode:

```
>>> reginald = Žirafe()
```

Ta koda pove Pythonu naj ustvari objekt v razredu Zirafe in ga dodeli spremenljivki reginald. Podobno kot pri funkciji imenu razreda sledijo oklepaji. Kasneje v tem poglavju bomo videli kako ustvariti objekte in uporabljati parametre v oklepajih.

Toda kaj objekt reginald dela? No, za sedaj še nič. Da bi bili naši objekti uporabni moramo pri ustvarjanju razredov definirati funkcije, ki jih lahko uporabimo z objekti v tem razredu. Namesto, da uporabite ključno besedo `pass` po opredelitvi razreda, dodamo definicije funkcij.

Definiranje funkcij v razredih

V 7. poglavju smo spoznali funkcije kot način ponovne uporabe kode. Ko določimo funkcijo, ki je povezana z razredom, to storimo na enak način kot katero koli drugo funkcijo, razen da jo zamaknemo pod definicijo razreda. Na primer, tukaj je običajna funkcija, ki ni povezana z razredom:

```
>>> def to_je_normalna_funkcija():
    print("Sem normalna funkcija")
```

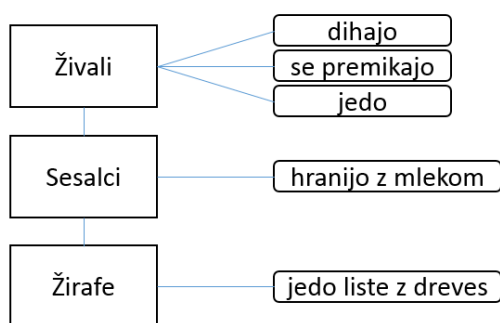
In tukaj sta funkciji, ki spadata v razred:

```
>>> class ToJeMojCudenRazred:
    def to_je_funkcija_v_razredu():
        print('Jaz sem razredna funkcija')
    def to_je_tudi_funkcija_v_razredu():
        print('Jaz sem tudi funkcijo razreda. Vidiš?')
```

Dodajanje značilnosti razreda kot funkcij

Spomnimo se otroških razredov razreda Zive. Vsakemu razredu lahko dodamo značilnosti za opis kaj je in kaj lahko dela. Značilnost je lastnost, ki jo imajo vsi člani razreda (tudi otroci).

Na primer, kaj imajo vse živali skupnega? Če začnemo: vsi dihajo. Prav tako se premikajo in jedo. Kaj pa sesalci? Sesalci svoje mladiče hranijo z mlekom. In dihajo, se premikajo in jedo. Vemo, da žirafe jedo listje visoko v drevesih in tako kot vsi sesalci, s svojim mlekom hranijo mladiče, dihajo, se premikajo in jedo hrano. Ko dodamo te značilnosti v naš drevesni diagram, dobimo nekaj takega:



Te značilnosti je mogoče razumeti kot dejanja ali funkcije - stvari, ki jih objekt tega razreda lahko naredi.

Če želite dodati funkcijo v razred, uporabimo ključno besedo `def`. Torej razred `Zivali` bo izgledal takole:

```
>>> class Zivali(Zive):
    def dihaj(self):
```

```

    pass
def premakni(self):
    pass
def jej(self):
    pass

```



V prvi vrstici tega vnosa definiramo razred, kot smo to storili prej, vendar namesto pass v naslednji vrstici, določimo funkcijo dihaj in ji damo en parameter: self. self parameter je način, da eno funkcija v razredu lahko pokliče drugo funkcijo v razredu (in v nadrejenem razredu). To uporabo parametrov bomo videli kasneje.

V naslednji vrstici ključna beseda pass pove Pythonu, da še ne vemo kaj bo funkcija delala. Nato dodamo funkcijo premakni in jej, ki za zdaj tudi še ne delata ničesar. Kmalu bomo preoblikovali razrede in v funkcije postavili ustrezno kodo. To je pogost način razvoja programov. Pogosto programerji ustvarijo razrede s funkcijami, ki ne delajo ničesar. Na ta način ugotovijo, kaj naj razred sploh dela. Nato se lotijo podrobnosti posameznih funkcij.

Prav tako lahko dodamo funkcije drugima dvema razredoma Sesalci in Zirafe. Vsak razred bo lahko uporabil značilnosti (funkcije) svojega starševskega razreda. To pomeni, da vam ni treba narediti enega zapletenega razreda; svoje funkcije lahko daste v najvišji razred s temi značilnostmi. (To je dober način za preprostejše in lažje razumljive razrede.)

```

>>> class Sesalci(Zivali):
    def hrani_mladike_z_mlekom(self):
        pass
>>> class Zirafe(Sesalci):
    def je_liste_z_dreves(self):
        pass

```

Zakaj uporabljati razrede in objekte?

Dodali smo funkcije v naše razrede, toda zakaj sploh uporabljati razrede in objekte, ko bi lahko le napisali običajne funkcije, imenovane dihaj, premakni, jej in tako naprej? Da odgovorimo na to vprašanje, bomo uporabili našo žirafa, imenovano Reginald, ki smo jo prej ustvarili kot objekt razreda Zirafe:

```
>>> reginald = Zirafe()
```

Ker je reginald objekt, lahko pokličemo (ali izvajamo) funkcije njegovega razreda (razred Zirafe) in njegovih starševskih razredov. Funkcije na objektu kličemo z uporabo operatorja pika in imena funkcije. Če želimo žirafi Reginald reči naj se premakne ali je, lahko pokličemo funkcije, kot je ta:

```

>>> reginald = Zirafe()
>>> reginald.premakni()
>>> reginald.je_listje_z_dreves()

```

Recimo, da ima Reginald prijatelja žirafa po imenu Harold. Ustvarimo drugi žirafski objekt, imenovan harold:

```
>>> harold = Zirafe()
```

Ker uporabljamo objekte in razrede, lahko povemo Python točno za katero žirafa govorimo, ko želimo pognati funkcijo premakni. Na primer, če bi želeli, da se Harold premakne in pustili Reginalda na mestu, lahko uporabimo našo funkcije premakni na objektu harold, kot sledi:

```
>>> harold.premakni()
```


V tem primeru bi se premaknil samo Harold.

Naj malo spremenimo razrede, da bi bilo to bolj očitno. Za vsako funkcijo bomo dodali izpis na zaslon, namesto da bi uporabili pass:

```
>>> razred Zivali(Zive):
    def dihaj(self):
        print("dihanje")
    def premakni(self):
        print("premikanje")
    def jej(self):
        print("je hrano")
>>> razred Sesalci(Zivali):
    def hrani_mladice_z_mlekom(self):
        print("hranjenje mladih")
>>> razred Zirafe(Sesalci):
    def je_listje_z_dreves(self):
        print("je liste z dreves")
```



Sedaj, ko ustvarimo naše objekte reginalda in harolda in pokličemo njune funkcije, lahko vidimo, kaj se dejansko zgodi:

```
>>> reginald = Zirafe()
>>> harold = Zirafe()
>>> reginald.premakni()
premikanje
>>> harold.je_listje_z_dreves()
je liste z dreves
```

V prvih dveh vrsticah ustvarimo spremenljivki reginald in harold, ki sta objekta razreda Zirafe. Nato kličemo funkcijo premakni za reginalda in Python izpiše premikanje v naslednji vrstici. Na enak način kličemo funkcijo je_listje_z_dreves za harolda in Python izpiše je liste z dreves. Če bi bile to resnično žirafe, ne pa predmeti v računalniku, bi ena žirafa hodila, druga pa bi jedla liste z dreves.

Objekti in razredi v slikah

Kako pa, če bolj grafično pristopili k objektom in razredom?

Vrnimo se k modulu turtle, s katerim smo se igrali v 4. poglavju. Ko uporabljamo turtle.Pen(), Python ustvari objekt razreda Pen, ki ga zagotavlja modul turtle (podobno kot naša objekta reginald in harold). Ustvarimo lahko dva objekta turtle (imenovana Avery in Kate), prav tako kot smo ustvarili dve žirafi:

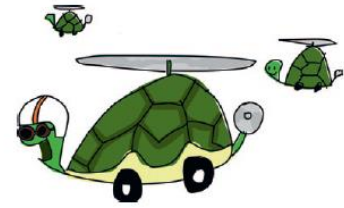
```
>>> import turtle
>>> avery = turtle.Pen()
>>> kate = turtle.Pen()
```

Vsaka želva (avery in kate) je član razreda Pen.

Tukaj objekti postanejo močni. Ob ustvarjenih dveh objektih, lahko kličemo funkcije na vsakem od njih in risanje bo potekalo neodvisno. Poskusite to:

```
>>> avery.forward(50)
>>> avery.right(90)
>>> avery.forward(20)
```

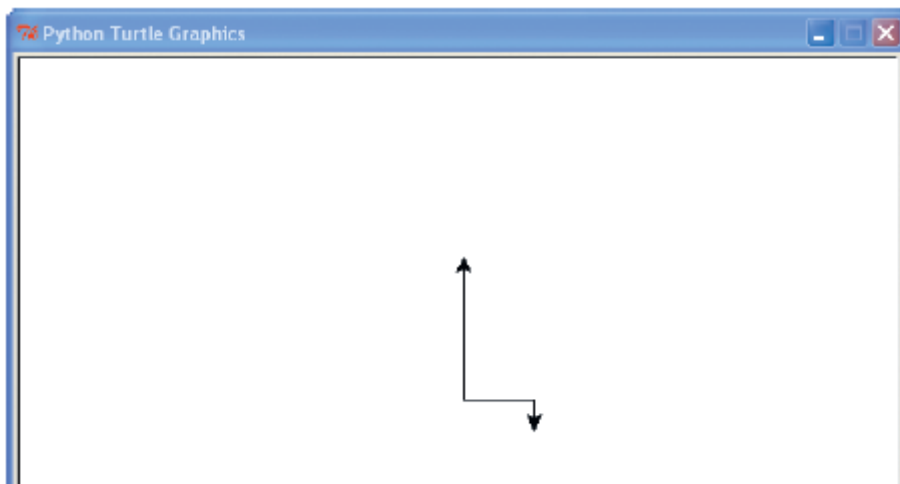
S to serijo navodil povemo, da se Avery premika naprej 50 pik, zavije desno za 90 stopinj in se premakne naprej za 20 pik ter konča obrnjen navzdol. Zapomni si, da želve vedno začnejo obrnjene na desno. Zdaj je čas, da premaknemo Kate.



```
>>> kate.left(90)
>>> kate.forward(100)
```

Kate povemo, naj zavije levo za 90 stopinj, se nato premakne naprej 100 pik in konča obrnjena navzgor.

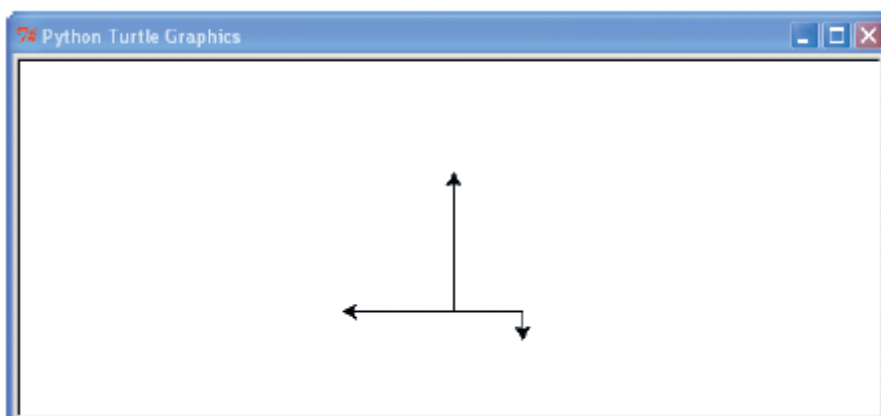
Doslej imamo različni črti s puščicama, ki se premikata v različnih smereh: Avery gleda navzdol in Kate gleda navzgor.



Dodajmo še eno želvo, Jakoba, in ga premaknimo brez da bi motil ostala dva.

```
>>> jacob = turtle.Pen()
>>> jacob.left(180)
>>> jacob.forward(80)
```

Najprej ustvarimo nov Pen objekt, imenovan jacob, nato ga obrnemo levo 180 stopinj in ga premaknemo naprej za 80 pik. Naša risba sedaj izgleda tako, s tremi želvami:



Ne pozabite, da vsakič, ko pokličemo `turtle.Pen()`, da ustvarimo želvo, dodamo nov, neodvisen objekt. Vsak objekt je še vedno instanca razreda `Pen` in na njih lahko uporabljamo iste funkcije za vsak objekt, ker pa uporabljamo objekte, lahko premikamo vsako želvo neodvisno. Tako kot naši neodvisni objekti žiraf (Reginald in Harold), so Avery, Kate in Jacob neodvisni objekti želv. Če

ustvarimo nov objekt z enakim imenom spremenljivke kot objekt, ki smo ga že ustvarili, ni nujno, da stari objekti izginejo. Poskusite sami: ustvarite še eno želvo Kate in jo poskusite premikati.

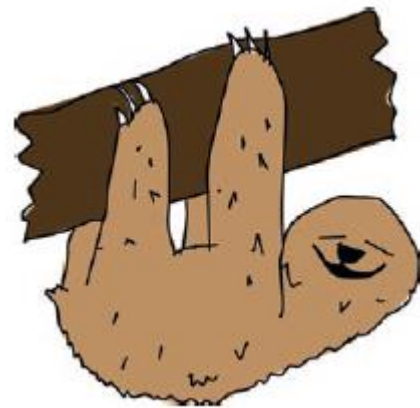
Druge uporabne lastnosti objektov in razredov

Razredi in predmeti olajšajo združevanje funkcij. So tudi resnično uporabni, če želimo razmišljati o programu v manjših delih.

Zamislite si na primer res veliko programsko aplikacijo, kot je urejevalnik besedil ali 3D računalniška igra. Za večino ljudi je skoraj nemogoče razumeti velike programe, kot celoto, ker gre za tako veliko kodo. Če pa te glomazne dele programa razdelite na manjše dele, ima vsak del določen smisel - če obvladaš jezik, seveda!

Pri pisanju velikega programa vam to omogoča tudi razdeliti delo v skupine programerjev. Najbolj zapleteni programi, ki jih uporabljate (na primer spletni brskalnik) so napisali mnogi ljudje ali skupine ljudi, ki delajo na različnih delih istočasno po vsem svetu.

Zdaj si predstavljajte, da želimo razširiti nekatere razrede, ki smo jih ustvarili v tem poglavju (Zivali, Sesalci, in Zirafe), vendar imamo preveč dela in želimo, da nam pomagajo prijatelji. Delo bi lahko razdelili, tako da ena oseba programira razred Zivali, druga razred Sesalci in še ena razred Zirafe.



Podedovane funkcije

Tisti, ki ste pozorni, ste verjetno opazili, da bo na boljšem tisti, ki bo delal na razredu Zirafe, saj bo imel na voljo vse funkcije predhodnih razredov. Razred Zirafe podeduje funkcije razredov Zivali in nato Sesalci. Z drugimi besedami, ko ustvarimo objekt žirafa, lahko uporabimo funkcije, definirane v razredu Zirafe kot funkcije, opredeljene v razredih Sesalci in Zivali. In na isti način, če ustvarimo objekt Sesalci, lahko uporabimo funkcije ki so opredeljeni v razredu sesalcev in starševskem razredu Zivali.

Čeprav je Reginald objekt razreda Zirafe, lahko še vedno pokličete funkcijo premakni, ki smo jo določili v razredu Zivali, ker so funkcije, ki so določene v katerem koli starševskem razredu, na voljo svojim otrokom:

```
>>> reginald = Zirafe()
>>> reginald.premakni()
premikanje
```

Dejansko lahko vse funkcije, ki smo jih opredelili v razredu Zivali in v razredu Sesalci, pokličete iz našega predmeta reginald, ker so funkcije podedovane:

```
>>> reginald = Zirafe()
>>> reginald.dihaj()
dihanje
>>> reginald.jej()
jedo hrano
>>> reginald.hrani_mladice_z_mlekom()
```

hranjenje mladih

Funkcije, ki kličejo druge funkcije

Ko pokličemo funkcije na objektu, uporabimo ime spremenljivke objekta. Na primer, tukaj lahko pokličete funkcijo `premakni` za žirafa `Reginald`:

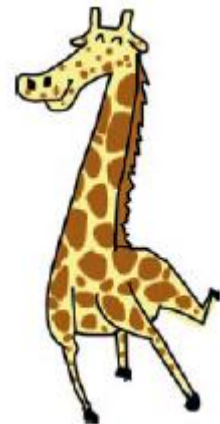
```
>>> reginald.premakni()
```

Če želite imeti funkcijo v razredu `Zirafe`, ki kličejo funkcijo `premakni`, bomo uporabili parameter `self`. Parameter `self` je način, da ena funkcija v razredu pokliče drugo funkcijo. Denimo, da dodamo funkcijo, ki se imenuje `najdi_hrano` v razredu `Zirafe`:

```
>>> class Zirafe(Sesalci):
    def najdi_hrano(self):
        self.premakni()
        print("Našel sem hrano!")
        self.jej()
```

Ustvarili smo funkcijo, ki vključuje dve drugi funkciji. To je precej pogosto pri programiranju. Pogosto boste napisali funkcijo, ki naredi nekaj koristnega in jo boste potem uporabili znotraj druge funkcije. (To bomo naredili v poglavju 13, kjer bomo napisali bolj zapletene funkcije za igro.) Uporabimo `self` in dodajmo še nekaj funkcij v razred `Zirafe`:

```
>>> class Zirafe(Sesalci):
    def najdi_hrano(self):
        self.premakni()
        print("Našel sem hrano!")
        self.jej()
    def je_listje_z_dreves(self):
        self.jej()
    def zaplesi(self):
        self.premakni()
        self.premakni()
        self.premakni()
        self.premakni()
```



Uporabljamo funkcije `jej` in `premakni` iz starševskega razreda `Zivali`, da določimo `je_listje_z_dreves` in `zaplesi` za razred `Zirafe`, ker so to podedovane funkcije. Z dodajanjem funkcije na ta način pokličete druge funkcije in ena funkcija lahko naredi precej več. Poglejte, kaj se zgodi, ko pokličemo funkcijo `zaplesi` - naša žirafa se premakne 4-krat (besedilo "premikanje" se izpiše 4-krat):

```
>>> reginald = Zirafe()
>>> reginald.zaplesi()
premikanje
premikanje
premikanje
premikanje
```

Inicializacija objekta

Včasih pri ustvarjanju predmeta želimo določiti nekaj vrednosti (imenovane tudi lastnosti) za kasnejšo uporabo. Ko inicializiramo objekt, ga pripravimo za uporabo.

Predpostavimo, da želimo nastaviti število `lis` na našem objektu žirafe, ko jo ustvarimo - ko jo inicializiramo. To naredimo s funkcijo `__init__` (na vsaki strani `init` sta po dve podčrti).

To je posebna vrsta funkcije v razredih v Pythonu in mora imeti to ime. Funkcija `init` je način, kako nastavimo lastnosti za objekt, ko je objekt prvič ustvarjen. Python bo samodejno poklical to funkcijo, ko ustvarimo nov objekt. Tukaj je način uporabe:

```
>>> class Zirafe:
    def __init__(self, lise):
        self.lise_zirafe = lise
```

Najprej definiramo funkcijo `init` z dvema parametroma, `self` in `lise` s kodo `def __init__(self, lise):`. Tako kot druge funkcije, ki smo jih definirali v razredu, mora tudi `init` funkcija imeti `self` kot prvi parameter. Nato navedemo parameter `lise` (njeno lastnost). Objektno spremenljivko `lise_zirafe` nastavimo na parameter s kodo `self.lise_zirafe = lise`. To pomeni: "Vzemite vrednost parametra in jo shranite za pozneje (z uporabo spremenljivke objekta `lise_zirafe`)." Prav tako kot ena funkcija v razredu lahko pokliče drugo funkcijo z uporabo parametra `self`, tudi do spremenljivk v razredu dostopamo z uporabo `self`.

Nato ustvarimo nekaj novih objektov žiraf (Ozward in Gertrude) in prikažimo njihovo število lis:

```
>>> ozwald = Zirafe(100)
>>> gertrude = Zirafe(150)
>>> print(ozwald.lise_zirafe)
100
>>> print(gertrude.lise_zirafe)
150
```

Najprej ustvarimo instanco razreda `Zirafe`, ki uporablja vrednost parametra `100`. To ima za posledico klic `__init__` funkcijo in z uporabo `100` za vrednost parametra `lise`. Nato smo ustvarili še eno instanco razreda `Zirafe`, tokrat s `150` lisami. Na koncu izpišemo objektno spremenljivko `lise_zirafe` za vsako od naših žiraf in vidimo, da sta rezultata `100` in `150`. Delovalo je!

Ne pozabite, da lahko pri ustvarjanju objekta razreda, kot na primer `ozwald` zgoraj, dostopamo do spremenljivk ali funkcij z uporabo operatorja pika in ime (na primer, `ozwald.lise_zirafe`). Ko pa ustvarjamo te iste spremenljivke in funkcije znotraj razreda, uporabljamo `self` (`self.lise_zirafe`).

Kaj ste se naučili

V tem poglavju smo uporabili razrede za ustvarjanje skupin stvari in delali objekte (instance) teh razredov. Naučili ste se, kako otrok razreda podeduje funkcije svojega starša in čeprav sta dva objekta istega razreda, nista nujno kloni. Na primer, objekt žirafe ima lahko svoje število lis. Naučili ste se kako klicati (ali izvajati) funkcije v objektu in kako shranjujemo stvari v objektne spremenljivke. Nazadnje smo uporabili `self` parameter v funkcijah, ki se nanašajo na druge funkcije in spremenljivke v razredu. Ti pojmi so temeljnega pomena za Python in videli jih boste znova in znova pri branju preostanka te knjige.

Vaje

Nekatere ideje v tem poglavju bodo bolj smiselne jih uporabite. Preizkusite jih z naslednjimi primeri in nato poiščite odgovore na naslovu <https://nostarch.com/pythonforkids>.

1: Žirafji ples

Dodajte funkcije v razred Zirafe za premikanje leve in desne noge naprej in nazaj. Funkcija premikanja leve noge naprej lahko izgleda takole:

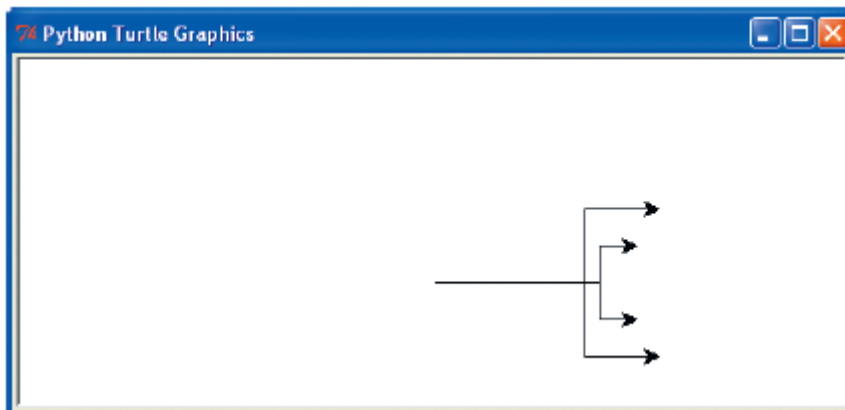
```
>>> def leva_noga_naprej(self):  
    print("levo nogo naprej")
```

Nato ustvarite funkcijo, imenovano ples, da naučite Reginalda plesati (funkcija bo poklicala štiri noge, ki ste jih pravkar ustvarili). Rezultat te nove funkcije bo preprost ples:

```
>>> reginald = Zirafe()  
>>> reginald.ples()  
levo nogo naprej  
levo nogo nazaj  
desno nogo naprej  
desno nogo nazaj  
levo nogo nazaj  
desno nogo nazaj  
desno nogo naprej  
levo nogo naprej
```

2: Želvi štirizob

Ustvarite naslednjo sliko štirizoba s štirimi objekti Pen modula turtle (natančna dolžina linij ni pomembna). Ne pozabite najprej uvoziti modula turtle!



9. poglavje: Pythonove vgrajene funkcije

Python ima dobro založena programska orodja, vključno z velikim številom funkcij in modulov, ki so pripravljeni za uporabo. Kot zanesljivo kladivo ali kolesarski ključ, ta vgrajena orodja – deli kode, res naredijo pisanje programov veliko lažje.

V 7. poglavju ste se naučili, da je treba module pred uporabo uvoziti. Vgrajenih funkcij ni treba predhodno uvažati; na voljo so takoj. V tem poglavju, si bomo ogledali nekaj najbolj uporabnih vgrajenih funkcij in se nato osredotočili na eno: funkcijo `open`, ki vam omogoča, da odprete datoteke za branje in pisanje.

Uporaba vgrajenih funkcij

Ogledali si bomo 12 vgrajenih funkcij, ki jih običajno uporabljajo Python programerji. Videli bomo, kaj delajo in kako jih uporabljati. Pokazali bomo primere, kako lahko pomagajo v vaših programih.

Funkcija `abs`

Funkcija `abs` vrne absolutno vrednost števila, ki je vrednost števila brez njenega predznaka. Na primer, absolutna vrednost 10 je 10, absolutna vrednost -10 pa 10. Če želite uporabiti `abs` funkcijo, jo preprosto pokličite s številko ali spremenljivko kot njenim parametrom, kot tule:

```
>>> print(abs(10))
10
>>> print(abs(-10))
10
```

S funkcijo `abs` lahko na primer izračunate absolutno vrednost premika lika v igri, ne glede na to, v katero smer ta lik potuje. Na primer, da lik naredi tri korake v desno (pozitivno 3) in nato deset korakov v levo (negativno 10, ali -10). Če nam smer ne bi bila važna (pozitivno ali negativno), bi bila absolutna vrednost teh števil 3 in 10. To lahko uporabite v igri na plošči, kjer si mečete dve kocki in nato premaknete svoj lik za največje število korakov v kateri koli smeri, ki temelji na vsoti števil na kockah. Če shranimo število korakov v spremenljivko, s kodo spodaj lahko ugotovimo, če se lik premika. Morda želimo prikazati podatke o premikanju (v tem primeru bomo prikazali samo "Lik se premika"):

```
>>> koraki = -3
>>> if abs(koraki) > 0:
    print('Lik se premika')
```

Če ne bi uporabili `abs`, bi bila koda videti tako:

```
>>> koraki = -3
>>> if koraki < 0 or koraki > 0:
    print('Lik se premika')
```

Kot lahko vidite, z uporabo `abs` naredimo izjavo, ki je krajša in lažje razumljiva.



Funkcija bool

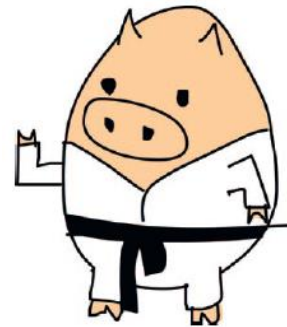
Ime bool je krajše za Boolean, kar programerji uporabljajo za podatke, ki lahko zavzamejo dve vrednosti True ali False (pravilno ali napačno).

Funkcija bool sprejme en parameter in vrne True ali False glede na njegovo vrednost. Pri uporabi bool za številke, 0 vrne False, katera koli druga številka vrne True. Takole lahko uporabite bool z različnimi števili:

```
>>> print(bool(0))
False
>>> print(bool(1))
True
>>> print(bool(1123.23))
True
>>> print(bool(-500))
True
```

Ko za bool uporabljate druge vrednosti, kot so nizi, vrne False, če je niz prazen (z drugimi besedami, None ali " – nobenega znaka med narekovaji). V nasprotnem primeru vrne True, kot je prikazano tukaj:

```
>>> print(bool(None))
False
>>> print(bool('a'))
True
>>> print(bool(' '))
True
>>> print(bool('Kako rečeš prašiču, ki trenira karate? Zrezek!'))
True
```



Funkcija bool bo vrnila False za prazne sezname, tuple in slovarje ali True v ostalih primerih:

```
>>> moj_seznam = []
>>> print(bool(moj_seznam))
False
>>> moj_seznam = ['k', 'r', 'n', 'e', 'k', 'i']
>>> print(bool(moj_seznam))
True
```

Funkcijo bool lahko uporabite tudi za ugotavljanje, če je spremenljivka nastavljena ali ne. Če na primer vprašamo uporabnika naj vnesemo leto rojstva, bi lahko naša izjava uporabila bool za testiranje vrednosti, ki jo vnesejo:

```
>>> leto = input('leto rojstva:')
Leto rojstva:
>>> if not bool(leto.rstrip()):
    print('Vnesti morate leto vašega rojstva')
Vnesti morate leto vašega rojstva
```

Prva vrstica tega primera uporablja funkcijo input za vnos vrednosti preko tipkovnice v spremenljivko leto. S pritiskom na enter v naslednji vrstici (brez tipkanje karkoli drugega) shranimo vrednost enter v spremenljivko. (V 7. poglavju smo uporabili sys.stdin.readline(), kar je drug način za isto stvar.)

V naslednji vrstici, if stavek preveri Boolean vrednost spremenljivke po uporabi funkcije rstrip (ki odstrani vse presledke in enter znake z desne strani niza). Ker uporabnik ni ničesar vnesel, funkcija bool vrne False. Ker ta izjava uporablja ključno besedo not, je to kot, da bi rekli "Storite to, če funkcija NE vrne True", zato se v naslednji vrstici izpiše sporočilo: 'Vnesti morate leto vašega rojstva'.

Funkcija dir

Funkcija `dir` (kratko za `directory`) vrne informacije o katerikoli vrednosti. V bistvu vam v abecednem redu izpiše funkcije, ki jih lahko uporabite. Na primer, če želite prikazati funkcije, ki so na voljo za vrednost seznama, vnesite to:

```
>>> dir(['kratek', 'seznam'])
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__',
 '__getnewargs__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
 '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
 '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
 'sort']
```

Funkcija `dir` deluje na karkoli, vključno z nizi, števkami, funkcijami, moduli, objekti in razredi. Včasih informacije, ki jih vrne, niso najbolj uporabne. Če pokličete `dir` na številko 1, prikaže obilico posebnih funkcij (tistih, ki se začnejo in končajo s podčrto), ki jih uporablja le Python in je za nas nekoristno (večino lahko ignorirate):

```
>>> dir(1)
['_abs_', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
 '__delattr__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__',
 '__floordiv__', '__format__', '__ge__', '__getattr__', '__getnewargs__',
 '__gt__', '__hash__', '__index__', '__init__', '__int__',
 '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__',
 '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__',
 '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
 '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
 '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
 '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
 '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
 'bit_length', 'conjugate', 'denominator', 'imag', 'numerator', 'real']
```

Funkcija `dir` je lahko uporabna, če imate spremenljivko in hitro želite ugotoviti, kaj bi lahko storili z njo. Na primer, zaženite `dir` z uporabo spremenljivke `popcorn`, ki vsebuje niz. Dobite seznam funkcij, ki jih ponuja razred `string` (vsi nizi so člani razreda `string`):

```
>>> popcorn = 'Rad imam pokovko!'
>>> dir(popcorn)
['_add_', '__class__', '__contains__', '__delattr__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__',
 '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
 '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize',
 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format',
 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

Na tej točki lahko uporabite pomoč za kratek opis katere koli funkcije v seznamu. Tu je primer pomoči za funkcija upper:

```
>>> help(popcorn.upper)
Help on built-in function upper:
upper(...)
S.upper() -> str
Return a copy of S converted to uppercase.
```

Vrnjene informacije vas lahko malo zmedejo, zato si jih pobliže oglejmo. Oklepaji (...) pomenijo, da je upper vgrajena funkcija razreda string in v tem primeru ne sprejema nobenih parametrov. Puščica (->) v naslednji vrstici pomeni, da ta funkcija vrne niz (str). Zadnja vrstica ponuja kratek opis, kaj funkcija naredi.

Funkcija eval

Funkcija eval (kratko za evaluate) sprejme kot parameter niz in ga požene, kot da je Pythonov ukaz. Na primer, eval('print("wow")') bo dejansko zagnal ukaz print("wow").

Funkcija eval deluje le s preprostimi izrazi, na primer naslednji:

```
>>> eval('10 * 5 ')
50
```

Izrazi, ki so razdeljeni na več kot eno vrstico (npr. pogojni stavki) na splošno ne bodo ovrednoteni, kot v tem primeru:

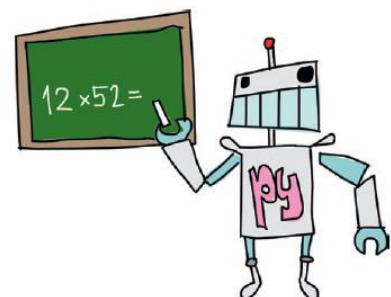
```
>>> eval('''if True:
print("to sploh ne bo delovalo") ''')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<string>", line 1
if True: print("to sploh ne bo delovalo")
^
SyntaxError: invalid syntax
```

Funkcija eval se pogosto uporablja za prenos uporabnikovega vnosa v Python izraze. Napišete lahko preprost kalkulator, ki prebere vnesene enačbe in jih Python nato izračuna (ovrednoti).

Ker se uporabniški vnos prebere kot niz, mora Python najprej izvesti pretvorbe števil in operatorjev. Funkcija eval omogoča enostavno pretvorbo:

```
>>> vas_izracun = input('Vnesite izračun:')
Vnesite izračun: 12 * 52
>>> eval(vas_izracun)
624
```

V tem primeru uporabimo input za branje tega, kar uporabnik vnese v spremenljivko vas_izracun. V naslednji vrstici vnesemo izraz 12 * 52 (morda vaša starost, pomnožena s številom tednov v enem letu). Za izračun smo uporabili eval, rezultat pa je izpisan v zadnji vrstici.



Funkcija exec

Funkcija `exec` je podobna kot `eval`, le da jo lahko uporabite za zagon bolj zapletenih programov. Razlika med obema je ta, da `eval` vrne vrednost (nekaj, kar lahko shranite v spremenljivki), `exec` pa ne. Tukaj je primer:

```
>>> moj_programcek = '''print(' šunka ')
print("sendvič") '''
>>> exec(moj_programcek)
šunka
sendvič
```

V prvih dveh vrsticah ustvarjamo spremenljivko z večvrstičnim nizom, ki vsebuje dva print stavka, nato pa za izvedbo niza uporabite ukaz `exec`.

Funkcijo `exec` lahko uporabite za zagon programčkov, ki jih Python prebere iz datotek - res, programi znotraj programov! To je lahko zelo uporabno pri pisanju dolgih, zapletenih aplikacij. Lahko bi ustvarili na primer igro Dvoboj robotov, kjer se dva robota premikata po zaslonu in poskušata napadati drug drugega. Igralci bi lahko napisali navodila za svoje robote kot mini Python programe. Igra Dvoboj robotov bi v tem primeru prebrala skripte in uporabila `exec` za zagon.

Funkcija float

Funkcija `float` pretvori niz ali število v število s plavajočo vejico (floating point), ki je število z decimalnim mestom (imenovano tudi realno število). Na primer, število 10 je celo število (integer), števila: 10.0, 10.1 in 10.253 pa so decimalna števila (imenovana tudi floats – floating point numbers).

Decimalna števila (namesto celih števil) bi na primer lahko uporabljali v programu, kjer bi imeli opravka z denarjem. Uporabljajo se tudi v grafičnih programih (npr. 3D igre) za preračunavanje kako in kje risati stvari na zaslonu.

Niz lahko pretvorite v decimalno število preprosto tako, da pokličete `float`:

```
>>> float('12 ')
12.0
```

V nizu lahko uporabite tudi decimalno mesto:

```
>>> float('123.456789')
123.456789
```

Funkcijo `float` lahko uporabite za pretvorbo v ustrezne številke, kar je še posebej koristno, ko je treba primerjati vrednosti, ki jih uporabnik vnese z drugimi vrednostmi. Na primer, če želimo preveriti ali je starost osebe nad določeno vrednostjo, bi lahko naredili to:

```
>>> vnos_starost = input('Vnesite svojo starost:')
Vnesite svojo starost: 20
>>> starost = float(vnos_starost)
>>> if starost > 13:
    print('Prestari ste za %s let.' %(starost - 13))
Prestari ste za 7.0 let.
```

Funkcija int

Funkcija `int` pretvori niz ali število v celo število (ali integer), kar v bistvu pomeni, da zanemari vse, kar je za decimalno piko. Tukaj lahko na primer pretvorite decimalna števila v cela števila:

```
>>> int(123.456)
123
```

Ta primer pretvori niz v celo število:

```
>>> int('123')
123
```

Toda, če poskusite pretvoriti niz, ki vsebuje decimalno število v celo število, dobite sporočilo o napaki. Tukaj skušamo pretvoriti niz, ki vsebuje decimalno piko s funkcijo int:

```
>>> int('123.456')
Traceback (most recent call last):
  File "<pysheell>", line 1, in <module>
    int('123.456')
ValueError: invalid literal for int() with base 10: '123.456'
```

Kot lahko vidite, je rezultat sporočilo ValueError.

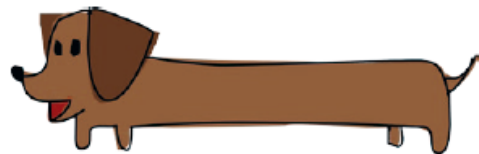
Funkcija len

Funkcija len vrne dolžino objekta ali v primeru niza število znakov v nizu. Da bi dobili dolžina tega niza bi naredili tole:

```
>>> len('to je testni niz')
21
```

Če se uporablja s seznamami ali tuple, len vrne število predmetov v tem seznamu ali tuple:

```
>>> seznam_bitij = ['samorog', 'kiklop',
'vila', 'škrat', 'zmaj', 'trol']
>>> print(len(seznam_bitij))
6
```



Uporabljen s slovarjem (map), len vrne število predmetov v njem:

```
>>> sovrazniki = {'Batman': 'Joker', "Superman": "Lex Luthor", "Spiderman":
"Green Goblin"}
>>> print(len(sovrazniki))
3
```

Funkcija len je še posebej uporabna, ko delate z zankami. Uporabimo ga lahko za prikaz indeksnih pozicij elementov na seznamu, kot je ta:

```
>>> sadje = ['jabolko', 'banana', 'klementina', 'ananas']
(1) >>> dolzina = len(sadje)
(2) >>> for x in range(0, dolzina):
(3)     print("Sadje pri indeksu %s je %s" %(x, sadje [x]))
Sadje pri indeksu 0 je jabolko
Sadje pri indeksu 1 je banana
Sadje pri indeksu 2 je klementina
Sadje pri indeksu 3 je ananas
```

Tukaj shranimo dolžino seznama v spremenljivo dolzina (1), nato jo uporabimo v funkciji range v for zanki (2). Ko se zanka sprehaja po elementih seznama izpišemo sporočilo (3), ki prikazuje pozicijski indeks in vrednost.

Funkciji max in min

Funkcija max vrne največji element v seznamu, tuple ali nizu. Tukaj vidimo, kako ga uporabimo na seznamu števil:

```
>>> stevila = [5, 4, 10, 30, 22]
>>> print(max(stevila))
30
```

Na nizu z znaki, ločenimi z vejicami ali presledki bo tudi delovalo:

```
>>> strings = 's, t, r, i, n, g, S, T, R, I, N, G'
>>> print(max(strings))
t
```

Kot prikazuje ta primer, so črke razvrščene po abecednem redu in male črke so za velikimi, zato je t večji kot T.

Toda ni vam treba uporabljati seznamov, tuple ali nizov. Funkcijo max lahko kličete tudi direktno na zelenih elementih:

```
>>> print(max(10, 300, 450, 50, 90))
450
```

Funkcija min deluje podobno kot max, le da vrne najmanjši element v seznamu, tuple ali nizu. Tukaj je naš seznam števil z uporabo min, namesto max:

```
>>> stevila = [5, 4, 10, 30, 22]
>>> print(min(stevila))
4
```

Recimo, da igrate igro ugibanja s štirimi igralci in vsak mora uganiti številko, ki je manjša od vaše številke. Če katerikoli igralec ugiba nad vašo številko, izgubijo vsi igralci, če pa vsi ugibajo nižje, zmagajo. Lahko bi uporabili max in ugotovili, če so vsi ugibali nižje, takole:

```
>>> ugani_stevilo = 61
>>> igralci_ugibajo = [12, 15, 70, 45]
>>> if max(igralci_ugibajo) > ugani_stevilo:
>>>     print('Bum! Vsi ste izgubili')
else:
>>>     print('Zmagali ste!')
Bum! Vsi ste izgubili
```

V tem primeru shranimo številko za uganit z uporabo spremenljivke ugani_stevilo. Ugibanja igralcev so shranjene v seznamu igralci_ugibajo. If stavek preveri največje ugibano številko v igralci_ugibajo in, če kateri koli igralec ugiba nad številko, natisnemo sporočilo "Bum! Vsi ste izgubili."

Funkcija range

Funkcija range, kot smo že videli, se v glavnem uporablja v for zankah, za ponavljanje odsekov kode določeno število krat. Prva dva parametra, ki se nanašata na območje, se imenujeta start in stop. Take primere smo že videli.



Številke, ki jih generira range, se začnejo s številom kot prvim parametrom in se končajo s številom, ki je za eno manjše kot drugi parameter. V nadaljevanju je prikazano, kaj se zgodi, ko natisnemo številke, ki jih ustvari range med 0 in 5:

```
>>> for x in range(0, 5):
    print(x)
0
1
2
3
4
```

Funkcija range dejansko vrne poseben objekt, imenovan iterator, ki ponavlja akcijo določeno število krat. V tem primeru vrne naslednje največje število vsakič, ko se pokliče.

Iterator lahko pretvorite v seznam (z uporabo funkcije list). Če nato izpišete vrnjeno vrednost, boste videli številke, ki jih vsebuje:

```
>>> print(list(range(0, 5)))
[0, 1, 2, 3, 4]
```

Uporabite lahko še tretji parameter, ki se imenuje korak (step). Če vrednost koraka ni vključena, se uporabi število 1 kot privzeto. Toda kaj se zgodi, ko vnesemo število 2? Tukaj je rezultat:

```
>>> stej_po_dve = list(range(0, 30, 2))
>>> print(stej_po_dve)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

Vsaka številka na seznamu se poveča za dva. Seznam se konča s številko 28, kar je 2 manj od 30. Lahko uporabite tudi negativne korake:

```
>>> stej_nazaj_po_dve = list(range(40, 10, -2))
>>> print(stej_nazaj_po_dve)
[40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12]
```

Funkcija sum

Funkcija sum sešteje elemente v seznamu in vrne skupno vrednost. Tukaj je primer:

```
>>> moj_seznam = list(range(0, 500, 50))
>>> print(moj_seznam)
[0, 50, 100, 150, 200, 250, 300, 350, 400, 450]
>>> print(sum(moj_seznam))
2250
```

V prvi vrstici ustvarimo seznam števil med 0 in 500, pri čemer uporabimo range s korakom 50. Nato izpišemo seznam. Nazadnje izpišemo sum s parametrom moj_seznam (2250).

Delo z datotekami

Datoteke Python so enake kot druge datoteke v računalniku: dokumenti, slike, glasba, igre... res, vse na računalniku je shranjeno v datotekah.

Oglejmo si, kako odpreti in delati z datotekami v Pythonu z vgrajeno funkcijo open. Toda najprej moramo ustvariti novo datoteko za preizkušanje.

Ustvarjanje poskusne datoteke

Igrali se bomo z besedilno datoteko, ki jo bomo imenovali test.txt. Sledite korakom.

Ustvarjanje nove datoteke v operacijskem sistemu Windows

1. Izberite Start in v iskalnik napišite Notepad.
2. V prazno datoteko vnesite nekaj vrstic.
3. Izberite File, Save.
4. Ko se prikaže pogovorno okno, izberite svojo delovno mapo
5. V polje Ime datoteke na dnu pogovornega okna vnesite test.txt.
6. Nazadnje kliknite gumb Shrani.

Odpiranje datoteke v Pythonu

Vgrajena funkcija open odpre datoteko v lupini Python in prikaže njeno vsebino. Kako poveste funkciji, katero datoteko želite odpreti. Poglejmo primer za Windows datoteko in jo nato preberimo.

Odpiranje datoteke v Windows

```
>>> test_file = open('c:\\pot do vaše mape\\test.txt')
>>> text = test_file.read()
>>> print(text)
```

```
Nekoč je živel deček z imenom Marcelo,
ki je sanjal, da je pojedel krof.
Ko se je zbudil, se je začudil,
da je njegova postelja razpadla
in ugotovil je, da je bil precej bolj okrogel
```

V prvi vrstici uporabljamo open, ki vrne objekt datoteke s funkcijami za delo z datotekami. Parameter, ki ga uporablja funkcija open je niz, ki Pythonu pove, kje najti datoteko.

```
c:\\<pot do vaše datoteke>\\test.txt.
```

Dve obrnjeni poševnica v imenu datoteke povesta Pythonu, da je poševnica zgolj to, in ne neke vrste ukaza. (kot ste se naučili v poglavju 3, imajo samo obrnjene poševnice poseben pomen v Pythonu, zlasti v nizih.) Datotečni objekt shranimo v spremenljivko test_file.

V drugi vrstici uporabljamo funkcijo read, s katero preberemo vsebino datoteke in jo shranimo v spremenljivko text. V zadnji vrstici izpišemo vsebino datoteke.

Pisanje v datoteke

Datotečni objekt, ki ga vrne open, ima poleg read še druge funkcije. Novo, prazno datoteko lahko ustvarimo z uporabo drugega parametra, niza 'w', ko pokličemo funkcijo:

```
>>> test_file = open('c:\\myfile.txt', 'w')
```

Parameter 'w' pove Pythonu, da želimo pisati v datoteko in ne brati iz nje.

Sedaj lahko s pomočjo funkcije write dodamo vsebino v datoteko:

```
>>> test_file = open('c:\\myfile.txt', 'w')
```

```
>>> test_file.write('To je moja testna datoteka')
20
```

Na koncu moramo povedati Pythonu, da smo končali s pisanjem v datoteko z uporabo funkcije close:

```
>>> test_file = open('c:\\myfile.txt', 'w')
>>> test_file.write('Kaj je zeleno in glasno? Žaba!')
>>> test_file.close()
```

Če sedaj odprete datoteko s svojim urejevalnikom besedil, bi morali videti, da vsebuje besedilo "Kaj je zeleno in glasno? Žaba! Ali pa uporabite Python za ponovno branje:

```
>>> test_file = open("myfile.txt")
>>> print(test_file.read())
Kaj je zeleno in glasno? Žaba!
```



Kaj ste se naučili

V tem poglavju ste se naučili nekaj o Pythonovih vgrajenih funkcijah: float in int, ki lahko spremenita decimalna števila v cela in obratno. Videli ste tudi, kako funkcija len poenostavi zanko in kako se Python lahko uporablja za odpiranje datotek, da iz njih lahko prebere vsebino ali vanje piše.

Vaje

Preizkusite naslednje primere, da povadite z nekaterimi Pythonovimi vgrajenimi funkcijami. Poiščite odgovore na <https://nostarch.com/pythonforkids>.

1: Skrivnostna koda

Kakšen je rezultat naslednje kode? Najprej poskusi uganiti, nato pa preveri v lupini.

```
>>> a = abs(10) + abs(-10)
>>> natisni(a)
>>> b = abs(-10) + -10
>>> natisni(b)
```

2: Skrito sporočilo

Poskusite uporabiti dir in help, da bi stavek v nizu spremenili v posamezne besede in nato ustvarite programček za izpis vseh besed v naslednjem nizu, začenši s prvo besedo (this):

```
"this if is you not are a reading very this good then way you to have hide done a it message wrong"
```

3: Kopiranje datoteke

Ustvarite program za kopiranje datoteke. (Namig: Odprite datoteko, ki jo želite kopirati, jo preberite in ustvarite novo datoteka - kopija.) Preverite, ali vaš program deluje tako, da izpišete vsebino obeh datotek na zaslonu.

10. poglavje: Uporabni moduli

Kot ste spoznali v 7. poglavju, je modul v Pythonu zbirka funkcij, razredov in spremenljivk. Python uporablja module za združevanje funkcij in razredov, da bi jih lažje uporabljali. Na primer, modul `turtle`, ki smo ga uporabljali v prejšnjih poglavjih, združuje funkcije in razrede, ki se uporabljajo za ustvarjanje platna za želvo, ki se izrisuje na zaslonu.

Ko uvozite modul v program, lahko uporabite vso njegovo vsebino. Ko smo uvozili `turtle` modul v 4. poglavju smo imeli dostop do razreda `Pen`, ki smo ga uporabili za ustvarjenje objekta platno za želvo:

```
>>> import turtle
>>> t = turtle.Pen()
```

Python ima veliko modulov za raznovrstna opravila. V tem poglavju si bomo ogledali nekaj najbolj uporabnih in poskusili nekatere njihove funkcije.

Izdelava kopij z modulom `copy`

Modul `copy` vsebuje funkcije za ustvarjanje kopij objektov. Ko pišete program, običajno ustvariti nove objekte, včasih pa je koristno ustvariti kopijo objekta in jo nato uporabiti za oblikovanje novega objekta, še posebej, ko je izdelava objekt bolj zapletena. Recimo, da imamo razred `Zival`, z `__init__` funkcijo, ki sprejme parametre `vrsta`, `stevilo_nog` in `barva`.

```
>>> class Zival:
    def __init__(self, vrsta, stevilo_nog, barva):
        self.vrsta = vrsta
        self.stevilo_nog = stevilo_nog
        self.barva = barva
```

Z naslednjo kodo lahko ustvarimo nov objekt razreda `Zival`. Ustvarimo roza hippogriffa s šestimi nogami, imenovanega `harry`.

```
>>> harry = Zival('hippogriff', 6, 'roza')
```

Recimo, da želimo čredo roza hippogriffov s šestimi nogami? Prejšnjo kodo bi lahko ponavljali, lahko pa uporabimo kopiranje, ki ga najdete v modulu `copy`:

```
>>> import copy
>>> harry = Zival("hippogriff", 6, "roza")
>>> harriet = copy.copy(harry)
>>> print(harry.vrsta)
hippogriff
>>> print(harriet.vrsta)
hippogriff
```

V tem primeru izdelamo objekt in ga označimo s spremenljivko `harry`, nato ustvarimo kopijo tega objekta in jo označimo `harriet`. To sta dva popolnoma različna objekta, čeprav imata isto vrsto. Tu smo prihranili le nekaj tipkanja, ko pa so objekti veliko bolj zapleteni, je njihovo kopiranje lahko zelo koristno.

Prav tako lahko ustvarimo in kopiramo seznam živali.

```
>>> harry = Zival("hippogriff", 6, "roza")
```

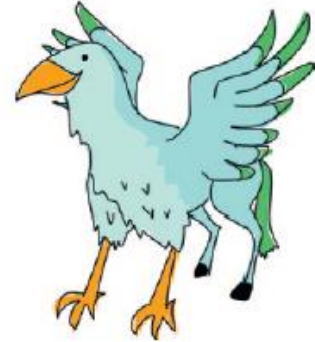


```

>>> carrie = Zival("himera", 4, "zelene pike")
>>> billy = Zival("bogill", 0, "pisan")
>>> moje_zivali = [harry, carrie, billy]
>>> druge_zivali = copy.copy(moje_zivali)
>>> print(druge_zivali [0] .vrsta)
hippogriff
>>> print(druge_zivali [1] .vrsta)
himera

```

V prvih treh vrsticah ustvarimo tri živali in jih shranimo kot harry, carrie in billy. V četrti vrstici, te predmete shranimo v seznam moje_zivali. Nato uporabimo copy za ustvarjanje novega seznama druge_zivali. Na koncu natisnemo vrsto prvih dveh objektov ([0] in [1]) v seznamu druge_zivali in vidimo, da so enake kot v prvotnem seznamu: hippogriff in himera. Naredili smo kopijo seznama in ni nam bilo treba ustvarjati novih objektov.



Toda poglejte, kaj se zgodi, če spremenimo vrsto enega od objektov Zival v izvornem seznamu moje_zivali (hippogriff to ghou). Python spremeni vrsto tudi v druge_zivali.

```

>>> moje_zivali [0] .vrsta = 'ghoul'
>>> print(moje_zivali [0] .vrsta)
ghoul
>>> print(druge_zivali [0] .vrsta)
ghoul

```

To je čudno. Ali nismo spremenili vrste samo v moje_zivali? Zakaj se je vrsta spremenila v obeh seznamih?

Vrste so se spremenile, ker je copy dejansko shallow copy, kar pomeni, da ne kopira objektov znotraj objektov, ki smo jih kopirali. Tukaj je kopiralo glavni seznam objektov, ne pa posameznih objektov znotraj seznama. Torej ostanemo z novim seznamom, ki pa nima novih objektov - seznam druge_zivali ima enake tri objekte.

Če dodamo novo žival v prvi seznam (moje_zivali), se v kopiji ne pojavi (druge_zivali). Kot dokaz, natisnite dolžino vsakega seznama po dodajanju druge živali, na primer:

```

>>> sally = Zival("sfinga", 4, "peščena")
>>> moje_zivali.append(sally)
>>> print(len(moje_zivali))
4
>>> print(len(druge_zivali))
3

```

Kot lahko vidite, se dodana žival prvemu seznamu ni pojavila v drugem. Ko uporabimo len in izpišemo rezultate, ima prvi seznam štiri elemente drugi pa tri.

Druga funkcija v modulu copy, deepcopy, dejansko ustvarja kopije vseh objektov znotraj objektov, ki se kopirajo. Ko uporabljamo deepcopy za kopiranje moje_zivali, dobimo nov seznam skupaj s kopijami vseh objektov. Posledično sprememba ene od naših prvotnih živali ne bodo vplivali na objekte v novem seznamu. Tukaj je primer:

```

>>> druge_zivali = copy.deepcopy(moje_zivali)
>>> moje_zivali [0] .vrsta = 'wyrm'
>>> print(moje_zivali [0] .vrsta)
wyrm

```

```
>>> print(druge_zivali [0] .vrsta)
ghoul
```

Ko spremenimo vrsto prvega objekta v izvirnem seznamu od ghoul na wyrm, se kopirani seznam ne spremeni, kot lahko vidimo v izpisu.

Na sledi ključnim besedam z modulom keywords

Ključne besede v Pythonu so katerekoli besede, ki so del jezika samega, kot if, else in for. Modul keywords vsebuje funkcijo imenovano iskeyword in spremenljivka imenovano kwlist. Funkcija iskeyword vrne True, če je kateri koli niz ključna beseda v Pythonu. Spremenljivka kwlist vrne seznam vseh ključnih besed v Pythonu.

V naslednji kodi bodite pozorni, da funkcija iskeyword vrne True za niz 'if' in False za niz 'ozwald'. Če izpišemo kwlist se izpišejo vse ključne besede, kar je koristno, ker ključne besede niso vedno enake. Nove različice (ali starejše različice) Pythona imajo lahko različne ključne besede.

```
>>> import keywords
>>> print(keyword.iskeyword('if'))
True
>>> print(keyword.iskeyword('ozwald'))
False
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',
'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

V prilogi najdete opis vseh ključnih besed.

Z modulom random do naključnih števil

Modul random vsebuje številne uporabne funkcije za generiranje naključnih števil, kot bi na primer prosil računalnik "izberi številko." Najbolj uporabne funkcije v naključnem modulu so randint, choice in shuffle.

Uporaba randint vrne naključno celo število

Funkcija randint izbere naključno številko v razponu števil, na primer med 1 in 100, med 100 in 1000, ali med 1000 in 5000. Tukaj je primer:

```
>>> import random
>>> print(random.randint(1, 100))
58
>>> print(random.randint(100, 1000))
861
>>> print(random.randint(1000, 5000))
3795
```

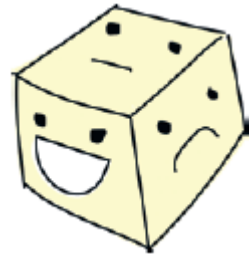
Funkcijo randint lahko uporabite za enostavno (in nadležno) igrico ugibanja števil z uporabo while zanke, kot je ta:

```
>>> import random
>>> num = random.randint(1, 100)
```

```

(1) >>> while True:
(2)     print('Ugibajte število med 1 in 100')
(3)     guess = input()
(4)     i = int(guess)
(5)     if i == num:
(6)         print('Uganil si')
(7)         break
(8)     elif i < num:
(9)         print('Poskusi večje število')
(10)    elif i > num:
(11)        print('Poskusi manjše število')

```



Najprej uvozimo modul `random` in potem spremenljivko `num` nastavimo na naključno vrednost z uporabo `randint` v razponu od 1 do 100. Nato ustvarimo neskončno `while` zanko (1) (ali vsaj dokler igralec ne ugame številke). Nato izpišemo sporočilo (2) in uporabimo `input` za vnos uporabnikovega števila, ki ga shranimo v spremenljivki `guess` (3). Pretvorimo vnos v celo število z uporabo `int` in jo shranimo v spremenljivko `i` (4). Nato primerjamo z naključno izbranim številom (5). Če sta vnos in naključno generirana številka enaka, izpišemo "Uganil si" in nato zapustimo zanko (6). Če številki nista enaki, preverimo, če je število, ki ga je igralec vnesel višje od naključnega števila (7), ali nižje (8), in izpišemo ustrezno sporočilo.

Ta koda je dolga, zato jo boste morda želeli vnesti v novo okno ukazne lupine ali ustvariti besedilni dokument, ga shraniti in nato pognati v IDLE. Tule je opomnik, kako odpreti in zagnati shranjeni program:

1. Zaženite IDLE in izberite File, Open.
2. Prebrskajte do svoje mape, kjer ste datoteko shranili in kliknite na ime datoteke, da jo izberete.
3. Kliknite Odpri.
4. Ko se odpre novo okno, izberite Run, Run Module.

Evo, kaj se zgodi, ko zaženemo program:

```

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
----- RESTART: D:/python_vaje/ugibanje_stevila.py -----
Ugibajte število med 1 in 100
50
Poskusi manjšo številko
Ugibajte število med 1 in 100
25
Poskusi manjšo številko
Ugibajte število med 1 in 100
12
Poskusi večjo številko
Ugibajte število med 1 in 100
18
Poskusi manjšo številko
Ugibajte število med 1 in 100
14
Poskusi večjo številko
Ugibajte število med 1 in 100
16
Uganil si
>>> |
Ln: 23 Col: 4

```

Uporaba choice za izbiro naključnega predmeta iz seznama

Če želite naključen element iz seznama namesto naključnega števila iz določenega razpona, lahko uporabite choice. Na primer, če bi morda želeli, da Python izbere sladico za vas.

```
>>> import random
>>> sladice = ["sladoled", "palačinke", "napolitanke", "piškotki",
               "lizika"]
>>> print(random.choice(sladice))
piškotki
```

Izgleda, da boste imeli piškotke - sploh ni slaba izbira.

Uporaba shuffle za premešanje seznama

Funkcija shuffle spremeni seznam tako, da premeša elemente. Če vzporedno delate v IDLE in ste vnesli sladice iz prejšnjega primera, lahko preskočite direktno na ukaz random.shuffle v spodnji kodi.

```
>>> import random
>>> sladice = ["sladoled", "palačinke", "napolitanke", "piškotki",
               "lizika"]
>>> random.shuffle(sladice)
>>> print(sladice)
["palačinke", "sladoled", "lizika", "napolitanke", "piškotki"]
```

Rezultate mešanja si lahko ogledate po izpisu seznama, naročilo je povsem drugačno. Če bi napisali igro s kartami, bi za mešanje seznama kart lahko uporabili shuffle.

Nadzor lupine z modulom sys

Modul sys vsebuje sistemske funkcije, ki jih lahko uporabite za krmiljenje Pythonove lupine. Tukaj si bomo pogledali, kako uporabljati objekta stdin in stdout ter spremenljivko version.

Branje z objektom stdin

Objekt stdin (kratko za standard input) v modulu sys zahteva od uporabnik, da vnese podatke, ki jih nato program uporabi. Kot ste videli že v 7. poglavju, ima ta objekt funkcijo readline, ki se uporablja za branje vrstice besedila vnesenega s tipkovnico, dokler se ne pritisne enter. Deluje kot funkcija input, ki smo jo uporabili v igri Ugani naključno število. Vnesite na primer:

```
>>> import sys
>>> v = sys.stdin.readline()
Kdor se smeje zadnji, najpočasneje misli
```

Python bo shranil vrsto. 'Kdor se smeje zadnji, najpočasneje misli' v spremenljivko v. Izpišimo vrednost v:

```
>>> print(v)
Kdor se smeje zadnji, najpočasneje misli
```

Ena od razlik med input in funkcijo readline je, da z readline, lahko omejite število znakov za vnos. Vendar to deluje le v komandni vrstici in ne v IDLE. Na primer:

```
>>> v = sys.stdin.readline(13)
Kdor se smeje zadnji, najpočasneje misli
```

```
>>> print(v)
Kdor se smeje
```

Pisanje z objektom stdout

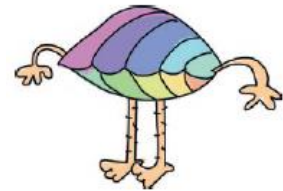
Za razliko od stdin, s stdout objektom (kratko za standard output) izpisujemo sporočila v lupini (ali konzoli). V nekaterih primerih je isto kot print, vendar je stdout datotečni objekt, zato ima enake funkcije, ki smo jih uporabili v 9. poglavju pri write. Tukaj je primer:

```
>>> import sys
>>> sys.stdout.write("What does a fish say when it swims into a wall?
Dam.")
What does a fish say when it swims into a wall? Dam. 52
```

Ste opazili, da se na koncu izpisa izpiše še število znakov. To vrednost bi lahko shranjevali v spremenljivko in na ta način vedeli, koliko znakov smo izpisovali na zaslon.

Katere verzijo Pythona uporabljam?

Spremenljivka version prikaže verzijo Pythona, ki je lahko koristen podatek, če želite vedeti podatke o verziji. Včasih kakšen program ne teče v starejših verzijah in na ta način lahko obvestite uporabnika, naj posodobiti Python. Poglejmo si izpis verzije:



```
>>> import sys
>>> print(sys.version)
3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)]
```

Obdelovanje časa z modulom time

Pythonov modul time vsebuje funkcije za prikaz časa, čeprav morda nekoliko nepričakovano. Poskusite to:

```
>>> import time
>>> print(time.time())
1300139149.34
```

Številka, ki jo je vrnil ukaz time () je dejansko število sekund od 1. januarja 1970, ob 00:00:00 natančno. Samo po sebi se to morda ne zdi ravno koristen podatek, vendar lahko služi namenu. Če želite izvedeti, kako dolgo vaš program izvaja določeno kodo, lahko shranite čas na začetku in koncu ter primerjati vrednosti. Poskusimo ugotoviti, kako dolgo bo trajalo, da natisnete vse številke od 0 do 999. Najprej ustvarite takšno funkcijo:

```
>>> def lot_of_numbers (max):
    for x in range(0, max):
        print(x)
```

Nato pokličite funkcijo z max nastavljeno na 1000:

```
>>> lot_of_numbers(1000)
```

Nato ugotovite, kako dolgo traja funkcija, tako da spremenite naš program s časovnim modulom.

```
>>> def lot_of_numbers (max):
    (1) t1 = time.time()
```



```
(2) for x in range(0, max):
    print(x)
(3) t2 = time.time()
(4) print('Trajalo je %s sekund' %(t2-t1))
```

Če ponovno zaženemo program, dobimo naslednji rezultat (ki je odvisen od hitrosti vašega sistema):

```
>>> lot_of_numbers(1000)
0
1
2
3
...
997
998
999
Trajalo je 2.5159196853637695 sekund
```

To deluje tako: najprej pokličemo funkcijo `time()` in dodelimo vrednost spremenljivki `t1` (1). V for zanki natisnemo vse številke (2). Po zanki ponovno pokličemo `time()` v spremenljivko `t2` (3). Ker je trajalo nekaj sekund za izpis vseh števil, bo vrednost v `t2` višja od `t1`, ker je preteklo več sekund po 1. januarja 1970. Izpišemo razliko med `t2` in `t1` (4) in dobimo število sekund izvajanja zanke.

Pretvarjanje datuma z `asctime`

Funkcija `asctime` vzame datum kot tuple in ga pretvori v nekaj bolj berljivega. (Ne pozabite, da je tuple kot seznam z elementi, ki jih ne morete spremeniti.) Kot ste videli v poglavju 7, klic `asctime` brez parametrov prikaže trenutni datum in čas v berljivi obliki.

```
>>> import time
>>> print(time.asctime())
Pon Mar 11 22:03:41 2013
```

Če želite poklicati `asctime` s parametrom, najprej ustvarimo tuple z vrednosti za datum in čas. Tukaj bomo na primer ustvarili tuple s spremenljivko `t`:

```
>>> t = (2007, 5, 27, 10, 30, 48, 6, 0, 0)
```

Vrednosti v zaporedju so leto, mesec, dan, ure, minute, sekunde, dan v tednu (0 je ponedeljek, 1 je torek in tako naprej), dan v letu (vstavili smo 0) in, če je poletni čas (0 ni, 1 je). Po klicu `asctime` s tuple, dobimo to:

```
>>> import time
>>> t = (2020, 2, 23, 10, 30, 48, 6, 0, 0)
>>> print(time.asctime(t))
Ned Feb 23 10:30:48 2020
```

Pridobivanje datuma in časa z lokalnim časom

Za razliko od `asctime`, funkcija `localtime` vrne trenutni datum in čas kot objekt, z vrednostmi v približno enakem vrstnem redu kot `asctime` vnos. Če natisnete objekt, boste videli ime razreda in vsako od vrednosti, označenih kot `tm_year`, `tm_mon` (za mesec), `tm_mday` (za dan v mesecu), `tm_hour` in tako naprej.

```
>>> import time
>>> print(time.localtime())
```

```
time.struct_time(tm_year = 2020, tm_mon = 2, tm_mday = 23, tm_hour = 22,
tm_min = 18, tm_sec = 39, tm_wday = 0, tm_yday = 73, tm_isdst = 0)
```

Če želite natisniti tekoče leto in mesec, lahko uporabite njun indeks (kot pri tuple, ki smo jih uporabljali z `asctime`). Na podlagi našega primera vidimo, da je leto na prvem mestu (položaj 0) in mesec na drugem (1). Zato uporabljamo `leto = t [0]` in `mesec = t [1]`, takole:

```
>>> t = time.localtime()
>>> leto = t [0]
>>> mesec = t [1]
>>> print(leto)
2020
>>> print(mesec)
2
```

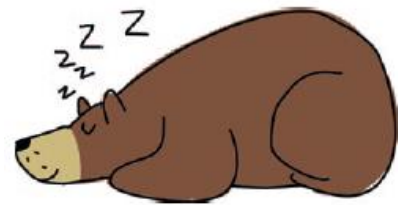
Vidimo, da smo v drugem mesecu leta 2020.

Pavza s funkcijo `sleep`

Funkcija `sleep` je zelo uporabna, ko želite kakšno zakasnitev ali upočasnitev programa. Če vsako sekundo želimo natisniti eno število od 1 do 61, lahko uporabimo naslednjo zanko:

```
>>> for x in range(1, 61):
    print(x)
    time.sleep(1)
```

To doda zakasnitev pri izpisu vsake številke. V 12. poglavju bomo uporabili funkcijo `sleep`, da bomo animacijo naredili malo bolj realistično.



Uporaba modula `pickle` za shranjevanje informacije

Modul `pickle` (stisnjeno kot vložene kumarice) se uporablja za pretvorbo objektov za zapis v datoteko in da potem podatke iz take datoteke lahko tudi preberemo. Morda bo modul `pickle` za vas uporaben, če boste napisali igrico in boste želeli shraniti informacije o napredku igralca. Na primer: tukaj bomo shranili stanje v igri:

```
>>> game_data = {"položaj igralca": "N23 E45", "žepi": ["ključi", "žepni
nož", "polirani kamen"], "nahrbtnik": ["vrv", "kladivo", "jabolko"],
"denar": 158.50 }
```

Tu ustvarimo Python slovar, ki vsebuje igralčev trenutni položaj v naši navidezni igri, seznam predmetov v igralčevem žepu in nahrbtniku ter količino denarja. Ta slovar lahko shranimo v datoteko tako, da datoteko odpremo za pisanje in nato pokličemo `dump` funkcijo modula `pickle`, kot sledi:

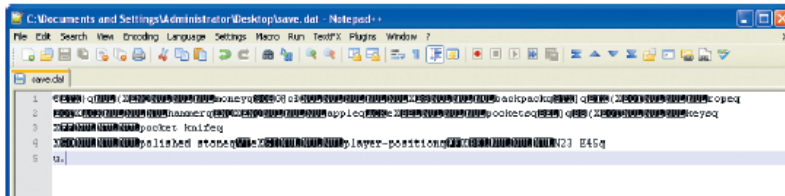


```
(1) >>> import pickle
(2) >>> game_data = { "položaj igralca": "N23 E45", "žepi": ["ključi",
"žepni nož", "polirani kamen"], "nahrbtnik": ["vrv", "kladivo", "jabolko"],
"denar": 158,50 }
(3) >>> save_file = open('save.dat', 'wb')
(4) >>> pickle.dump(game_data, save_file)
(5) >>> save_file.close()
```

Najprej uvozimo modul `pickle` (1) in ustvarimo slovar naših podatkov o igri (2). (3) Odpremo datoteko `save.dat` s parametrom `wb`, ki Pythonu pove, da naj piše v binarno datoteko. Pri (4), smo uporabili

funkcijo dump za prenos slovarja v datoteko. Prvi parameter je spremenljivka podatka, drugi pa spremenljivka datoteke. Nazadnje (5) zapremo datoteko.

Opomba: Besedilne datoteke vsebujejo samo znake, ki jih ljudje lahko preberejo. Slike, glasbene datoteke, filmi in objekti pickle pa imajo informacije, ki niso vedno čitljive za ljudi. To so binarne datoteke. Če bi odprli datoteko save.dat, bi videli, da ne izgleda kot besedilna datoteka; izgleda, da je mešanica normalnega besedila in posebnih znakov.



Objekte lahko tudi naložimo iz tako shranjene datoteke:

```
>>> load_file = open('save.dat', 'rb')
>>> loaded_game_data = pickle.load(load_file)
>>> load_file.close()
```

Najprej odpremo datoteko s parametrom rb, kar pomeni beri binarno. Datoteko nato naložimo s funkcijo load v spremenljivko loaded_game_data. Nazadnje zapremo datoteko. Če želite dokazati, da so bili shranjeni podatki pravilno naloženi, natisnite spremenljivko:

```
>>> print(loaded_game_data)
{"denar": 158.5, "nahrbtnik": ["vrv", "kladivo", "jabolko"], "položaj igralca": "N23 E45", "žepi": ["ključi", "žepni nož", "polirani kamen"]}
```

Kaj ste se naučili

V tem poglavju ste se naučili, kako Python moduli združujejo funkcije, razrede in spremenljivke in kako jih uporabiti z ukazom **import**. Videli ste, kako kopiramo objekte (**copy**), ustvarimo naključna števila (**randint**) in naključno zmešamo sezname (**shuffle**), pa tudi kako delati s časom v Pythonu (**time**). Na koncu ste se naučili, kako shraniti in naložiti podatke iz datotek z uporabo modula **pickle**.

Vaje

Poskusite naslednje vaje z uporabo Python modulov. Preveri svoje odgovore na <https://nostarch.com/pythonforkids>.

1: Kopirani avtomobili

Kaj bo naslednja koda natisnila?

```
>>> import copy
>>> class Avto:
>>>     pass
>>> avto1 = Avto()
>>> avto1.kolesa = 4
>>> avto2 = avto1
```

```
>>> avto2.kolesa = 3
>>> print(avto1.kolesa)
(kaj bo izpisano v tej vrstici)
>>> avto3 = copy.copy(avto1)
>>> avto3.kolesa = 6
>>> print(avto1.kolesa)
(kaj bo izpisano v tej vrstici)
```

2: Shranjene priljubljene vrednosti

Ustvarite seznam svojih najljubših stvari, nato pa uporabite pickle za shranjevanje v datoteko z imenom favourites.dat. Zaprite Python lupino in nato znova odprite in izpišite seznam priljubljenih tako, da naložite datoteko.

11. poglavje: Več želvjne grafike

Poglejmo si še enkrat modul turtle, ki smo ga uporabili v 4. poglavju. Kot boste videli v tem poglavju, z želvami lahko naredite veliko več, kot le rišete črne črte. Uporabite jih lahko za risanje bolj naprednih geometrijskih oblik, uporabljate barve in narisane oblike pobarvate.

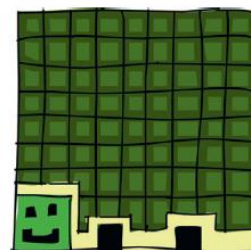
Začnimo z osnovnim kvadratom

Naučili smo se že, kako želvo uporabiti za preprosto risanje. Pred uporabo želve moramo uvoziti turtle modul in ustvarite objekt Pen:

```
>>> import turtle
>>> t = turtle.Pen()
```

Tukaj je koda, ki smo jo uporabili za risanje kvadrata v 4. poglavju:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
```



V 6. poglavju ste spoznali zanke. Z našim znanjem sedaj znamo to narediti s for zanko:

```
>>> t.reset()
>>> for x in range(1, 5):
    t.forward(50)
    t.left(90)
```

V prvi vrstici povemo naj se objekt Pen ponastavi. Nato začnemo for zanko, ki se bo štela od 1 do 4 z range (1, 5). Potem se z v zanki premaknemo za 50 in zavijemo levo za 90 stopinj. Z uporabo for zanke smo privarčevali kar nekaj kode.

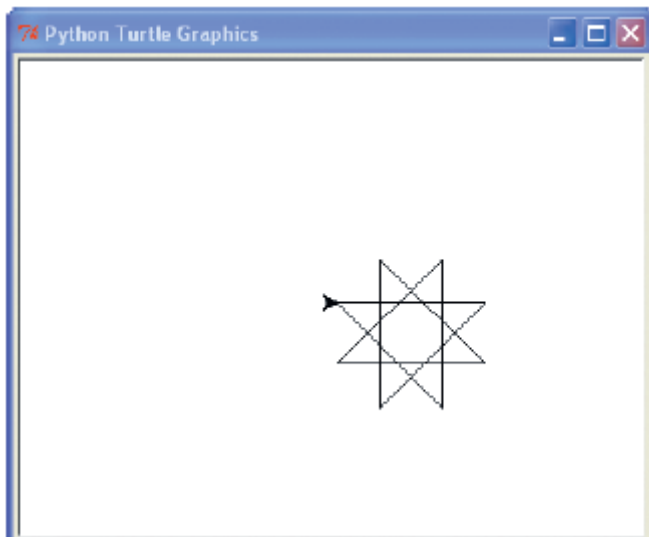
Risanje zvezd

Zdaj, z nekaj preprostimi spremembami v naši zanki, lahko ustvarimo nekaj še bolj zanimivega.

Vnesite naslednje:

```
>>> t.reset()
>>> for x in range(1, 9):
    t.forward(100)
    t.left(225)
```

Ta koda proizvaja osemkrako zvezdo:



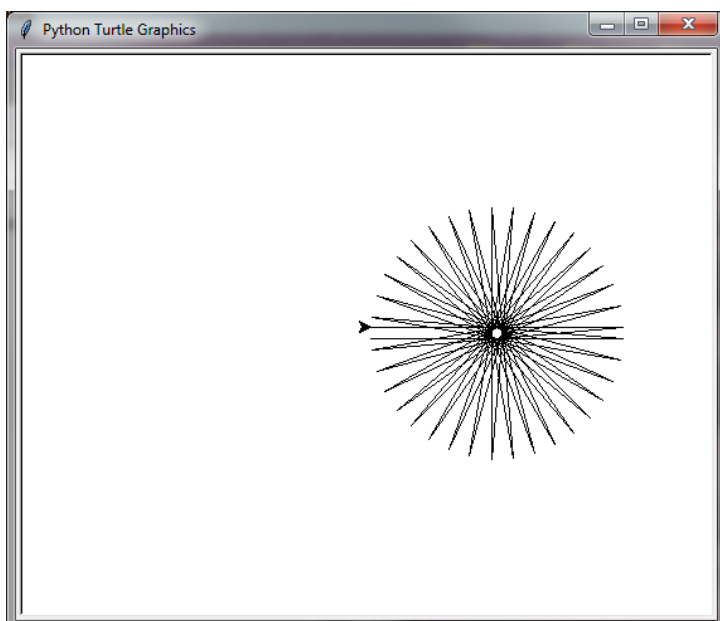
Sama koda je zelo podobna kodi, ki smo jo uporabili za pripravo kvadrata, z nekaj izjemami:

- Namesto zanke s štirimi ponavljanji (1, 5), smo tokrat ponovili osem krat (1, 9).
- Namesto 50 korakov naprej, smo se premikali po 100 pik.
- Namesto obračanja 90 stopinj smo se obrnili za 225 stopinj levo.

Zdaj pa zvezdo še malo izboljšajmo. Z uporabo 175 stopinj in 37 ponavljanji, lahko naredimo zvezdo s še več kraki, z naslednjo kodo:

```
>>> t.reset()
>>> for x in range(1, 38):
    t.forward(100)
    t.left(175)
```

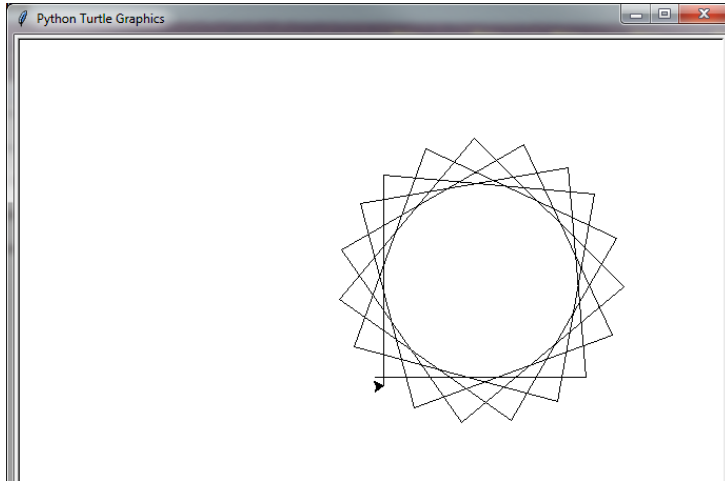
Tu je rezultat te kode:



Medtem, ko se igramo z zvezdami, tule je koda za izdelavo spiralne zvezda:

```
>>> t.reset()
>>> for x in range(1, 20):
    t.forward(100)
    t.left(95)
```

S spreminjanjem zavijanja in zmanjšanjem ponovitev, želva konča s precej drugačnim tipom zvezde.



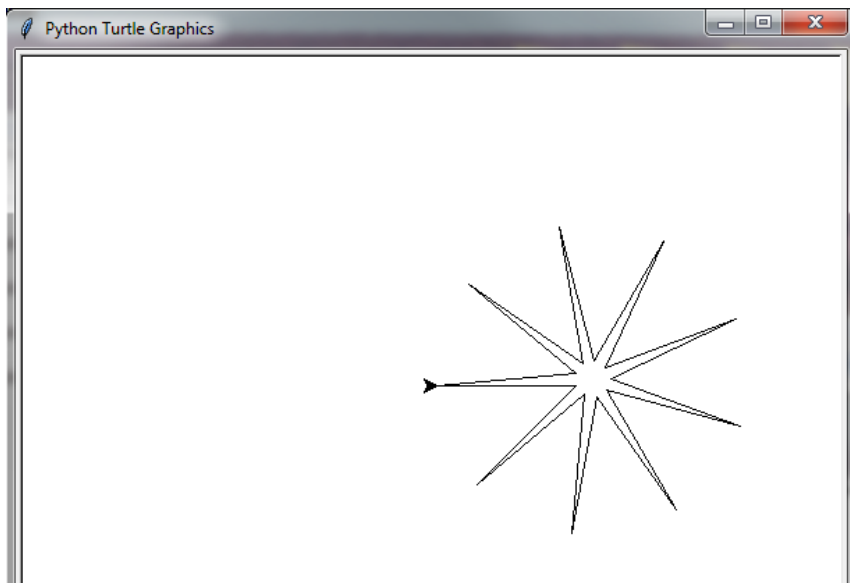
Z uporabo podobne kode lahko ustvarite različne oblike, od osnovnega kvadrata do spiralne zvezde. Kot vidite, smo z uporabo for zank, precej poenostavili naše risanje. Brez for zanke bi se kar natipkali.

Sedaj pa uporabimo še if stavek za nadzor obračanja in ustvarimo nove oblike zvezd. V tem primeru želimo, da se želva obrne enkrat za en kot in drugič za drugega.

```
>>> t.reset()
>>> for x in range(1, 19):
    t.forward(100)
    if x % 2 == 0:
        t.left(175)
    else:
        t.left(225)
```

Tu ustvarimo zanko, ki se bo izvajala 18-krat (`range(1, 19)`) in pove želvi, da se premakne naprej 100 pik (`t.forward(100)`). Nov tukaj je if stavek (`if x % 2 == 0 :`). Ta izjava preveri, če je x parno število s pomočjo operatorja modulo (ostanek pri deljenju s celim številom), $\%$ v izrazu `x % 2 == 0`, pomeni kot bi rekel, "ostanek pri deljenju x z 2" je enak 0. Če na primer 5 delimo z 2, dobimo 2 celi in ostanek 1. Če 1 primerjamo z nič, dobimo False. Drži, da 1 ni parno število.

V peti vrstici naše kode povemo želvi naj zavijemo levo 175 stopinj (`t.left(175)`), če je število x parno (če je `x % 2 == 0 :`); sicer (`else`), v zadnji vrstici rečemo, da je treba zaviti za 225 stopinj (`t.left(225)`). Tu je rezultat zagona te kode:



Risanje avtomobila

Z želvo lahko narišemo tudi bolj zapletene oblike. Za naš naslednji primer bomo naredili precej primitiven avto. Najprej narišimo telo avtomobila. V IDLE izberite File, New Window in v okno vnesite naslednjo kodo:

```
import turtle
t = turtle.Pen()
t.reset()
t.color(1,0,0)
t.begin_fill()
t.forward(100)
t.left(90)
t.forward(20)
t.left(90)
t.forward(20)
t.right(90)
t.forward(20)
t.left(90)
t.forward(60)
t.left(90)
t.forward(20)
t.right(90)
t.forward(20)
t.left(90)
t.forward(20)
t.end_fill()
```

Nato pripravimo prvo kolo.

```
t.color(0,0,0)
t.up()
t.forward(10)
t.down()
t.begin_fill()
t.circle(10)
t.end_fill()
```

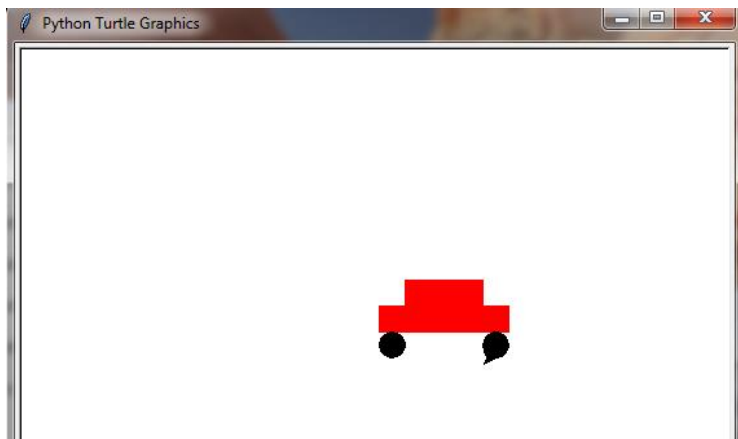
Na koncu narišemo še drugo kolo.

```

t.setheading(0)
t.up()
t.forward(90)
t.right(90)
t.forward(10)
t.setheading(0)
t.begin_fill()
t.down()
t.circle(10)
t.end_fill()

```

Izberite File, Save As. Datoteki dajte ime, na primer avto.py. Izberite Run, Run Modul, da preizkusite kodo. In tukaj je naš avto:



Morda ste opazili, da smo uporabili nekaj novih ukazov:

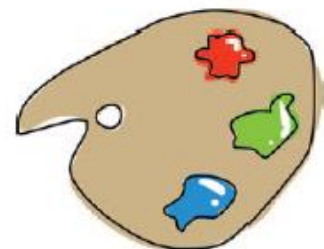
- color se uporablja za spreminjanje barve peresa.
- begin_fill in end_fill se uporabljata za zapolnitev območja platna z barvo.
- circle nariše krog določene velikosti.
- setheading zavije v želeno smer.

Oglejmo si, kako lahko te funkcije uporabimo za dodajanje barve našim risbam.

Kako pobarvamo stvari

Funkcija color sprejme tri parametre. Prvi določa količino rdeče, drugi količino zelene in tretji količino modre barve. Na primer, za rdeč avto smo uporabili color (1,0,0), ki pravi, da naj uporablja 100 odstotkov rdeče pero.

Ta rdeče zeleno moder recept se imenuje RGB (Red, Green, Blue). Tako so barve predstavljene v vašem računalniku. Z mešanica teh primarnih barv dobimo vse druge barve, tako kot, ko si mešate modro in rdečo barvo, da dobite vijolično ali rumeno in rdečo, da dobite oranžno. Barve rdeča, zelena in modra se imenujejo primarne barve.



Podobno kot pri slikanju slikar meša barve z različnimi količinami barve, delamo to tukaj. Recimo, da imamo 3 lončke barv. Vsak lonček je poln, to je vrednost 1 (ali 100 odstotkov). Nato zmešamo vse rdeče barve in vse zelene barve v kadi, da dobimo rumeno (to je 1 in 1 vsake, ali 100 odstotkov vsake barve). Vrnimo se h kodiranju. Za

rumen krog narisano z želvo, bi uporabili 100 odstotkov rdeče in zelene barve, vendar brez modre barve:

```
>>> t.color(1,1,0)
>>> t.begin_fill()
>>> t.circle(50)
>>> t.end_fill()
```

1,1,0 v prvi vrstici predstavlja 100 odstotkov rdeče, 100 odstotkov zelene in 0 odstotkov modre barve. V naslednji vrstici povemo naj želva zapolni obliko s to barvo RGB (t.begin_fill) in nato narišemo krog (t.circle). V zadnji vrstici end_fill pove želvi naj zaključi barvanje.

Funkcija za risanje pobarvanega kroga

Da bi si olajšali preizkus z različnimi barvami, ustvarimo funkcijo, ki smo jo uporabili za pripravo zapolnjenega kroga.

```
>>> def mycircle(rdeca, zelena, modra):
    t.color(rdeca, zelena, modra)
    t.begin_fill()
    t.circle(50)
    t.end_fill()
```

Lahko narišemo zelen krog z uporabo samo zelene barve:

```
>>> mycircle(0, 1, 0)
```

Ali lahko narišemo temnejši zelen krog z uporabo le polovice zelena barva (0.5):

```
>>> mycircle(0, 0.5, 0)
```

Če se želite igrati z barvami RGB na svojem zaslonu, poskusite z risanjem kroga najprej s polno rdečo, nato polovico (1 in 0.5), nato pa s polno modro in pol modre, takole:

```
>>> mycircle(1, 0, 0)
>>> mircircle(0.5, 0, 0)
>>> mycircle(0, 0, 1)
>>> mycircle(0, 0, 0.5)
```

Opomba: Če je vaše platno prepolno, uporabite t.reset (), da izbrišete stare risbe. Prav tako ne pozabite, da lahko premikate želvo brez risanja črt s pomočjo t.up (), da dvignete pero (uporabite t.down (), da ponovno začnete risati).

Različne kombinacije rdeče, zelene in modre bodo ustvarile veliko različnih barv, kot je na primer zlata:

```
>>> mycircle(0.9, 0.75, 0)
```

Tukaj je svetlo rožnata:

```
>>> mycircle(1, 0.7, 0.75)
```

In tukaj sta dve različici za različne oranžne odtenke:

```
>>> mycircle(1, 0.5, 0)
>>> mycircle(0.9, 0.5, 0.15)
```


Poskusite nekaj barv zmešati sami!

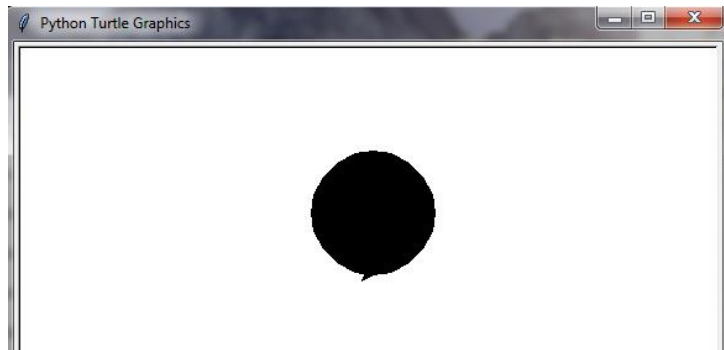
Ustvarjanje čiste črne in bele

Kaj se zgodi, ko ponoči izklopite vse luči? Vse je črno. Enako se zgodi z barvami na računalniku. Nobena svetloba, pomeni nobene barve, torej krog z 0 za vse primarne barve, bo črna:



```
>>> mycircle(0, 0, 0)
```

Tukaj je rezultat:



Drži tudi obratno, če uporabljate 100 odstotkov vseh treh barv. V tem primeru dobite belo. Vnesite to, da izbrišete svoj črni krog:

```
>>> mycircle(1, 1, 1)
```

Funkcija za risanje kvadrata

Videli ste, da oblike napolnimo z barvo tako, da povemo želvi začetek polnjenja z uporabo `begin_fill`, za konec polnjenja uporabljamo funkcijo `end_fill`. Zdaj bomo poskusili še nekaj poskusov z oblikami in polnjenjem. Uporabimo funkcijo risanja kvadrata z začetka poglavja in mu dodajte velikost kot parameter.

```
>>> def mysquare(velikost):  
    for x in range(1, 5):  
        t.forward(velikost)  
        t.left(90)
```

Preizkusite svojo funkcijo tako, da jo pokličete z velikostjo 50, takole:

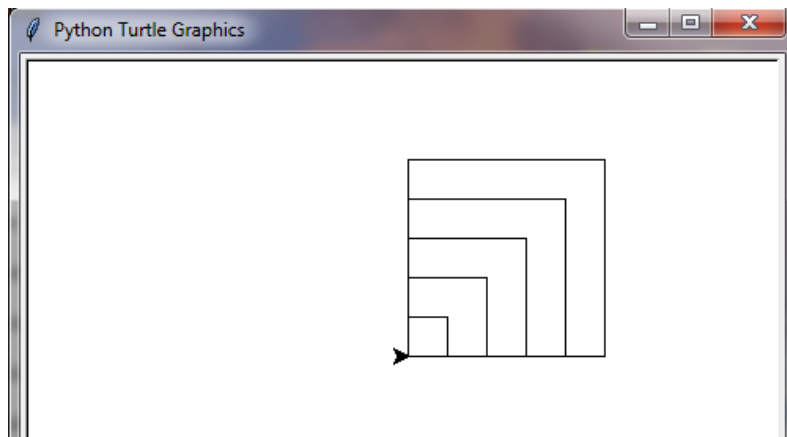
```
>>> mysquare(50)
```

To ustvari majhen kvadrat.

Sedaj pa preizkusimo našo funkcijo z različnimi velikostmi. Naslednji koda ustvarja pet zaporednih kvadratov velikosti 25, 50, 75, 100 in 125.

```
>>> t.reset()  
for x in range(25, 126, 25):  
    mysquare(x)
```

Tako bi morali izgledati ti kvadrati:



Risanje zapolnjenih kvadratov

Za risanje zapolnjenega kvadrata najprej ponastavimo platno in ponovno pokličemo kvadratno funkcijo s to kodo:

```
>>> t.reset()
>>> t.begin_fill()
>>> mysquare(50)
```

Videti bi moral prazen kvadrat, dokler ne končate polnjenja:

```
>>> t.end_fill()
```

Spremenimo našo funkcijo, tako da lahko narišemo bodisi zapolnjen bodisi prazen kvadrat. Za to potrebujemo še en parameter in nekoliko kompleksnejšo kodo.

```
>>> def mysquare(velikost, zapolnjeno):
    if zapolnjeno == True:
        t.begin_fill()
    for x in range(1, 5):
        t.forward(velikost)
        t.left(90)
    if zapolnjeno == True:
        t.end_fill()
```

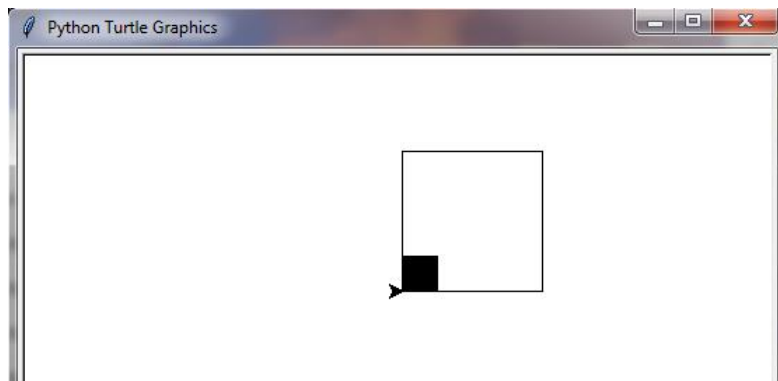
V prvi vrstici spremenimo definicijo naše funkcije z dvema parametroma: velikost in zapolnjeno. Nato preverimo, če je vrednost zapolnjeno nastavljeno na True, if zapolnjeno == True. Če je, pokličimo begin_fill in povemo želvi naj zapolni obliko, ki smo jo narisali. Zanko štirikrat ponovimo (for x in range(0, 4)), da narišemo štiri strani pravokotnika (premik naprej in levo), nato ponovno preverimo, če je zapolnjeno True, if zapolnjeno == True. Če je, barvanje končamo s t.end_fill, in želva zapolni kvadrat z barvo. Sedaj lahko z naslednjo vrstico narišemo zapolnjeni kvadrat:

```
>>> mysquare(50, True)
```

Z naslednjo vrstico ustvarimo prazen kvadrat:

```
>>> mysquare(150, False)
```

Po teh dveh klicih funkcije mysquare dobimo naslednji sliki, ki sta videti kot kvadratno oko.



Narišete lahko še druge oblike in jih pobarvate.

Risanje zapolnjenih zvezd

V našem zadnjem primeru bomo dodali barvo zvezdi, ki smo jo narisali prej. Izvirna koda je bila taka:

```
>>> for x in range(1, 19):
    t.forward(100)
    if x % 2 == 0:
        t.left(175)
    else:
        t.left(225)
```

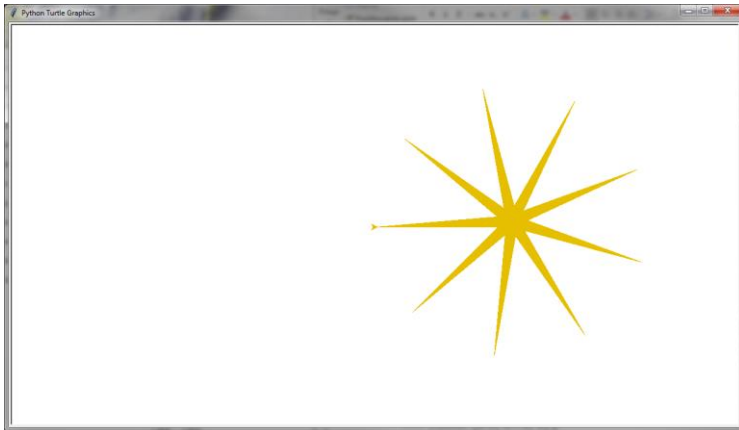
Sedaj bomo naredili funkcijo `mystar`. Uporabili bomo `if` stavek funkcije `mysquare` in dodali parameter velikost.

```
>>> def mystar(velikost, zapolnjeno):
    if zapolnjeno == True:
        t.begin_fill()
    for x in range(1, 19):
        t.forward(velikost)
        if x % 2 == 0:
            t.left(175)
        else:
            t.left(225)
    if zapolnjeno == True:
        t.end_fill()
```

V prvih dveh vrsticah te funkcije preverimo, če je `zapolnjeno` `true` in če je, začnemo barvati. Ponovno preverjamo v zadnjih dveh vrsticah in če je `zapolnjeno` `True`, prenehamo z barvanjem. Tako kot pri funkciji `mysquare` prenesemo velikost zvezde v obliki parametra in uporabimo to vrednost, ko pokličemo `t.forward`. Nastavimo barvo na zlato (90% rdeča, 75% zelena in 0 odstotkov modre barve), nato pokličimo funkcijo.

```
>>> t.color(0.9, 0.75, 0)
>>> mystar(120, True)
```

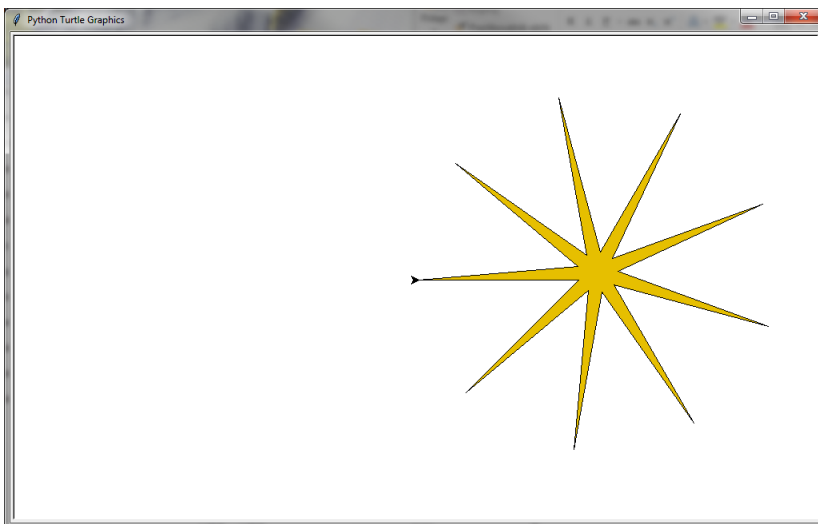
Želva bo pripravila to pobarvano zvezdo:



Če želite zvezdici dodati oris, spremenite barvo v črno in ponovno narišite zvezdo brez barvanja:

```
>>> t.color(0,0,0)
>>> mystar(120, False)
```

In zvezda je sedaj zlata s črnim orisom, kot ta:



Kaj ste se naučili

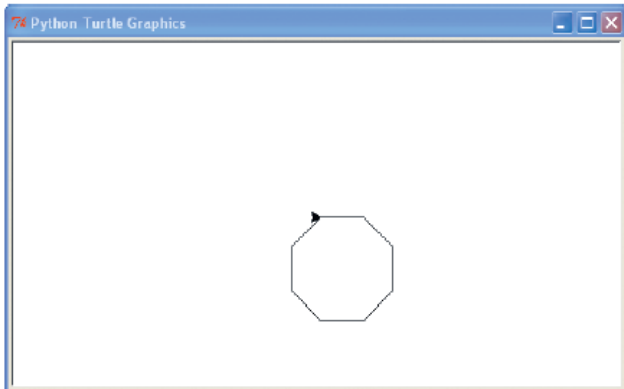
V tem poglavju ste se naučili kako uporabiti modul turtle za risanje nekaj osnovnih geometrijskih oblik z uporabo for zanke in if stavka za nadzor, kaj želva naredi na zaslonu. Spreminjali smo barvo in pobarvali oblike, ki smo jih narisali. Prav tako smo v funkcijah uporabili kodo za risanje in si s tem olajšali risanje oblik z možnostjo barvanja.

Vaje

V naslednjih poskusih boste izdelali svoje slike z želvo. Kot vedno so rešitve na voljo na <https://nostarch.com/pythonforkids>.

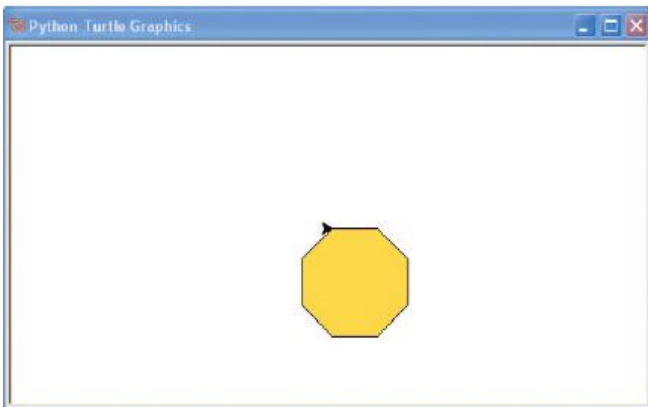
1: Risanje osemkotnika

V tem poglavju smo risali zvezde, kvadrate in pravokotnike. Kako pa bi ustvarili funkcijo za risanje osemkotnika? (Namig: poskusite obrniti želvo za 45 stopinj.)



2: Risanje pobarvanega osemkotnika

Zdaj, ko imate funkcijo za risanje osemkotnika, jo spremenite tako, da bo narisalo zapolnjen osemkotnik. Poskusite narisati osemkotnik z orisom, kot smo naredili pri zvezdici.



3: Druga funkcija za risanje zvezd

Ustvarite funkcijo za risanje zvezdice, ki bo imela dva parametra: velikost in število krakov. Začetek funkcije naj bi izgledal takole:

```
def narisi_zvezdo(velikost, kraki):
```

12. poglavje: Uporaba tkinter za boljšo grafiko

Težava pri uporabi želve za risanje je, da ... so ... želve ... res ... zelo ... počasne. Tudi, ko je želva pri največji hitrosti, še vedno ne gre hitro. To sicer res ni problem za želve, je pa problem za računalniško grafiko.

Računalniška grafika, zlasti v igrah, mora omogočati hitro gibanje. Če imate igralno konzolo ali igrate igre na računalniku, pomislite za trenutek na grafiko, kar vidite na zaslonu. Dvodimenzionalna (2D) grafika je v ravnini - liki se v igrah običajno premikajo samo gor in dol ali levo in desno, kot v mnogih Nintendo DS, prenosnem PlayStation (PSP) in mobilnih telefonih. V psevdo-tridimenzionalnih (3D) igrah - so slike bolj resnične, vendar se liki običajno premikajo le v ravnini (to je znano tudi kot izometrična grafika). In končno, imamo 3D igre, kjer se slike na zaslonu zdijo resnične. Naj gre za igre z 2D, pseudo-3D ali 3D grafiko, vse imajo eno skupno stvar: potrebo, da se kar najhitreje izrišejo na zaslonu računalnika.

Če še nikoli niste poskušali ustvariti lastne animacije, poskusite ta preprost projekt:

1. Poiščite prazen zvezek in v spodnjem kotu prve strani nekaj narišite (morda pajaca iz črt).
2. Na vogalu naslednje strani narišite isto sliko, vendar z nekoliko premaknjeno nogo.
3. Na naslednji strani ponovno narišite z nogo še bolj premaknjeno.
4. Postopoma na vse strani narišite sličice, vsakič z majhno spremembo.

Ko končate, hitro prelistajte strani in videti bi morali premikanje vaše figure. To je osnovna metoda, ki se uporablja z vsemi animacijami, ne glede na to ali gre za risanke na TV ali igre na vaši konzoli ali računalniku. Slika se izriše in nato ponovno izriše z rahlo spremembo, da bi ustvarila iluzijo gibanja. Da bi slika izgledala kot, da se premika, morate prikazati vsak okvir ali del animacije zelo hitro.

Python ponuja različne načine za ustvarjanje grafike. Poleg modula turtle, lahko uporabite zunanje module (ki jih je treba namestiti posebej), kot tudi tkinter modul, ki ga imate v običajni namestitvi Pythona. Tkinter se lahko uporablja za ustvarjanje celih aplikacij, kot je preprost urejevalnik besedil ali risarski program. V tem poglavju bomo raziskali kako uporabljati tkinter za ustvarjanje grafike.

Ustvarjanje gumba za klikanje

Za naš prvi primer bomo z uporabo tkinterja ustvarili osnovno aplikacijo z gumbom. Vnesite to kodo:

```
>>> from tkinter import *
>>> tk = Tk()
>>> btn = Button(tk, text = "Klikni me")
>>> btn.pack()
```



V prvi vrstici uvozimo vsebino modula tkinter. Uporaba s from ime-modula import * omogoča, da uporabimo vsebino modula brez vsakokratnega dodajanja imena modula. V nasprotju z uporabo v prejšnjih primerih, ko smo morali vključiti ime modula za dostop do njegove vsebine:

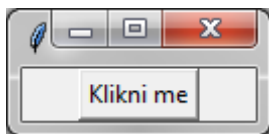
```
import turtle
t = turtle.Pen()
```

Ko uporabljamo import *, nam ni treba klica turtle.Pen, kot smo delali v poglavjih 4 in 11. To ni tako uporabno pri modulu turtle, je pa uporabno takrat, ko uporabljate module z veliko razredi in funkcijami, saj ni potrebnega toliko pisanja.

```
from turtle import *  
t = Pen()
```

V naslednji vrstici (v našem primeru gumba) ustvarimo spremenljivko, ki vsebuje objekt razreda Tk s tk = Tk(), podobno kot smo ustvarili objekt Pen pri želvi. Tk objekt ustvari osnovno okno na katerega lahko nato dodajamo druge stvari, kot so gumbi, vnosna polja ali platno za risanje. To je glavni razred, ki ga zagotavlja modul tkinter - brez ustvarjanja objekta razreda Tk ne bi mogli narediti grafike ali animacije.

V tretji vrstici ustvarimo gumb z btn = Button in prenesemo spremenljivko tk kot prvi parameter in »klikni me« kot besedilo, s katerim se bo prikazal gumb (tk, text = "klikni me"). Čeprav smo ta gumb dodali v okno, ne bo prikazan dokler ne vnesete vrstice btn.pack(), ki omogoči prikaz gumba. Prav tako uredi vse elemente za pravi prikaz na zaslonu. Rezultat bi moral biti nekaj takega:



Gumb »Klikni me« še ne dela veliko. Lahko ga klikate ves dan, vendar se ne bo nič zgodilo, dokler ne bomo nekoliko spremenili kode. (Zaprte okno, ki ste ga ustvarili prej!) Najprej ustvarimo funkcijo za izpis določenega besedila:

```
>>> def pozdrav():  
print('Živijo')
```

Potem spremenimo naš primer, da uporabimo to novo funkcijo:

```
>>> from tkinter import *  
>>> tk = Tk()  
>>> btn = Button(tk, text = "klikni me", command = pozdrav)  
>>> btn.pack()
```

Upoštevajte, da smo le malo spremenili prejšnji različica te kode: dodali smo parameter command, ki pove naj Python uporablja funkcijo pozdrav, ko je kliknjen gumb. Ko kliknete gumb, boste v lupini videli izpisano "Živijo". To se prikaže vsakič, ko kliknete gumb.

To je prvič, da smo uporabili imenovane parametre v katerem koli primeru naše kode, zato se malo pogovorimo o njih, pred nadaljevanjem z našo risbo.

Uporaba imenovanih parametrov

Imenovani parametri so enaki kot običajni parametri, razen tega, da namesto uporabe posebnega vrstnega reda parametrov, ki so določeni s funkcijo, tu poimenujemo parameter in vrstni red parametrov ni pomemben. Včasih imajo funkcije veliko parametrov in morda ne potrebujemo vrednosti za vsak parameter. Imenovani parametri so način, ko lahko zagotovimo vrednosti samo za parametre, ki jim moramo določiti vrednost. Recimo, da imamo funkcijo, imenovano oseba, ki ima dva parametra: širino in višino.

```
>>> def oseba(sirina, visina):  
print("Sem %s cm širok, %s cm visok" %(sirina, visina))
```

Običajno lahko to funkcijo kličemo takole:

```
>>> oseba(50, 180)  
Jaz sem 50 cm širok, 180 cm visok
```

Pri uporabi imenovanih parametrov, bi lahko poklicali to funkcijo in ji določili imena parametrov z vrednostmi:

```
>>> oseba(visina = 180, sirina = 50)  
Jaz sem 50 cm širok, 180 cm visok
```

Imenovani parametri bodo postali še posebej uporabni, ko bomo več delali z modulom tkinter.

Ustvarjanje platna za risanje

Gumbi so fina orodja, vendar niso posebej uporabna, ko želimo nekaj narisati na zaslonu. Ko želimo nekaj narisati, potrebujemo drugačno sestavino: platno (canvas), ki je objekt razreda Canvas (ki ga ponuja modul tkinter).

Za kreiranje platna Pythonu sporočimo širino in višino platna (v piksljih). Sicer pa je koda podobna kodi za gumb. Tukaj je primer:

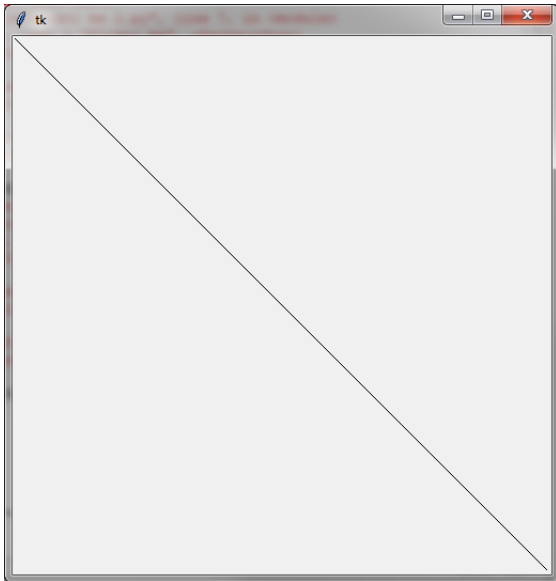
```
>>> from tkinter import *  
>>> tk = Tk()  
>>> platno = Canvas(tk, width = 500, height = 500)  
>>> platno.pack()
```

Kot v primeru gumba se bo pojavilo okno aplikacije, ko vnesemo tk = Tk (). V zadnji vrstici, platno postavimo na zaslon s platno.pack (), ki nastavi velikost platna na širino 500 pik in višino 500 pik, kot je določeno v tretji vrstici kode. Kot pri gumbu, tudi tu funkcija pack razporedi elemente na zaslon. Če te funkcija ne kličemo, nič ne bo pravilno prikazano.



Risanje črt

Če želimo na platnu narisati črto, uporabimo koordinate v točkah. Koordinate določajo položaj na površini. Na tkinter platnu koordinate opisujejo, kako daleč proti desni (od leve proti desni) in kako daleč navzdol (od zgoraj navzdol) naj postavimo sliko. Ker je naše platno široko 500 točk in 500 točk visoko, so koordinate spodnjega desnega kota zaslona (500, 500). Če želimo narisati črto, prikazano na naslednji sliki, bomo uporabili začetne koordinate (0, 0) in končne koordinate (500, 500).



Koordinate določimo z uporabo funkcije `create_line`, kot je prikazano tule:

```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 500, height = 500)
>>> platno.pack()
>>> platno.create_line(0, 0, 500, 500)
1
```

Funkcija `create_line` vrne 1, ki je identifikator - o tem bomo več izvedeli pozneje. Če bi isto stvar naredili z modulom `turtle`, bi potrebovali naslednjo kodo:

```
>>> import turtle
>>> turtle.setup(width = 500, height = 500)
>>> t = turtle.Pen()
>>> t.up()
>>> t.goto(-250, 250)
>>> t.down()
>>> t.goto(500, -500)
```

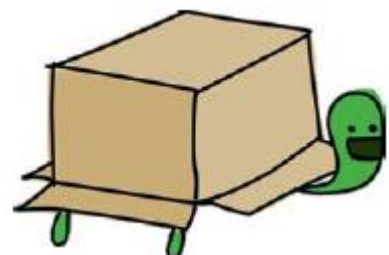
Torej je `tkinter` že boljši. Rahlo je krajši in nekoliko enostavnejši.

V nadaljevanju si oglejmo nekatere funkcije, ki so na voljo objektom `canvas` (`platno`), ki jih lahko uporabimo za bolj zanimive slike.

Risanje likov

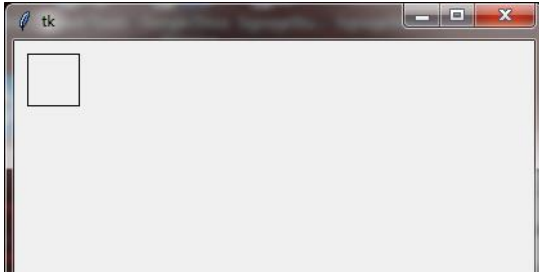
Z modulom `turtle` smo narisali kvadrat s premikanjem naprej, obračanjem, premikanjem naprej, ponovno obračanje in tako naprej. Naredili smo pravokotnik ali kvadrat, tako da smo spremenili dolžino premikanja.

V modulu `tkinter` je precej lažje narediti kvadrat ali pravokotnik. Vse, kar moramo vedeti, so koordinate za kote. Tukaj je primer (prejšnja okna lahko zaprete):



```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> platno.create_rectangle(10, 10, 50, 50)
```

V tej kodi uporabimo tkinter za izdelavo platna velikosti 400 pik, širine 400 pik in nato v zgornjem levem kotu narišemo kvadratek, kot je ta:

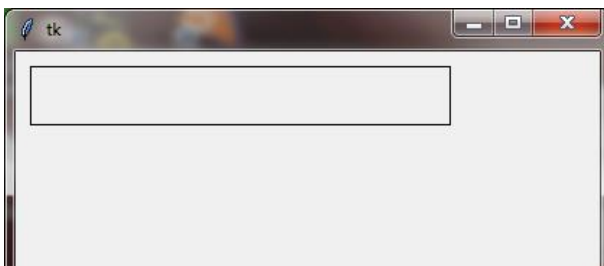


Parametre, ki smo jih prenesli v `platno.create_rectangle` v zadnji vrstici so koordinate za zgornji levi in spodnji desni kot kvadrata. Te koordinate zagotavljamo kot razdaljo z leve strani platna in oddaljenost od vrha platna. V tem primeru sta prvi dve koordinati (zgornji levi kot) je 10 slikovnih pik levo in 10 slikovnih pik navzdol (to so prve številke: 10, 10). Spodnji desni kot kvadrata je 50 slikovnih pik od leve in 50 slikovnih pik navzdol (druga števila: 50, 50).

Te skupine koordinat bomo imenovali `x1`, `y1` in `x2`, `y2`. Za risanje pravokotnika lahko povečamo razdaljo drugega kota od stranice platna (povečamo vrednosti parametra `x2`):

```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> platno.create_rectangle(10, 10, 300, 50)
```

V tem primeru so zgornje leve koordinate pravokotnika (njen položaj na zaslonu) 10, 10 in koordinate spodaj-desno 300, 50. Rezultat je pravokotnik, ki ima enako višino kot naš prvotni kvadrat (50 pikslov), a veliko širši.



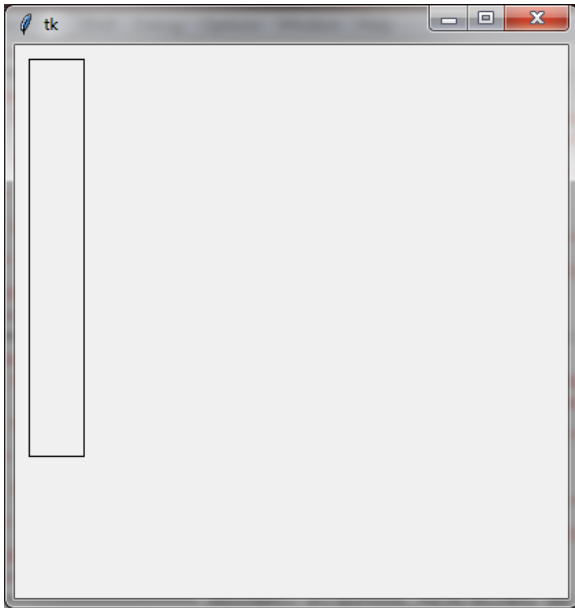
Prav tako lahko narišemo pravokotnik s povečanjem razdalje drugega kota od vrha platna (povečanje vrednosti parameter `y2`):

```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> platno.create_rectangle(10, 10, 50, 300)
```

V tem klicu funkcije `create_rectangle` smo v bistvu rekli:

- Pojdi 10 pik po platnu v desno (od zgoraj levo).
- Pojdi po 10 pik po platnu navzdol. To je prvi kot pravokotnika.
- Pravokotnik nariši do 50 pik v desno.
- Riši dol do 300 pik.

Končni rezultat bi moral biti nekaj takega:



Risanje množice pravokotnikov

Kaj pa poljenje platna z različnimi pravokotniki? To lahko storimo z modulom `random` in nato ustvarimo funkcijo, ki uporablja naključna števila za koordinate zgornjega levega in spodnjega desnega kota pravokotnika.

Uporabili bomo funkcijo `randrange` iz modula `random`. Ko to funkcijo kličemo s številko, vrne naključno celo število med 0 in vnesenim številom. Na primer, klic `randrange(10)` bi vrnil število med 0 in 9, `randrange(100)` vrne število med 0 in 99, in tako naprej.

Ustvarimo novo okno z izbiro File, New Window in vnesimo naslednjo kodo:

```
from tkinter import *
import random
tk = Tk()
platno = Canvas(tk, width = 400, height = 400)
platno.pack()
def nakljucni_pravokotnik(sirina, visina):
    x1 = random.randrange(sirina)
    y1 = random.randrange(visina)
    x2 = x1 + random.randrange(sirina)
    y2 = y1 + random.randrange(visina)
    platno.create_rectangle(x1, y1, x2, y2)
```

Najprej definiramo funkcijo (`def nakljucni_pravokotnik`) z dvema parametroma: `sirina` in `visina`. Nato ustvarimo spremenljivke za zgornji levi kot pravokotnika z uporabo `randrange` funkcije z `x1 =`

random.randrange (sirina) in y1 = random.randrange (visina). V bistvu druga vrstica te funkcije pravi: "Ustvarite spremenljivko x1 in nastavite vrednost na naključno število med 0 in vrednostjo v parametru sirina."

Naslednji dve vrstici ustvarita spremenljivki spodnjega desnega kota pravokotnika, pri čemer upoštevamo zgornje leve koordinate (bodisi x1 ali y1) in dodamo naključno število tem vrednostim. Tretja vrstica funkcije pravi: "Ustvarite spremenljivko x2 z vrednostjo že ustvarjene x1 povečane za naključno vrednost."

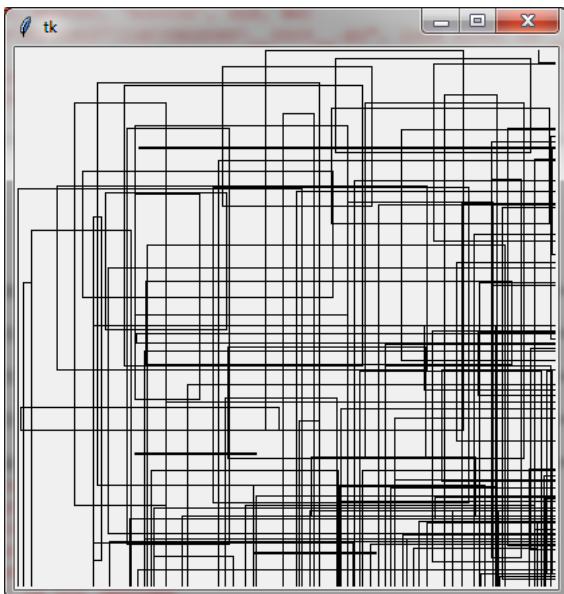
Končno, s platno.create_rectangle, uporabimo spremenljivke x1, y1, x2 in y2 za risanje pravokotnika na platno. Če želite preizkusiti našo funkcijo nakljucni_pravokotnik, mu bomo posredovali širino in višina platna. Dodajte še spodnjo kodo za klic funkcije:

```
nakljucni_pravokotnik(400, 400)
```

Shranite kodo, ki ste jo vnesli (izberite File, Save in vnesite ime datoteke, kot je randomrect.py) in nato izberite Run, Run Modul. Ko vidite, da funkcijo deluje, zapolnite zaslon s pravokotniki z ustvarjanjem zanke, da funkcijo nakljucni_pravokotnik večkrat pokličete. Poskusimo s for zanko za 100 naključnih pravokotnikov. Dodajte naslednje kodo, shranite svoje delo in poskusite znova zagnati:

```
for x in range(0, 100):  
    nakljucni_pravokotnik(400, 400)
```

Ta koda na platnu izgleda kot nekakšna moderna umetnost:



Nastavljanje barv

Seveda želimo naši grafiki dodati barve. Spremenimo funkcijo nakljucni_pravokotnik, da se barva za pravokotnik vnese kot dodatni parameter (barva). Vnesite to kodo v novo okno, in ko shranite, pokličite datoteko colorrect.py:

```
from tkinter import *  
import random  
tk = Tk()  
platno = Canvas(tk, width = 400, height = 400)
```

```

platno.pack()
def nakljucni_pravokotnik(sirina, visina, barva):
    x1 = random.randrange(sirina)
    y1 = random.randrange(visina)
    x2 = x1 + random.randrange(sirina)
    y2 = y1 + random.randrange(visina)
    platno.create_rectangle(x1, y1, x2, y2, fill=barva)

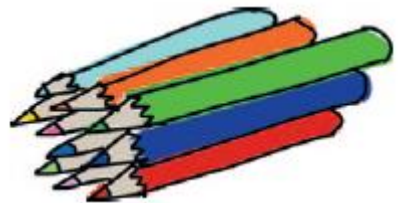
```

Funkcija `create_rectangle` sedaj zajema parameter `fill_color`, ki določa barvo, ki jo je treba uporabiti pri risanju pravokotnika. V tako funkcijo lahko prenesemo poimenovane barve (z uporabo `platno` 400 slikovnih pik široko za 400 slikovnih pik visok), da ustvarimo nekaj različnih barvnih pravokotnikov. Za ta primer, lahko uporabite kopiraj in prilepiti. To storite tako, da označite besedilo, ki ga želite kopirati, pritisnite `ctrl-C`, da ga kopirate, kliknite prazno vrstico in pritisnite `ctrl-V`, da ga prilepite. Dodajte to kodo v `colorrect.py`, tik pod funkcijo):

```

nakljucni_pravokotnik(400, 400, 'green')
nakljucni_pravokotnik(400, 400, 'red')
nakljucni_pravokotnik(400, 400, "blue")
nakljucni_pravokotnik(400, 400, 'orange')
nakljucni_pravokotnik(400, 400, 'yellow')
nakljucni_pravokotnik(400, 400, 'pink')
nakljucni_pravokotnik(400, 400, 'purple')
nakljucni_pravokotnik(400, 400, 'violet')
nakljucni_pravokotnik(400, 400, 'magenta')
nakljucni_pravokotnik(400, 400, 'cyan')

```



Mnoge od teh imenovanih barv bodo prikazale barvo, ki jo pričakujete, nekatere pa lahko prikažejo sporočilo o napaki (odvisno od tega ali uporabljate Windows, Mac OS X ali Linux).

Kaj pa barva po meri, ki ni poimenovana? V poglavju 11 smo nastavili barvo peresa z uporabo odstotkov barv rdeče, zelene in modre barve. Nastavitev količine vsake primarne barve (rdeča, zelena in modra) za uporabo v barvni kombinaciji s tkinterjem je nekoliko bolj zapleteno.

Ko smo delali z modulom `turtle`, smo ustvarili zlato barvo: 90% rdeče, 75% zelene in brez modre barve. V tkinterju lahko z naslednjo vrstico ustvarite isto zlato barvo:

```

nakljucni_pravokotnik(400, 400, '# ffd800')

```

Oznaka lojtra (hash - #) pred vrednostjo `ffd800` pove Pythonu, da mu dajemo šestnajstiško (hexadecimal) število. Šestnajstiška predstavitev številke se pogosto uporablja v računalniškem programiranju. Za osnovo uporablja število 16 (števke od 0 do 9, nato pa od A do F) in desetiška, ki ima osnovo 10 (števke od 0 do 9). Če se še niste učili številskih sistemov na drugačnih osnovah, je dovolj, če veste, da lahko pretvorite normalno decimalno številko v šestnajstiško obliko z uporabo oblike izpisa niza: `%x` (glejte "Vsebovane vrednosti v nizu"). Na primer, za pretvorbo decimalne številke 15 v heksadecimalno, lahko to storite takole:

```

>>> print('%x' % 15)
f

```

Če želimo, da se število izpiše na dve mesti, damo naslednji ukaz:

```

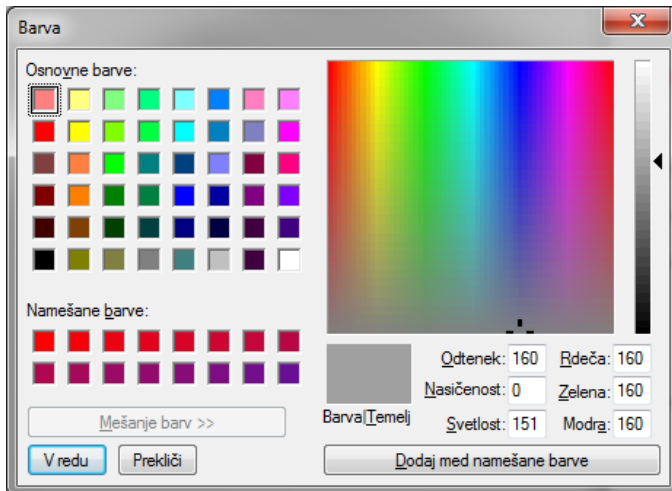
>>> print('%02x' % 15)
0f

```

Modul tkinter omogoča enostaven dostop do heksadecimalne vrednosti barve. Poskusite dodati naslednjo kodo v vaš program colorrect.py (lahko odstranite druge klice funkcije naključni_pravokotnik).

```
from tkinter import *
from tkinter import colorchooser
print(colorchooser.askcolor())
```

To vam pokaže barvno izbiro:



Ko izberete barvo in kliknete V redu, se prikaže tuple. Ta vsebuje tri številke in niz:

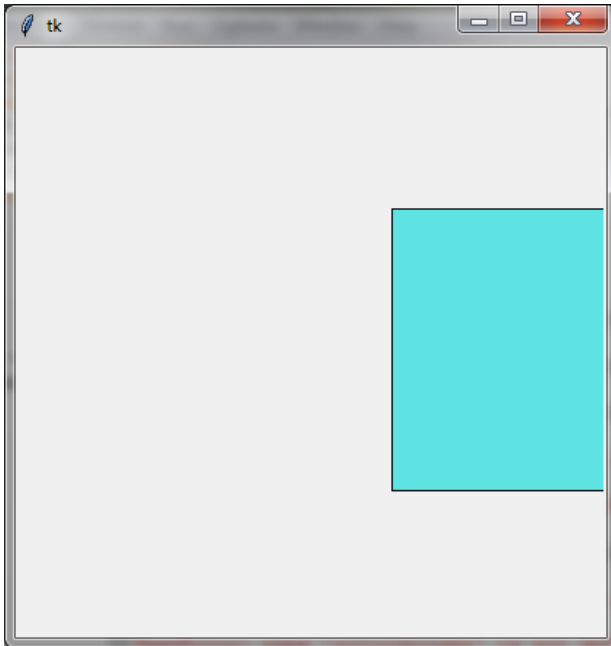
```
>>> print(colorchooser.askcolor())
((235.91796875, 86.3359375, 153.59765625), '# eb5699')
```

Tri številke predstavljajo količine rdeče, zelene in modre. V tkinter je količina vsake primarne barve za uporabo v barvni kombinaciji predstavljena s številom med 0 in 255 (ki se razlikuje od uporabe odstotka za vsako primarno barvo v modulu turtle). Niz v tuple vsebuje heksadecimalno različico teh treh števil. Lahko kopirate in prilepite vrednost niza za uporabo ali shranite tuple kot spremenljivko, nato pa uporabite indeksni položaj šestnajstiške vrednosti.

Uporabimo funkcijo naključni_pravokotnik, da vidimo, kako to deluje.

```
>>> c = colorchooser.askcolor()
>>> naključni_pravokotnik(400, 400, c [1])
```

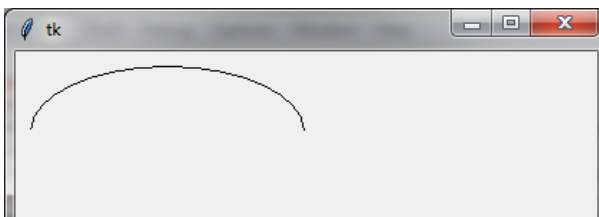
Tukaj je rezultat:



Risanje lokov

Lok je del obsega kroga ali druge krivulje. Da bi ga narisali s tkinter uporabimo `create_arc` funkcijo, s kodo, kot je ta:

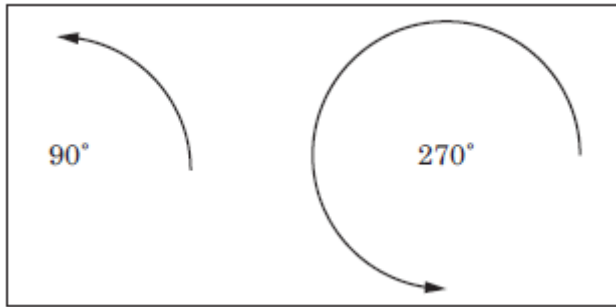
```
platno.create_arc(10, 10, 200, 100, extent = 180, style = ARC)
```



Če ste zaprli vsa okna tkinterja ali ponovno zagnali IDLE, poskrbite, da znova uvozite tkinter in nato ustvarite platno s to kodo:

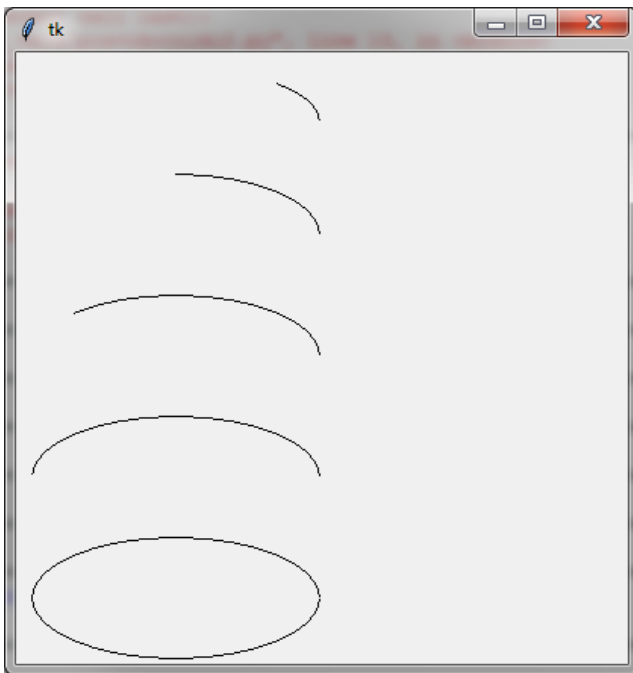
```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> platno.create_arc(10, 10, 200, 100, extent = 180, style = ARC)
```

Ta koda postavi zgornji levi kot pravokotnika, v katerem bo narisan lok, na koordinato (10, 10), kar je 10 pik v desno in 10 pik navzdol in na koordinato (200, 100) ali 200 pik desno in 100 pik navzdol. Naslednji parameter, `extent`, se uporablja za določitev stopinj kota loka. Spomnimo se iz poglavja 4, da so stopinje način merjenja razdalje za potovanje po krogu. Tukaj sta primera dveh lokov, kjer potujemo 90 stopinj in 270 stopinj po krogu:



Naslednja koda nariše več različnih lokov na platno tako, da lahko vidite, kaj se zgodi, ko uporabljamo različne stopinje s funkcijo `create_arc`.

```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> platno.create_arc(10, 10, 200, 80, extent = 45, style = ARC)
>>> platno.create_arc(10, 80, 200, 160, extent = 90, style = ARC)
>>> platno.create_arc(10, 160, 200, 240, extent = 135, style = ARC)
>>> platno.create_arc(10, 240, 200, 320, extent = 180, style = ARC)
>>> platno.create_arc(10, 320, 200, 400, extent = 359, style = ARC)
```



Opomba: V zadnjem liku uporabljamo 359 stopinj, namesto 360, ker tkinter meni, da je 360 enako 0 stopinj in ne bi nič narisal, če bi uporabili 360.

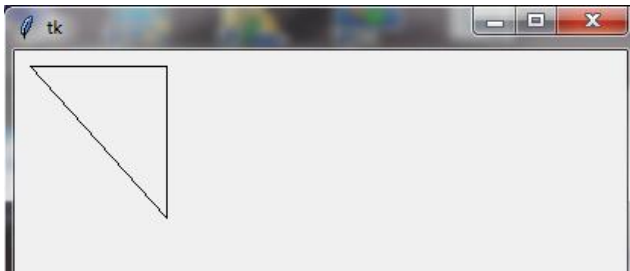
Risanje večkotnikov

Poligon je vsaka oblika s tremi ali več stranicami. Obstajajo pravilni večkotniki, kot so trikotnik, kvadrat, pravokotnik, petkotnik, šestkotnik in tako naprej, kot tudi nepravilni z različnimi stranicami, veliko več stranicami in čudnih oblik.

Pri risanju poligonov s tkinterjem morate vnesti koordinate za vsako točko mnogokotnika. Takole lahko narišemo trikotnik:

```
from tkinter import *
tk = Tk()
platno = Canvas(tk, width = 400, height = 400)
platno.pack()
platno.create_polygon(10, 10, 100, 10, 100, 110, fill = '',outline =
'black')
```

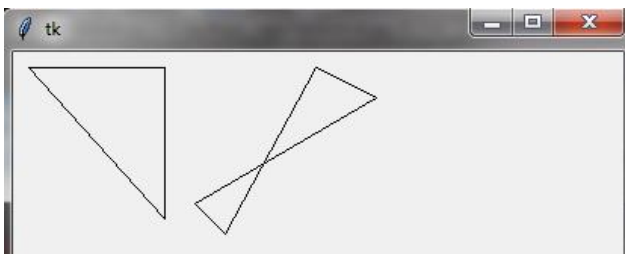
Ta primer nariše trikotnik z začetkom v x in y koordinate (10, 10), nato gre do (100, 10) in konča v oglišču (100, 110). Tukaj je rezultat:



Dodamo lahko še en nepravilni poligon (oblika s križanjem stranic) z uporabo te kode:

```
platno.create_polygon(200, 10, 240, 30, 120, 100, 140, 120, fill = '',
outline = 'black')
```

Ta koda se začne s koordinatami (200, 10), se premakne na (240, 30), nato na (120, 100) in nazadnje na (140, 120). tkinter avtomatično zaključi lik s povezavo do začetne točke. In tukaj je rezultat izvajanja kode:



Prikazovanje besedila

Poleg risanja oblik, po platnu lahko tudi pišemo besedila `create_text`. Ta funkcija ima le dve koordinati (x in y položaj besedila), skupaj z imenovanim parametrom za besedilo za izpis. V naslednji kodi ustvarimo naše platno in nato prikažemo stavek na koordinatah (150, 100). Shranite to kodo kot `text.py`.

```
from tkinter import *
tk = Tk()
platno = Canvas(tk, width = 400, height = 400)
platno.pack()
platno.create_text(150, 100, text = 'Živel je mož, imel je psa,')
```

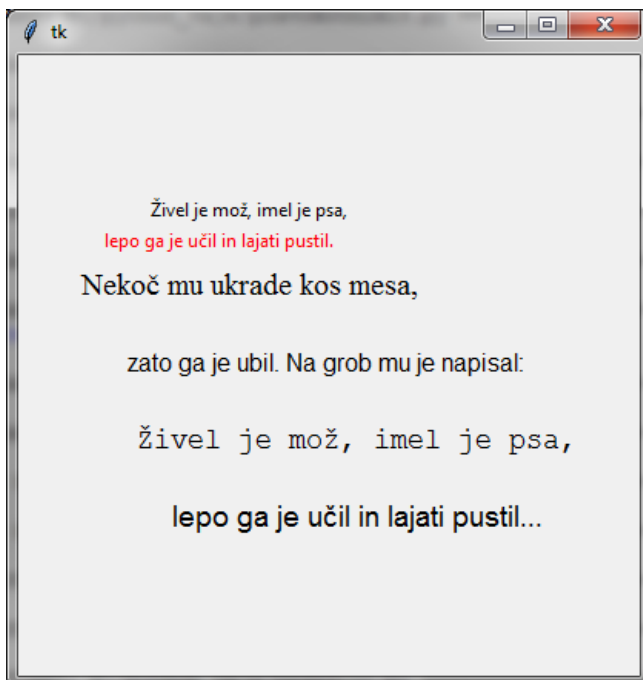
Funkcija `create_text` ima nekaj drugih uporabnih parametrov, kot na primer barva pisanja besedila. V naslednji kodi pokličemo `create_text` funkcijo z koordinatami (130, 120), besedilo, ki ga želimo za prikaz in rdečo barvo polnila:

```
platno.create_text(130, 120, text = 'lepo ga je učil in lajati pustil.',  
fill = 'red')
```

Lahko določimo tudi pisavo (ki se uporablja za prikazano besedilo) kot tuple z imenom pisave in velikostjo besedila. Na primer, tuple za pisavo Times velikosti 20 je ("Times", 20). V nadaljevanju kode, prikažemo besedilo z uporabo pisave Times nastavljeno na velikost 15, pisave Helvetica velikosti 12 in pisave Courier z velikostjo 14.

```
platno.create_text(150, 150, text = 'Nekoč mu ukrade kos mesa,', font  
=('Times', 15)) platno.create_text(200, 200, text = 'zato ga je ubil. Na  
grob mu je napisal:',font =('Helvetica', 12))  
platno.create_text(220, 250, text = 'Živel je mož, imel je psa,',font  
=('Courier', 14))  
platno.create_text(220, 300, text = 'lepo ga je učil in lajati pustil...',  
font =(' Courier ', 14))
```

In tukaj je rezultat teh funkcij z uporabo treh navedenih pisav v različni velikosti:



Prikazovanje slik

Za prikaz slike na platnu, ki uporablja tkinter, najprej naložite sliko in nato uporabite funkcijo `create_image`.

Vsaka slika, ki jo naložite, mora biti v imeniku, ki je dostopen za Python. V tem primeru imamo sliko `test.png` (slika je lahko GIF ali PNG) v korenskem imeniku `C:\`, (osnovni imenik pogona `C:`) lahko pa jo imate kjerkoli.

Sliko lahko prikažemo na naslednji način:

```
from tkinter import *  
tk = Tk()  
platno = Canvas(tk, width = 400, height = 400)  
platno.pack()  
moja_slika = PhotoImage(file = 'c:\\test.png')  
platno.create_image(0, 0, anchor = NW, image = moja_slika)
```

V prvih štirih vrsticah smo postavili platno kot v prejšnjih primerih. V peti vrstici se slika naloži v spremenljivko `moja_slika`. Ustvarimo `PhotoImage` z imenikom `'c:\\test.png'`. Če ste shranili sliko na namizje, ustvarite `PhotoImage` z nečim podobnim:

```
moja_slika = PhotoImage(file = 'C:\\Uporabniki\\Joe Smith\\namizje\\test.png')
```

Ko je slika naložena v spremenljivko, `platno.create_image(0, 0, anchor = NW, image = moja_slika)` prikaže z uporabo `create_image` funkcije. Koordinate `(0, 0)` so položaj, kjer bo slika prikazana in `anchor = NW` pove funkciji, da uporabi zgornji levi (NW, za severozahodno) rob slike kot izhodišče pri risanju (v nasprotnem primeru bo uporabljeno središče slike). Zadnji imenovani parameter, `image`, kaže na spremenljivko za naloženo sliko. Tukaj je rezultat:



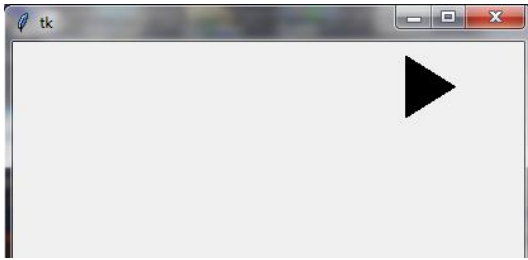
Ustvarjanje osnovne animacije

Obravnavali smo, kako ustvariti statične risbe - slike, ki jih ne premikate. Kaj pa ustvarjanje animacije?

Animacija ni nujno posebnost modula `tkinter`, vendar pa lahko obvladuje osnove. Na primer, ustvarimo lahko zapolnjen trikotnik in ga nato s pomočjo te kode premaknemo po zaslonu (ne pozabite, izberite `File, New Window`, shranite svoje delo in nato zaženite kodo z modulom `Run, Run Modul`):

```
import time
from tkinter import *
tk = Tk()
platno = Canvas(tk, width = 400, height = 200)
platno.pack()
platno.create_polygon(10, 10, 10, 60, 50, 35)
for x in range(0, 60):
    platno.move(1, 5, 0)
    tk.update()
    time.sleep(0.05)
```

Ko zaženete to kodo, se bo trikotnik začel gibati čez zaslon do konca svoje poti:



Kako to deluje? Podobno kot prej smo uporabili prve tri vrstice po uvozu tkinter za osnovni prikaz platna. V četrty vrstici s to funkcijo ustvarimo trikotnik:

```
platno.create_polygon(10, 10, 10, 60, 50, 35)
```

Opomba: Ko vnesete to vrstico, bo na zaslonu natisnjena številka. To je identifikator za večkotnik. Lahko ga uporabimo za sklicevanje nanj, kot je opisano v naslednjem primeru.

Nato ustvarimo preprosto for zanko, ki šteje od 0 do 59, for x in range (0, 60):. Blok kode znotraj zanke premakne trikotnik preko zaslona. Funkcija platno.move bo premakniti poljuben narisan element z dodajanjem vrednosti x in y koordinatam. Na primer, s platno.move (1, 5, 0) premikamo objekt z ID 1 (identifikator za trikotnik) 5 pik desno in 0 pik navzdol. Če ga želite vrniti, lahko uporabimo funkcijski klic platno.move (1, -5, 0).

Funkcija tk.update () obnovi prikaz, da posodobi zaslon (ga ponovno izriše). Če ne bi uporabili update, bi tkinter počakal do konca zanke in šele nato premaknil trikotnik, kar pomeni, da bi ga videli skočiti na končni položaj, namesto da se gladko premika čez platno. Zadnja vrstica zanke, time.sleep (0.05), pove Python naj počaka 5 stotink sekunde (0.05 sekunde), preden nadaljuje.

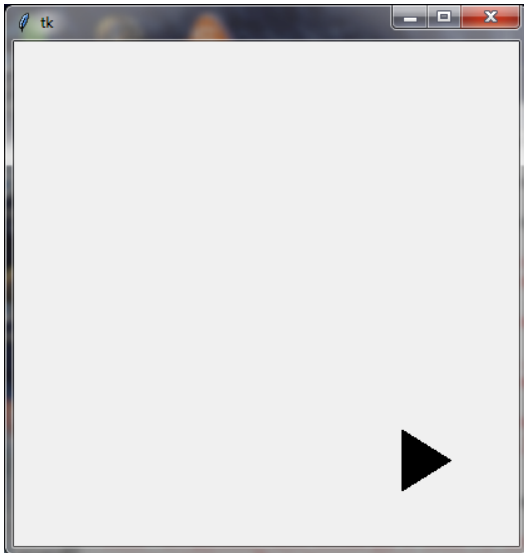
Da bi se trikotnik premaknil diagonalno po zaslonu, lahko kodo spremenimo tako, da pokliče move(1, 5, 5). Če želite poskusiti, zaprite platno in ustvarite novo datoteko (File, New Window) z naslednjo kodo:

```
import time
from tkinter import *
tk = Tk()
platno = Canvas(tk, width = 400, height = 400)
platno.pack()
platno.create_polygon(10, 10, 10, 60, 50, 35)
for x in range(0, 60):
    platno.move(1, 5, 5)
    tk.update()
    time.sleep(0.05)
```

Ta koda se razlikuje od izvornika v dveh podrobnostih:

- višina platna je 400, namesto 200, s platno = Canvas (tk, width = 400, height = 400).
- obema koordinatama x in y dodamo 5 platno.move (1, 5, 5).

Ko shranite kodo in jo poženete, dobite naslednji položaj trikotnika:



Če želite trikotnik premakniti diagonalno nazaj na začetni položaj, uporabite -5, -5 (dodajte to kodo pod zadnjo vrstico):

```
for x in range(0, 60):  
    platno.move(1, -5, -5)  
    tk.update()  
    time.sleep(0.05)
```

Odziv objekta na dogodek

Lahko naredimo, da se bo trikotnik premaknil ob kakšnem dogodku (event bindings). Dogodki so stvari, ki se zgodijo med izvajanjem programa, ko nekdo premika miško, pritisne tipko ali zapre okno. Tkinterju lahko poveste, naj opazuje dogodke in potem reagira nanje.

Za delo z dogodki (da Python nekaj naredi, ko se dogodek pojavi) najprej ustvarimo funkcijo. Program jo samodejno pokliče, ko se dogodek pojavi.

Če želite na primer premakniti trikotnik ob pritisku na enter, lahko definiramo to funkcijo:

```
def premakni_trikotnik(event):  
    platno.move(1, 5, 0)
```

Funkcija ima en parameter (event), ki ga tkinter uporablja za pošiljanje informacij o dogodku. Sedaj povemo tkinter, da je treba to funkcijo uporabiti za določen dogodek, z uporabo funkcije bind_all na platnu. Polna koda je videti takole:

```
from tkinter import *  
tk = Tk()  
platno = Canvas(tk, width = 400, height = 400)  
platno.pack()  
platno.create_polygon(10, 10, 10, 60, 50, 35)  
def premakni_trikotnik(event):  
    platno.move(1, 5, 0)  
platno.bind_all('<KeyPress-Return>', premakni_trikotnik)
```



Prvi parameter v tej funkciji opisuje dogodek, ki ga mora tkinter spremljati. V tem primeru se imenuje <KeyPress-Return>, ki je pritisk na tipko enter. Tkinterju povemo naj izvede funkcijo premakni_trikotnik, ko se zgodi pritisk na enter. Poženite to kodo, kliknite z miško v platno in nato poskusite pritisniti tipko Enter.

Kaj pa spreminjanje smeri trikotnika glede na različne tipke, kot so smerne tipke? To ni problem. Samo funkcijo premakni_trikotnik moramo nekoliko spremeniti:

```
def premakni_trikotnik(dogodek):
    if dogodek.keysym == 'Up':
        platno.move(1, 0, -3)
    elif dogodek.keysym == 'Down':
        platno.move(1, 0, 3)
    elif dogodek.keysym == 'Left':
        platno.move(1, -3, 0)
    else:
        platno.move(1, 3, 0)
```

Dogodek objekta, ki se prenese na premakni_trikotnik, vsebuje več spremenljivk. Ena od teh spremenljivk se imenuje keysym (za simbol tipke), ki je niz in ima vrednost dejanske pritisnjene tipke. Vrstica if dogodek.keysym == 'Up': pravi, če je niz tipke 'Up', kliči platno.move s parametri (1, 0, -3), kot v naslednji vrstici. Če keysym vsebuje 'Down', kot v elif dogodek.keysym == 'Down': pokličemo s parametri (1, 0, 3), in tako naprej.

Ne pozabite, da je prvi parameter identifikacijska številka za lik, ki je narisana na platnu, druga je vrednost, ki jo je treba prišteti x (vodoravno) koordinati, tretja pa je vrednost, ki jo prištejemo y (navpična) koordinati.

Zatem povemo tkinterju, da bo funkcija premakni_trikotnik uporabljena za obvladovanje dogodkov za štiri različne tipke (gor, dol, levo in desno). V nadaljevanju je prikazana koda. Ko vnesete to kodo, bo znova lažje ustvariti novo okno lupine tako, da izberete File, New Window. Preden poženete kodo, jo shranite s smiselnim imenom datoteke, na primer premik_trikotnika.py.

```
from tkinter import *
tk = Tk()
platno = Canvas(tk, width = 400, height = 400)
platno.pack()
platno.create_polygon(10, 10, 10, 60, 50, 35)
def premakni_trikotnik(dogodek):
    (1)     if dogodek.keysym == 'Up':
    (2)         platno.move(1, 0, -3)
    (3)     elif dogodek.keysym == 'Down':
    (4)         platno.move(1, 0, 3)
    (5)     elif dogodek.keysym == "Left":
    (6)         platno.move(1, -3, 0)
    (7)     else:
    (8)         platno.move(1, 3, 0)
platno.bind_all('<KeyPress-Up>', premakni_trikotnik)
platno.bind_all('<KeyPress-Down>', premakni_trikotnik)
platno.bind_all('<KeyPress-Left>', premakni_trikotnik)
platno.bind_all('<KeyPress-Right>', premakni_trikotnik)
```

V prvi vrstici funkcije premakni_trikotnik preverimo, če niz tipke vsebuje "Up" (1). Če vsebuje, premaknemo trikotnik navzgor s funkcijo move s parametri 1, 0, -3 pri (2). Prvi parameter je identifikator trikotnika, drugi je vrednost premika na desno (ne želimo se premakniti vodoravno, zato

je vrednost 0), tretja pa je vrednost premika navzdol (-3 pik). Nato preverimo, če keysym vsebuje "Down" (3) in če je, premaknemo trikotnik navzdol (3 pik) (4). Nazadnje pogledamo, če je vrednost keysym "Left" (5) in če je, premaknemo trikotnik levo (- 3 pik) (6). Če se nobena od vrednosti ne ujema, se konča z else (7) in premakne v desno (8). Trikotnik se bo sedaj premikal v smeri smernih tipk.

Več načinov uporabe identifikatorja

Ko uporabljamo funkcijo `create_` v objektu `canvas`, na primer `create_polygon` ali `create_rectangle`, se vrne identifikator. To identifikacijsko številko se lahko uporablja z drugimi funkcijami platna, kot smo videli pri funkciji `move`:

```
>>> from tkinter import *
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> platno.create_polygon(10, 10, 10, 60, 50, 35)
1
>>> platno.move(1, 5, 0)
```

Težava s tem primerom je, da `create_polygon` ne bo vedno vrnil 1. Če ustvarite še druge like, bi lahko vrnilo 2, 3 ali celo 100 (odvisno od števila oblik, ki so bile ustvarjene). Če spremenimo kodo in shranimo vrnjeno vrednost v spremenljivko, lahko kasneje uporabimo to spremenljivko (in ne samo sklicevanje na številko 1) in koda bo delovala neglede na to kaj je bilo vrnjeno:

```
>>> moj_trikotnik = platno.create_polygon(10, 10, 10, 60, 50, 35)
>>> platno.move(moj_trikotnik, 5, 0)
```

Funkcija `move` omogoča premikanje predmetov po zaslonu z uporabo njihovega identifikatorja. Obstajajo tudi druge funkcije objekta `canvas`, ki lahko kaj spremenijo. Funkcija `itemconfig` se lahko uporabi za spreminjanje nekaterih parametrov oblike, kot je barva za polnjenje in oris. Recimo, da ustvarimo rdeč trikotnik:

```
>>> tk = Tk()
>>> platno = Canvas(tk, width = 400, height = 400)
>>> platno.pack()
>>> moj_trikotnik = platno.create_polygon(10, 10, 10, 60, 50, 35, fill =
'red')
```

Trikotnik lahko spremenimo v drugo barvo z elementom `itemconfig` in uporabimo identifikator kot prvi parameter. Naslednja koda pravi: "Spremenite barvo polnila predmeta, ki ga določa številka v spremenljivki `moj_trikotnik` v modro."

```
>>> platno.itemconfig(moj_trikotnik, fill = 'blue')
```

Trikotniku bi lahko dali tudi drugačno barvo orisa, ponovno uporabite identifikator kot prvi parameter:

```
>>> platno.itemconfig(moj_trikotnik, outline = 'red')
```

Kasneje se bomo naučili, kako narediti še druge spremembe na risbi, kot je skrivanje in ponovna vidnost. Pri pisanju iger v naslednjem poglavju bomo ugotovili, da je to zelo uporabno za spreminjanje sličic, ko so prikazane na zaslonu.

Kaj ste se naučili

V tem poglavju ste uporabili **tkinter** modul za izdelavo enostavnih **geometrijskih likov na platnu**, **prikazovanje slike in izdelali osnovno animacijo**. Naučili ste se, kako uporabljati **dogodke za premikanje risb s tipkami**, kar bo koristno, ko bomo začeli programirati igre. Naučili ste se, kako v tkinter ustvarjene funkcije vrnejo **identifikacijsko številko**, ki se lahko uporablja za premikanje ali spreminjanje drugih lastnosti (barva, rob).

Vaje

Poskusite naslednje, če želite preizkusiti modul tkinter in osnovne animacije. Za rešitve obiščite <https://nostarch.com/pythonforkids>.

1: Izpolnite zaslon s trikotniki

Ustvarite program z uporabo tkinter, da zapolnite zaslon s trikotniki. Nato spremenite kodo, da zapolnite zaslon s trikotniki različnih barv.

2: Premikajoči se trikotnik

Spremenite kodo za premikanje trikotnika ("Ustvarjanje osnovne animacije"), da se premakne po zaslonu v desno, nato navzdol, nato nazaj na levo in nato nazaj na njegov začetni položaj (da obkroži zaslon).

3: Premikajoča se fotografija

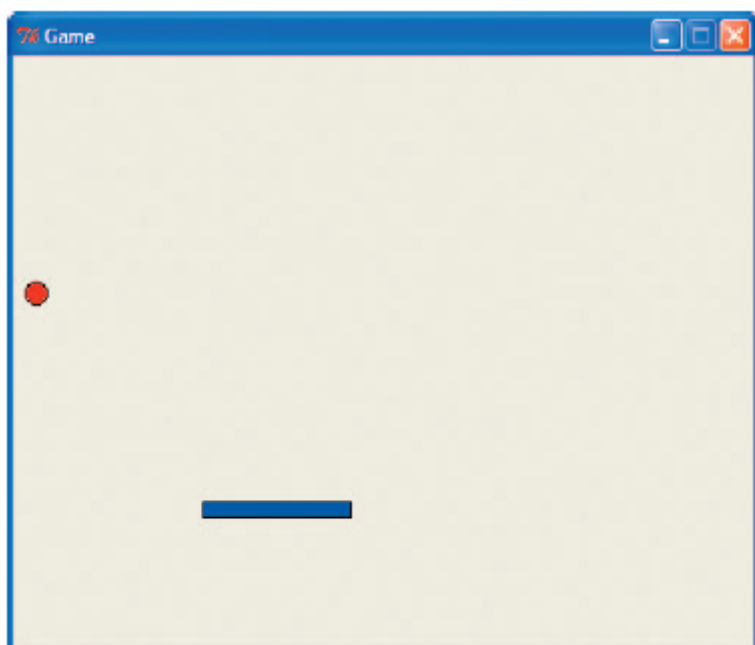
Poskusite prikazati fotografijo na platnu s pomočjo tkinter. Prepričajte se, da je slika PNG ali GIF! Bi lahko sprogramirali, da se premika po zaslonu?

13. poglavje: Začetek vaše prve igre: Odbij!

Doslej smo obdelali osnove računalniškega programiranja. Naučili ste se uporabljati spremenljivke za shranjevanje informacij, pogojne in ponavljalne stavke. Znete ustvariti funkcije za ponovno uporabo kode in uporabljati razrede in objekte, ki zapleteno kodo razdelijo na manjše obvladljive dele. Naučili ste se uporabiti grafiko na zaslonu z moduloma turtle in tkinter. Nastopil je čas, da to znanje uporabite za svojo prvo igro.

Udari žogico

Razvili bomo igro z odbijajočo žogico in loparjem. Žogica bo letela po zaslonu in igralec jo bo odbil z loparjem. Če žogica zadene dno zaslona, se igra konča. Takole naj bi izgledala naša igra:



Igra mogoče izgleda precej preprosta, vendar bo koda nekoliko bolj zapletena od tiste, ki smo jo napisali doslej, ker je veliko stvari za katere moramo poskrbeti. Animirati moramo lopar in žogico ter ugotoviti, kdaj se bo žogica dotaknila loparja ali sten.

V tem poglavju bomo začeli ustvarjati igro z dodajanjem platna in odbijajoče žogice. V naslednjem poglavju bomo zaključili igro z dodajanjem loparja.

Ustvarjanje igralnega platna

Najprej odprite novo datoteko v lupini Python (izberite File, New Window). Potem uvozite tkinter in ustvarite platno, na katerega bomo risali:

```
from tkinter import *
import random
import time
tk = Tk()
tk.title("Igra")
tk.resizable(0, 0)
```

```
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
```

To se nekoliko razlikuje od prejšnjih primerov. Najprej uvozimo modul `time` in `random` z `import time` in `import random` za uporabo v kodi nekoliko kasneje.

S `tk.title('Igra')` uporabimo funkcijo `title` v `tk` objektu (ki smo ga ustvarili s `tk = Tk()`), da bi dobili naslov okna. Potem uporabimo funkcijo `resizable`, da oknu nastavimo stalno velikost. Parametra `0, 0` pomenita »velikosti okna ni mogoče spreminjati niti vodoravno niti navpično.« Nato pokličemo funkcijo `wm_attributes` s parametri, ki okno postavijo pred vsa ostala okna (najvišje = `topmost`).

Pri ustvarjanju objekta `canvas` (platno) s `canvas =`, smo prenesli še nekaj imenovanih parametrov več kot v prejšnjih primerih. Na primer, `bd = 0` in `highlightthickness = 0` zagotovita, da okoli platna ni roba, kar izgleda lepše na zaslonu naše igre.

Vrstica `canvas.pack()` izvrši nastavitve platna glede na širino in višino navedeno v prejšnji vrstici. Nazadnje `tk.update()` pove tkinter naj se nastavi za animacijo v naši igri. Brez te zadnje vrstice, nič ne bi delovalo, kot bi pričakovali.

Sproti shranjujte vašo kodo. Izberite smiselno ime datoteke, ko prvič shranjujete, recimo `odbijanje.py`.

Ustvarjanje razreda Ball

Ustvarili bomo razred za žogo. Začeli bomo s kodo, da se žoga izriše na platnu. Tukaj je, kar potrebujemo narediti:

- Ustvariti razred imenovan `Ball`, ki bo imel parametre za platno in barvo žoge, ki jo bomo pripravili.
- Shraniti platno kot spremenljivko objekta, ker bomo na njem risali žogo.
- Na platno narisati pobarvan krog s parametrom za polnilo.
- Shraniti identifikator, ki ga tkinter vrne, ko nariše oval, ker bomo to uporabili za premikanje žoge po zaslonu.
- Premakniti oval na sredino platna.

To kodo je treba dodati takoj po vrsticah za uvoz modulov (po `import time`):

```
(1) class Ball:
(2)     def __init__(self, canvas, color):
(3)         self.canvas = canvas
(4)         self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
(5)         self.canvas.move(self.id, 245, 100)
        def draw(self):
            pass
```

Najprej definiramo `class Ball` (1). Nato ustvarimo inicializacijsko funkcijo (kot je opisano v poglavju 8), ki prevzame parametra `canvas` in `color` (2). Nastavimo spremenljivko objekta `canvas` na vrednost parametra `canvas` (3).

Kličemo funkcijo `create_oval` s petimi parametri: `x` in `y` koordinatami za zgornji levi kot (10 in 10), `x` in `y` koordinatami za spodnji desni kot (25 in 25), in barvo polnila za oval (4).

Funkcija `create_oval` vrne identifikator, ki ga shranimo v objektno spremenljivko `id`. Pri (5) premaknemo oval na sredino platna (položaj 245, 100).

V zadnjih dveh vrsticah razreda `Ball` ustvarimo funkcijo za izris `draw` (`self`), telo funkcije pa je trenutno še prazno (`pass`). Trenutno nič ne dela. Kmalu bomo dodali tudi to kodo.



Sedaj, ko smo ustvarili naš razred `Ball`, moramo ustvariti objekt tega razreda (ne pozabite, da razred opisuje, kaj lahko naredi, objekt pa je stvar, ki to počne). Dodajte naslednjo kodo na dno programa, da ustvarimo objekt rdeč krog:

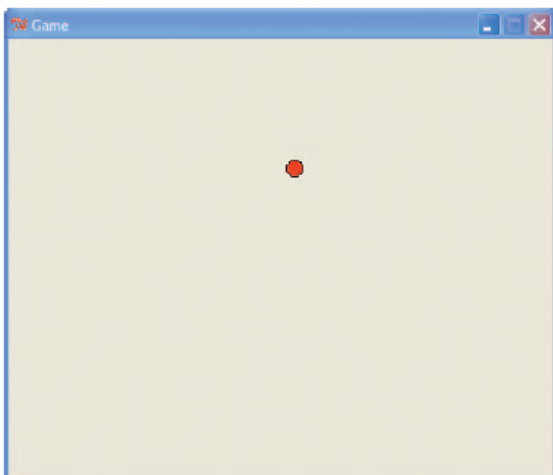
```
ball = Ball(canvas, "red")
```

Ta program lahko sedaj zaženete z uporabo `Run`, `Run Module`. Če to storite izven `IDLE`-a, se bo platno prikazalo za delček sekunde in takoj izginilo. Da se okno ne bi takoj zaprlo, moramo dodati še glavno zanko naše igre. (`IDLE` že ima glavno zanko, zato okno ne izgine.)

Glavna zanka je osrednji del programa. Naša glavna zanka, ta trenutek le pove `tkinter` naj se obnovi in izriše potrebne spremembe. Zanka je neskončna (ali vsaj dokler ne zapremo okna) in stalno obnavlja zaslon, vmes počaka le eno stotinko sekunde. To kodo bomo dodali na konec našega programa:

```
while 1:  
    tk.update_idletasks()  
    tk.update()  
    time.sleep(0.01)
```

Če sedaj poženetete kodo, bi morala biti žoga skoraj v središču platna:



Dodajanje nekaj akcije

Sedaj, ko imamo razred `Ball` nastavljen, je čas, da animiramo žogo. Premaknili jo bomo, odbili in spremenili smer.

Premaknimo žogo

Če želimo premakniti žogo, moramo spremeniti funkcijo draw na naslednji način:

```
def draw(self):
    self.canvas.move(self.id, 0, -1)
```

Ker je `__init__` shranil parameter canvas kot objektno spremenljivko, jo lahko uporabljamo s `self.canvas` in pokličemo funkcijo `move`.

Posredujemo tri parametre: id ovala in številki 0 in -1. 0 pravi, da se vodoravno ne premikamo in -1 pravi naj se premaknemo 1 piko na zaslону.

Majhne spremembe delamo, ker je dobra zamisel preizkušati stvari med potekom. Predstavljajte si, da napišemo celotno kodo naše igre naenkrat in nato odkrijemo, da ne deluje. Kje bi začeli iskati napake?

Druga sprememba je v glavni zanki na dnu našega programa. V bloku `while` zanke (to je naša glavna zanka!) dodamo klic na `draw` funkcijo, takole:

```
while 1:
    ball.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Če sedaj zaženete to kodo, bi se morala žogica premakniti navzgor po platnu in izginiti, ker koda prisili `tkinter`, da hitro obnavlja zaslon - ukaz `update_idletasks` in `update` povesta `tkinter` naj pohiti in izriše tisto, kar je na platnu.

Ukaz `time.sleep` je klic funkcije `sleep` v modulu `time`, ki Pythonu pove naj počaka eno stotinko sekunde (0,01). Da se naš program ne bi izvršil tako hitro, da ne bi ničesar videli.

Zanka v bistvu pravi: malo premakni žogo, ponovno izriši zaslon, počakaj trenutek in potem začni znova.

Opomba: Ko zaprete okno, se lahko prikažejo sporočila o napakah. To je zato, ker ob zapiranju okna prekinemo `while` zanke in Python se pritožuje.

Vaša igra naj bi zdaj izgledala takole:

```
from tkinter import *
import random
import time

tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
```

```

def draw(self):
    self.canvas.move(self.id, 0, -1)

ball = Ball(canvas, 'red')

while 1:
    ball.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)

```

Žogica naj se odbija

Žogica, ki izginja na vrhu zaslona, ni posebej koristna za igro, zato naredimo, da se bo odbijala. Najprej shranimo še nekaj objektnih spremenljivk v inicializacijski funkciji razreda Ball, kot je to:

```

class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        self.x = 0
        self.y = -1
        self.canvas_height = self.canvas.winfo_height()

```

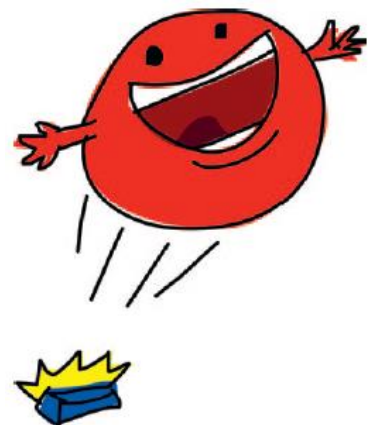
Dodali smo še tri vrstice. S `self.x = 0`, postavimo spremenljivko objekta `x` na 0, nato pa s `self.y = -1`, nastavimo spremenljivko `y` na -1. Spremenljivko objekta `canvas_height` nastavimo s klicem funkcije `platna.winfo_height`. Ta funkcija vrne trenutno višino platna.

Nato ponovno spremenimo `draw` funkcijo:

```

def draw(self):
(1) self.canvas.move(self.id, self.x, self.y)
(2) pos = self.canvas.coords(self.id)
(3) if pos[1] <= 0:
    self.y = 1
(4) if pos[3] >= self.canvas_height:
    self.y = -1

```



Pri (1), spremenimo klic na `move` funkcijo s posredovanjem parametrov objektnih spremenljivk `x` in `y`. Nato ustvarimo spremenljivko `pos` pri (2), tako da pokličemo `coords` funkcijo `platna`. Ta funkcija vrne trenutne koordinate `x` in `y` vsega, kar je narisano na platnu, če le poznate njeno identifikacijsko številko. V tem primeru posredujemo `id` spremenljivko objekta, ki vsebuje identifikator ovala.

Funkcija `coords` vrne koordinate kot seznam štirih števil. Če izpišemo rezultate te funkcije, bomo videli nekaj podobnega:

```

print(self.canvas.coords(self.id))
[255.0, 29.0, 270.0, 44.0]

```

Prvi dve številki na seznamu (255.0 in 29.0) vsebujeta zgornje-leve koordinate ovala (`x1` in `y1`); drugi dve (270.0 in 44.0) sta koordinati `x2` in `y2` spodaj-desno. Te vrednosti uporabimo v naslednjih vrsticah kode.

Pri (3) preverjamo, če je `y1` koordinata (to je vrh žogice!) manjša ali enaka 0. Če je tako, postavimo spremenljivko `y` na 1. Kot bi rekli, če se zadaneš vrha zaslona, ustavi odštevanje 1 od navpičnega položaja, torej nehaj se pomikati navzgor. Pri (4), preverjamo, če je `y2` koordinata (to je pod žogico!)

večja ali enaka spremenljivki `canvas_height`. Če je, nastavimo spremenljivko objekta nazaj na -1. Poženite to kodo in sedaj bi se morala žogica pomikati navzgor in navzdol, dokler ne zaprete okna.

Nastavitev začetne smeri žogice

Premikanje žogice počasi gor in dol še ni ravno igra, zato spremenimo nekaj stvari. Najprej spremenimo začetno smer kroglice. V `__init__`, spremenite ti dve vrstici:

```
self.x = 0
self.y = -1
```

na naslednje (poskrbite, da imate pravo število presledkov – osem na začetku vsake vrstice):

```
(1)     starts = [-3, -2, -1, 1, 2, 3]
(2)     random.shuffle(starts)
(3)     self.x = starts[0]
(4)     self.y = -3
```

Pri (1), ustvarimo spremenljivko `starts` s seznamom šestih števil in nato seznam zmešamo s klicem `random.shuffle` (2). Pri (3) nastavimo vrednost `x` na prvi element seznama, tako da je `x` sedaj poljubna številka iz seznama od -3 do 3.

Če nato spremenimo `y` na -3 pri (4) (da pospešimo žogo), potrebujemo še nekaj popravkov, da nam žogica ne bi ušla ob stranskih stenah zaslona. Dodajte naslednjo vrstico na koncu funkcije `__init__`, da shranite še širino platna v novo objektno spremenljivko `canvas_width`:

```
self.canvas_width = self.canvas.wininfo_width()
```

To novo spremenljivko objekta bomo uporabili v funkciji `draw` in preverili, če se žoga zadene v levo ali desno steno:

```
if pos[0] <= 0:
    self.x = 3
if pos[2] >= self.canvas_width:
    self.x = -3
```

Ker nastavljam `x` na 3 in -3, bomo enako storili z `y`, tako da se žoga premika z enako hitrostjo v vseh smereh. Vaša `draw` funkcija bi morala zdaj izgledati takole:

```
def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if pos[3] >= self.canvas_height:
        self.y = -3
    if pos[0] <= 0:
        self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3
```

Shranite in zaženite kodo in žogica bi se mora odbijati po zaslon. In tukaj je še celoten program:

```
from tkinter import *
import random
import time

tk = Tk()tk.title("Game")
```

```

tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()

```

```

class Ball:

```

```

    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()

```

```

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.canvas_height:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.canvas_width:
            self.x = -3

```

```

ball = Ball(canvas, 'red')

```

```

while 1:
    ball.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)

```

Kaj ste se naučili

V tem poglavju smo začeli ustvariti svojo prvo igro z uporabo modula tkinter. Ustvarili smo razred za žogo in jo animirali, da se je premikala po zaslonu. Uporabili smo koordinate za preverjanje dotika stranic in odboj žogice od stranic platna. Uporabili smo funkcijo shuffle iz modula random za določitev začetne smeri gibanja žogice. V naslednjem poglavju bomo igro dopolnili z dodajanjem loparja.

14. poglavje: Dokončanje vaše prve igre: Odbij!

V prejšnjem poglavju smo začeli ustvarjati našo prvo igro: Odbij! Ustvarili smo platno in dodali odbijajočo se žogico. Toda naša žoga se bo večno odbijala po zaslonu (ali vsaj dokler ne izklopite računalnika), kar ni ravno zanimiva igra. Dodali bomo lopar za igralca. Dodali bomo tudi element naključja, kar bo igro naredilo zanimivejšo in zabavnejšo.

Dodajanje loparja

Z žogo, ki se odbija ni veliko veselja, če je ne moreš s čim udariti. Čas je, da ustvarimo lopar!

Začnite z dodajanjem kode takoj za class Ball, da ustvarite razred Paddle (začnite z novo vrstico pod draw funkcijo razreda Ball):

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)

    def draw(self):
        pass
```

Ta dodana koda je skoraj enaka kot za razred Ball, razen da pokličemo create_rectangle (ne pa create_oval) in da pravokotnik premaknemo v položaj 200, 300 (200 pik desno in 300 pik dol).

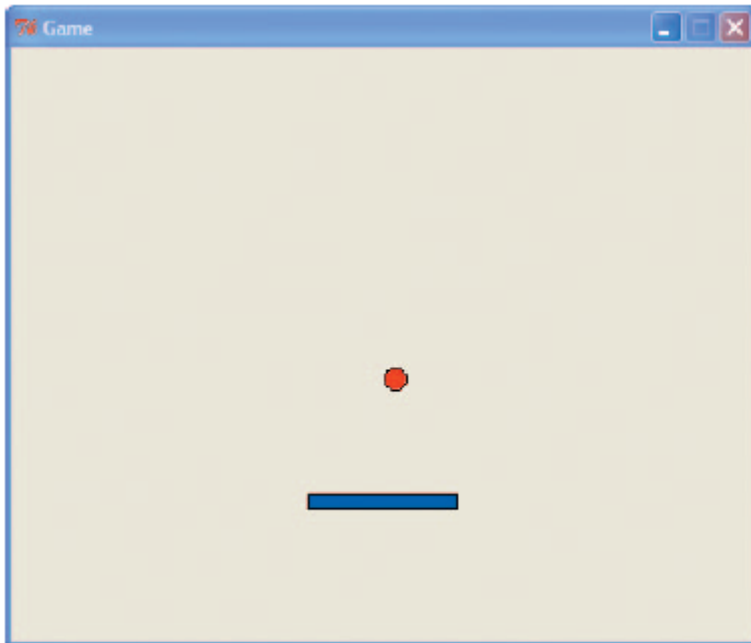
Nato na dnu vnosa kode ustvarite objekt razreda Paddle in spremenite glavno zanko, da pokličete draw funkcijo, kot je prikazano tukaj:

```
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, 'red')

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Če poženetе igro zdaj, bi morali videti žogico in pravokoten lopar:





Premikanje loparja

Da bi se lopar premaknil levo in desno, bomo povezali funkcije z dogodki leve in desne smerne tipke v razredu Paddle. Ko igralec pritisne puščico levo, bo spremenljivka x nastavljena na -2 (premik levo). S pritiskom desne smerne tipka x nastavimo na 2 (za premikanje desno).

Prvi korak je dodati x objektne spremenljivke v funkcijo `__init__` našega razreda Paddle in tudi spremenljivko za širino platna, kot smo naredili z razredom Ball:

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
```

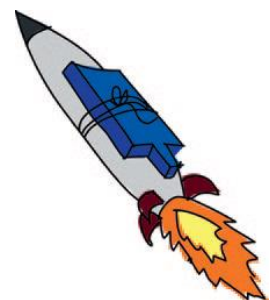
Zdaj potrebujemo funkcije za spreminjanje smeri med levo (`turn_left`) in desno (`turn_right`). Te bomo dodali takoj za `draw` funkcijo:

```
def turn_left(self, evt):
    self.x = -2

def turn_right(self, evt):
    self.x = 2
```

Ti dve funkciji lahko povežemo z ustreznimi tipkami v `__init__` funkciji s tema dvema vrsticama. Povezovanje smo uporabili v poglavju "Odziv objekta na dogodek", ko je Python klical funkcijo kot odziv na pritisnjeno tipko. V tem primeru povežemo funkcijo `turn_left` našega razreda Paddle na levo smerno tipko z uporabo dogodka '`<KeyPress-Left>`'. Nato povežemo funkcijo `turn_right` na desno smerno tipko z dogodkom '`<KeyPress-Right>`'. Naša funkcija `__init__` zdaj izgleda takole:

```
def __init__(self, canvas, color):
    self.canvas = canvas
```



```

self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
self.canvas.move(self.id, 200, 300)
self.x = 0
self.canvas_width = self.canvas.winfo_width()
self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
self.canvas.bind_all('<KeyPress-Right>', self.turn_right)

```

Funkcija draw za razred Paddle je podobna tisti za razred Ball:

```

def draw(self):
    self.canvas.move(self.id, self.x, 0)
    pos = self.canvas.coords(self.id)
    if pos[0] <= 0:
        self.x = 0
    elif pos[2] >= self.canvas_width:
        self.x = 0

```

Za premikanje loparja uporabljamo funkcijo move z ukazom self.canvas.move (self.id, self.x, 0). Nato s spremenljivko pos pogledamo koordinate loparja, da vidimo, če je zadel levo ali desno stran zaslona.

Namesto, da bi se odbil kot žoga, se lopar ustavi. Torej, ko je leva x koordinata (pos [0]) manjša ali enaka 0 (<= 0), x nastavimo na 0 s self.x = 0. Na enak način, ko je desna x koordinata (pos [2]) večja ali enaka širini platna (>= self.canvas_width), nastavimo x na 0 s self.x = 0.

Opomba: Če zdaj zaženete program, morate pred tem klikniti na okno, da bo reagiralo na tipke. Klik na okno nastavi aktivno okno.

Kdaj žoga zadene lopar

Na tej točki našega programa, žoga ne bo zadela loparja; v bistvu bo žoga letel naravnost skozi lopar. Žoga mora vedeti, kdaj udari v lopar, podobno kot mora vedeti, kdaj se zadane stene.

Te težave bi lahko rešili z dodajanjem kode v draw funkcijo (kjer imamo kodo, ki preverja stene), vendar je boljše, da premaknemo to vrsto kode v nove funkcije, da kodo razbijemo na manjše dele. Če na enem mestu postavimo preveč kode (na primer znotraj ene funkcije), je koda precej težje razumljiva. Naredimo potrebne spremembe.

Najprej spremenimo funkcijo __init__ v razredu Ball, da bomo lahko kot parameter prenesli objekt paddle:

```

class Ball:
(1) def __init__(self, canvas, paddle, color):
    self.canvas = canvas
(2)     self.paddle = paddle
    self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
    self.canvas.move(self.id, 245, 100)
    starts = [-3, -2, -1, 1, 2, 3]
    random.shuffle(starts)
    self.x = starts[0]
    self.y = -3
    self.canvas_height = self.canvas.winfo_height()
    self.canvas_width = self.canvas.winfo_width()

```

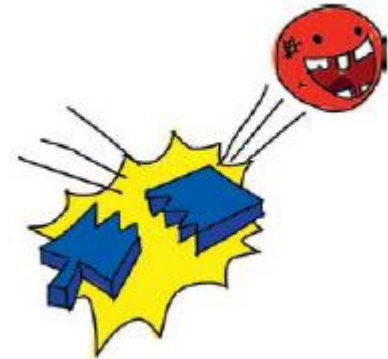
Pri (1) smo v __init__ dodali parameter paddle. Nato smo pri (2) dodelili parameter paddle objektni spremenljivki.

Ko smo rešili problem loparja, moramo spremeniti še kodo, kjer ustvarimo objekt ball. Ta sprememba je na dnu program, tik pred glavno zanko:

```
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, 'red')
```

```
while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Koda, ki jo potrebujemo, da vidimo, če je žogica zadela lopar, je nekoliko bolj zapletena kot koda za preverjanje zidov. To funkcijo bomo imenovali `hit_paddle` in jo dodali v `draw` funkcijo razreda `Ball`, kjer preverjamo, če je žogica zadela dno zaslona:



```
def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if pos[3] >= self.canvas_height:
        self.y = -3
    if self.hit_paddle(pos) == True:
        self.y = -3
    if pos[0] <= 0:
        self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3
```

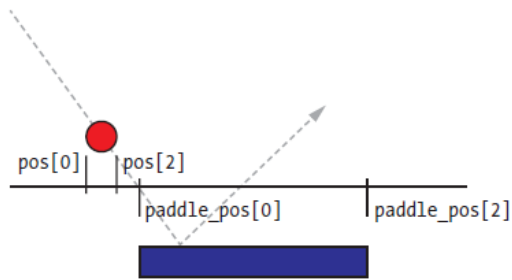
Kot lahko vidite v novo dodani kodi, če je `hit_paddle` `True`, spremenimo smer žoge tako, da nastavimo `y` na `-3` s `self.y = -3`. Vendar ne poskušajte zagnati igre že sedaj, saj funkcije `hit_paddle` še nimamo. Naredimo jo.

Dodajte funkcijo `hit_paddle` tik pred `draw` funkcijo.

```
(1) def hit_paddle(self, pos):
(2)     paddle_pos = self.canvas.coords(self.paddle.id)
(3)     if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
(4)         if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
            return True
        return False
```

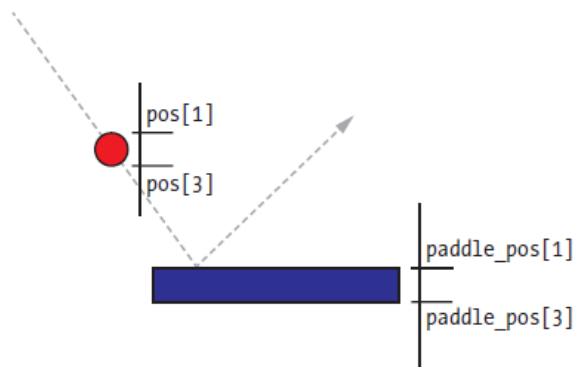
Najprej definiramo funkcijo s parametrom `pos` pri (1). Ta vrstica vsebuje trenutne koordinate žogice. Nato pri (2) dobimo koordinate loparja in jih shranimo v spremenljivko `paddle_pos`.

Pri (3) imamo prvi del našega `if`-stavka in rečemo: "Če je desna stran žogice večja od leve strani loparja in leva stran žogice manjša od desne strani loparja..." Tukaj, `pos[2]` vsebuje `x` koordinato za desna stran žogice in `pos[0]` vsebuje `x` koordinato za levo stran. Spremenljivka `paddle_pos[0]` vsebuje `x` koordinato za levo stran loparja in `paddle_pos[2]` vsebuje `x` koordinato za desno stran. Naslednji diagram prikazuje, kako izgledajo te koordinate preden žogica zadane lopar.



Žogica se spušča proti loparju, vendar v tem primeru vidite, da desna stran žogice (`pos [2]`) še ni prečkala leve strani loparja (to je `paddle_pos [0]`).

Pri (4) gledamo, če je dno kroglice (`pos [3]`) med vrhom loparja (`paddle_pos [1]`) in dnom loparja (`paddle_pos [3]`). V naslednjem diagramu, lahko vidite, da dno žogice (`pos [3]`) še ni udarilo na vrh loparja (`paddle_pos [1]`).



Torej bi glede na trenutni položaj žoge, `hit_paddle` funkcija vrnila `False`.

Opomba: Zakaj moramo videti, če je spodnji del žoge med vrhom in dnom loparja? Zakaj ne pogledamo le, če je spodnji del žogice zadel vrh loparja? Ker vsakič premikamo žogo preko platna s točkovnimi premiki. Če smo ravno preverili, če je žogica prišla do vrha loparja (`pos [1]`), smo morda zamudili ta položaj. V tem primeru bo žogica nadaljevala pot, lopar je ne bi ustavil.

Dodajanje elementa naključnosti

Zdaj je čas, da spremenimo naš program v igro, kjer ne bo samo odbijanje žogice in lopar. Igra potrebuje tudi element naključja, da igralec lahko tudi izgubi. V naši trenutni igri se bo žogica vedno odbijala, zato je ne moremo izgubiti.

Igro bomo zaključili z dodajanjem kode, ki bo igro končala, če žoga zadane spodnji rob zaslona.

Najprej dodamo spremenljivko `hit_bottom` na konec funkcije `__init__` v razredu `Ball`:

```
self.canvas_height = self.canvas.wininfo_height()
self.canvas_width = self.canvas.wininfo_width()
self.hit_bottom = False
```

Nato spremenimo glavno zanko na dnu programa, kot je tule:

```
while 1:
    if ball.hit_bottom == False:
        ball.draw()
```

```

    paddle.draw()
tk.update_idletasks()
tk.update()
time.sleep(0.01)

```

Zdaj zanka spremlja `hit_bottom`, če se žogica dejansko dotakne dna zaslona. Koda premika žogico in lopar le, če se žoga ni dotaknila dna. Igra se konča, ko se žogica in lopar nehata premikati. (ju ne animiramo več.) Končna sprememba je funkcija `draw` v razredu `Ball`:

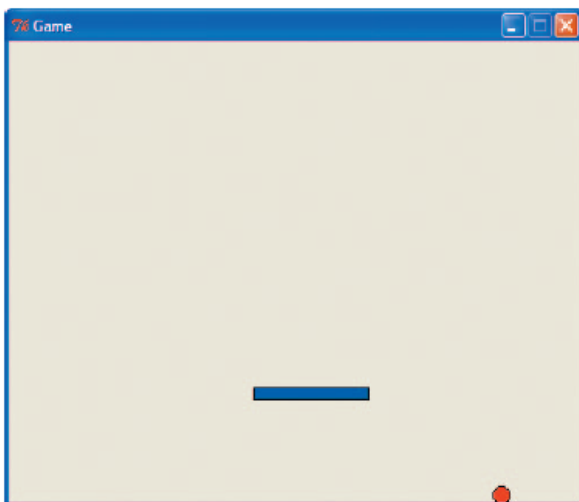
```

def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if self.hit_paddle(pos) == True:
        self.y = -3
    if pos[3] >= self.canvas_height:
        self.hit_bottom = True
    if pos[0] <= 0:
        self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3

```

Spremenili smo `if` stavek, da preverimo, če se žogica dotakne dna zaslona (to je, če je večja ali enaka `canvas_height`). Če je tako, v naslednji vrstici postavimo `hit_bottom` na `True`, namesto na spreminjanje vrednosti spremenljivke `y`, ker ni potrebe po odbijanju žogice, ko se je dotaknila dna zaslona.

Ko zdaj igrate in zgrešite žogico, se gibanje na zaslonu ustavi in igra se konča, ko se žoga dotakne dna platna:



Vaš program bi zdaj moral izgledati kot spodnja koda. Če imate težave pri zagonu igre, preverite, kaj ste vnesli narobe.

```

from tkinter import *
import random
import time

tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)

```

```

canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()

```

```

class Ball:

```

```

    def __init__(self, canvas, paddle, color):
        self.canvas = canvas
        self.paddle = paddle
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if self.hit_paddle(pos) == True:
            self.y = -3
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.canvas_width:
            self.x = -3

    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                return True
        return False

```

```

class Paddle:

```

```

    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)

    def turn_left(self, evt):
        self.x = -2

    def turn_right(self, evt):
        self.x = 2

    def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.canvas_width:

```

```

        self.x = 0

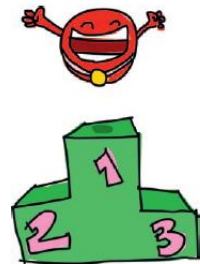
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, 'red')

while 1:
    if ball.hit_bottom == False:
        ball.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)

```

Kaj ste se naučili

V tem poglavju smo končali z ustvarjanjem naše prve igre, ki uporablja modul tkinter. Ustvarili smo razrede za lopar, ki ga uporabljamo v igri in uporabili koordinate za preverjanje, ko žoga zadane lopar ali stene našega platna. Uporabili smo krmiljenje s smernimi tipkami za gibanje loparja in uporabili glavno zanko za izrisovanje animacije. Nazadnje smo popravili kodo tako, da smo igri dodali element naključja in v primeru zgrešene žogice zaključili igro, ko žogica prileti do dna platna.



Vaje

Trenutno je naša igra nekoliko preprosta. Veliko stvari lahko spremenite, da ustvarite bolj profesionalno igro. Poskusite izboljšati svojo kodo na naslednje načine, da postane bolj zanimiva in nato preverite odgovore na <https://nostarch.com/pythonforkids>

1: Zakasnitev začetka igre

Naša igra se malo hitro začne in za prepoznavanje ukazov tipk morate najprej klikniti platno. Ali lahko dodate zakasnitev začetka igre, da igralec v miru klikne platno in začne igro? Ali celo bolje, lahko dodate dogodek, ki je povezan s klikom z miško in šele nato začnete igrati?

Namig 1: Povezave na dogodke ste že dodali v razredu Paddle, tako da je to lahko dober kraj za začetek.

Namig 2: Dogodek, ki poveže levi gumb miške, je niz '<Button-1>'.

2: Pravilen "Konec igre"

Ko se igra konča, le vse neha delovati in to ni prijazno do igralca. Poskusite dodati besedilo »Konec igre«, ko žogica zadane dno zaslona. Lahko uporabite funkcijo create_text, verjetno pa boste morali uporabiti tudi parameter state (ta je lahko normal ali hidden). Oglejte si itemconfig v »Več načinov uporabe identifikatorja«. Kot dodaten izziv, dodajte zakasnitev, tako da se besedilo ne prikaže takoj.

3: Pospešite žogico

Če igrate tenis, veste, da ko žoga zadene lopar včasih odleti hitreje kot prileti, odvisno od moči udarca. Žogica v naši igri leti s stalno hitrostjo, ne glede na to, ali se veslo premika ali ne. Poskusite spremeniti program, tako da se hitrost loparja prenese na hitrost žogice.

4: Zapišite rezultat igralca

Kaj pa shranjevanje rezultata? Vsakič, ko žogica zadene lopar, naj se rezultat poveča. Poskusite prikazati rezultat v zgornjem desnem kotu platna. Morda boste želeli pogledati nazaj na itemconfig in "Več načinov za uporabo identifikatorja".

15. poglavje: Izdelava grafike za igrico Mr. Stickman

Pri ustvarjanju igre (ali katerega koli programa) je dobro najprej narediti načrt. Vaš načrt mora vsebovati opis tega; zgodbo s ciljem in opis glavnih elementov in likov v igri. Ko boste programirali, vam bo opis pomagal, da boste skoncentrirani na to, kar poskušate narediti. Vaša igra morda ne bo povsem taka, kot je v opisu, kar pa je povsem v redu.

V tem poglavju bomo začeli razvijati zabavno igro imenovan g. Stickman drvi proti izhodu.

Načrt igre Mr. Stickman

Tukaj je opis naše nove igre:

- Skrivni agent Mr. Stickman je ujet v skrivališču dr. Nežnega. Želite mu pomagati, da pobegne skozi izhod na zgornjem nadstropju.
- Igra ima lik narisana s črtami, ki lahko teče na levo ali desno in skoči. V vsakem nadstropju so platforme, na katere mora skočiti.
- Cilj igre je doseči vrata za izhod, preden je prepozno in se igra konča.

Na podlagi tega opisa vemo, da bomo potrebovali več slik, vključno s tistimi za Mr. Stickmana, platforme in vrata. Očitno bomo potrebovali kodo, da vse skupaj povežemo, vendar preden se lotimo programiranja, bomo v tem poglavju ustvarili grafiko za našo igro. Na ta način bomo imeli nekaj, s čimer bomo lahko delali v naslednjem poglavju.

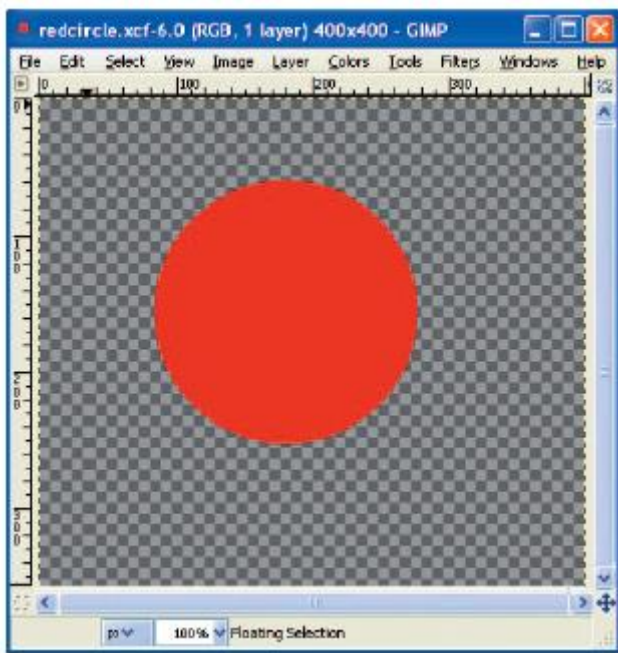


Kako bomo narisali elemente v naši igri? Lahko bi uporabimo grafiko, podobno kot za odbijanje žogice v prejšnjih poglavjih, vendar bi bilo to čisto preveč preproste za to igro. Namesto tega bomo ustvarili sličice.

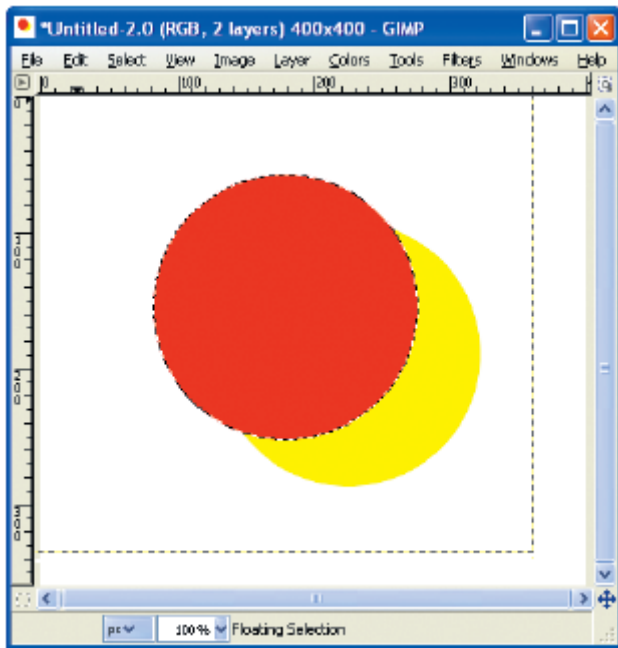
Sličice so stvari v igri, običajno nastopajoči. Sličice so ponavadi narisane v naprej in jih ne izrisujemo v programu z uporabo poligonov, kot v primeru igre Odbij!. Mr. Stickman bo sličica, prav tako pa tudi platforme in vrata. Če želite ustvariti te slike, morate imeti grafični program.

Namestimo GIMP

Na voljo je več grafičnih programov, vendar za to igro potrebujemo takega, ki podpira prosojnost (včasih se imenuje alfa kanal), ki omogoča, da slike vsebujejo razdelke, kjer ni barv. Potrebujemo slike s prozornimi deli, ker pri prehajanju ene slike preko druge ne želimo, da ozadje ene slike prekrije del druge slike. Na tej sliki, vzorec šahovnice v ozadju predstavlja prosojno območje:



Torej, če kopiramo celotno sliko in jo prilepimo na vrhu druge slike, ozadje ne bo smelo prekriti ničesar:



GIMP (<http://www.gimp.org/>), kratko za GNU Manipulation Image Program (Odprtokodni program za obdelavo slik), je brezplačen grafični program za različne operacijske sisteme, ki podpira prosojne slike. Prenesete in namestite ga na naslednji način:

Namestitve so na strani projekta GIMP na <https://www.gimp.org/downloads/>.

Prav tako ustvarite mapo za svojo igro. Narediti tako, z desno miškino tipko kliknite svoje namizje kjerkoli je prazen prostor in izberite Nova, Mapa. V pogovornem oknu vnesite stickman za ime mape.

Ustvarjanje elementov igre

Ko imate nameščen grafični program, ste pripravljeni za risanje. Naši elementi igre so:

- slike za črtasto figuro, ki lahko teče levo in desno in skoči
- slike za platformo, v treh različnih velikostih
- slike za vrata: ena odprta in ena zaprta
- slika za ozadje igre (ker je navadno belo ali sivo ozadje dolgočasno za igro)

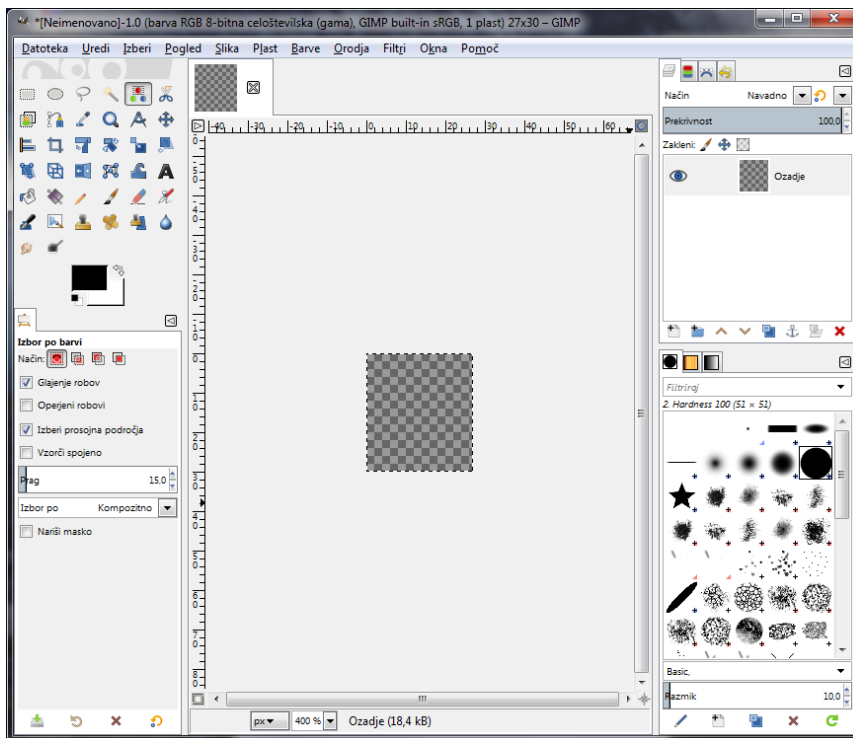
Preden začnemo risati, moramo za svoje slike pripraviti prosojno ozadje.

Priprava prosojnega ozadja slike

Če želite nastaviti sliko s prosojnostjo - alpha kanal – poženite GIMP in sledite tem korakom:

1. Izberite Datoteka, Nova.
2. V pogovornem oknu vnesite 27 slikovnih pik za širino slike in 30 slikovnih pik za višino.
3. Izberite Plast, Prosojnost, Dodaj kanal alfa.
4. Izberite Izberi, Vse.
5. Izberite Uredi, Izreži.

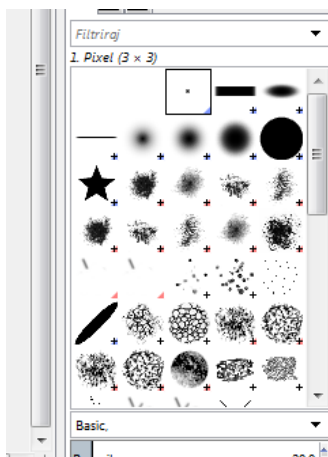
Končni rezultat mora biti slika, napolnjena s šahovnico temno sive in svetlo sive, kot je prikazano tukaj (povečano na 400 %):



Sedaj lahko začnemo ustvarjati našega tajnega agenta: Mr. Stickmana.

Risba Mr. Stickmana

Za prvo sliko v orodjih kliknite na svinčnik, izberite med oblikami piko (običajno na spodnji desni strani zaslona), kot prikazano na sliki.



Naredili bomo tri različne slike (ali okvirje) za prikaz figurice iz črt, ko teče in skoči na desno. Te okvire bomo uporabili za animacijo Mr. Stickmana, kot smo naredili z animacijo v poglavju 12.

Če povečate pogled teh slike, bodo morda videti takole:



Ni treba, da so vaše slike točno take, vendar naj bodo s tremi različnimi položaji gibanja. Ne pozabite, da je vsaka široka 27 pik in visoka 30 pik.

Mr. Stickman, ki teče desno

Najprej bomo naredili zaporedje okvirjev za Mr. Stickman, ki teče na desno. Ustvarite prvo sliko na naslednji način:

1. Narišite prvo sliko (levo sliko v zgornji ilustraciji).
2. Izberite Datoteka, Izvozi kot.
3. V pogovornem oknu vnesite stickman-r1. Če zraven ni končnice .png, kliknite na gumb + plus (+) z oznako Izberi vrsto datoteke (Po končnico) (spodaj, nad gumbom Pomoč).
4. V seznamu izberite PNG.
5. Shranite datoteko v delovno mapo, ki ste jo ustvarili prej (kliknite Brskajte po drugih mapah in poiščite pravilno mapo).

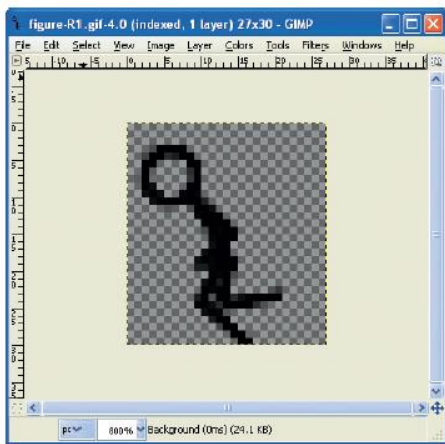


Iste korake uporabljajte za nova slike 27 pik širine in 30 pik višine in nato pripravi naslednjega Mr. Stickmana. Drugo sliko shranite kot stickman-R2.png in tretjo kot stickman-R3.png.

Mr. Stickman teče proti levi

Namesto ponovnega oblikovanja risb za premikanje Mr. Stickmana na levo, lahko uporabimo GIMP, da obrne obstoječe okvirje.

Če nimate še odprtih slik, odprite vsako sliko v zaporedju in izberite Orodja, Orodja preoblikovanja, Prezrcali. Ko kliknete sliko, bi morali videti preklap iz ene strani v drugo. Shranite slike kot stickman-L1.png, stickman-L2.png in stickman-L3.png.



Sedaj imamo šest slik za Mr. Stickmana, vendar še vedno potrebujejo slike za platforme in izhodna vrata.

Risanje platform

Ustvarili bomo tri platforme v različnih velikostih: 100 pik širine in 10 pik višine, 66 pik širine in 10 pik višine in 32 pik širine in 10 pik višine. Lahko jih pripravite kakor hočete, vendar naj bo ozadje prosojno, tako kot pri slikah Mr. Stickmana.

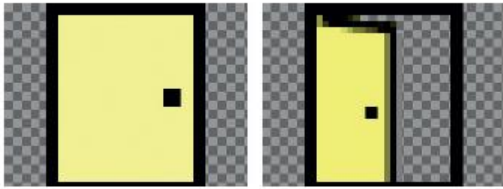
Takole naj bi izgledale tri slike platforme (povečano):



Tako kot prejšnje slike, shranite te slike v svojo delovno mapo. Poimenujte največjo platformo platform1.png, srednje veliko platform2.png in najmanjšo platform3.png.

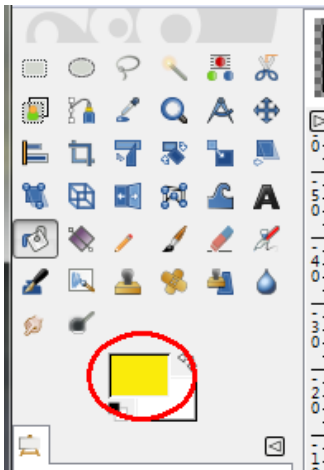
Risanje vrat

Velikost vrat mora biti sorazmerna velikosti Mr. Stickmana (širok 27 pik visok 30 pik) in potrebujemo dve sliki: eno za zaprta vrata in drugo za odprta vrata. Vrata bi lahko izgledala takole (znova povečano):



Sliki lahko ustvariti takole:

1. Izberite orodje za svinčnik ali pisalo in narišite črni okvir vrat in ključavnico.
2. Kliknite polje za barvo ospredja (na dnu GIMP Toolbox) za prikaz izbirnika barv. Izberite barvo, ki jo želite za vaša vrata. Spodaj je primer z izbrano rumeno barvo.



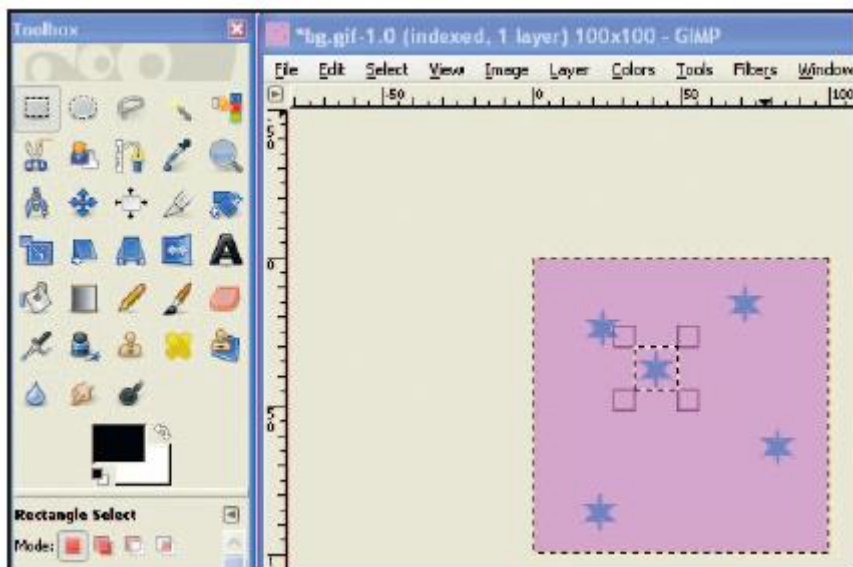
3. Izberite orodje Vedro (prikazano je izbrano v orodjarni) in zapolnite prostor z barvo, ki ste jo izbrali.
4. Izvozite sliki kot door1.png in door2.png.

Risanje ozadja

Zadnja slika, ki jo moramo ustvariti, je ozadje. To sliko bomo naredili 100 pik široko in 100 pik visoko. Ne potrebujemo prosojnega ozadja, ker bo to za ozadje, za vsemi drugimi elementi igre.

Če želite ustvariti ozadje, izberite Datoteka, Nova in podajte velikost slike: širina 100 in višina 100. Izberite ustrezno barvo za ozadje. Izbral sem temnejši roza odtenek.

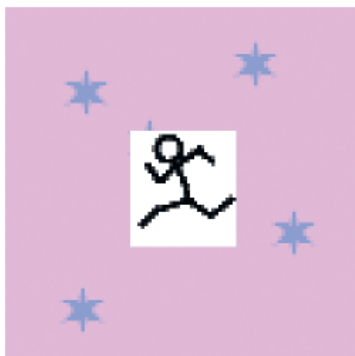
Tapeto lahko opremite s cvetjem, črtami, zvezdami, in tako - kakor mislite, da bi bilo primerno za igro. Če želite dodati v ozadje zvezdice, izberite drugo barvo, izberite orodje Svinčnik in pripravite svojo prvo zvezdo. Nato uporabite Orodje za izbor (čarobna palica ali pravokotni izbor), kliknite v zvezdico in kopirajte in prilepite po sliki (izberite Uredi, Kopiraj in nato Uredi, Prilepi). Prilepljeno sliko se da povleči po zaslonu s klikom. Tukaj je primer z nekaj zvezdicami:



Ko ste zadovoljni s svojo risbo, izvozite sliko kot background.png v svojo mapo.

Prosojnost

S pripravljenimi slikami, lahko sedaj vidimo razliko med prosojnimi in neprosojnimi slikami. Kaj bi se zgodilo, če bi bil Mr. Stickman na belem ozadju? Tukaj je odgovor:



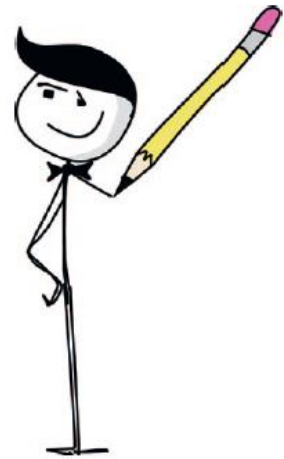
Belo ozadje Mr. Stickmana prekrije del tapete. Če pa uporabimo našo prosojno sliko, dobimo to:



Nič ozadja ni prekrita z ozadjem sličice. To je precej bolj profesionalno!

Kaj ste se naučili

V tem poglavju ste se naučili, kako napisati osnovni načrt za igro (v tem primeru Mr. Stickman drvi proti izhodu) in ugotovili, kje začeti. Ker za igro potrebujemo grafične elemente, jih ustvarimo s programom za urejanje slik. V tem procesu ste se naučili narediti ozadje teh slik prosojno, da ne prekrivajo drugih slik na zaslonu. V naslednjem poglavju bomo ustvarili nekaj razredov za našo igro.



16. poglavje: Razvoj igre Mr. Stickman

Ko smo ustvarili slike za našega Mr. Stickmana, lahko začnemo razvijati kodo. Opis igre v prejšnjem poglavju nam daje osnovno idejo o tem, kaj potrebujemo: glavno figuro iz črt, ki lahko teče in skače, ter platforme, na katere mora skočiti.

Potrebovali bomo kodo za prikaz in premikanje glavne sličice po zaslonu, kot tudi izris platform. Toda preden napišemo to kodo, moramo ustvariti platno za prikaz naše slike ozadja.

Izdelava razreda Game

Najprej bomo ustvarili razred, imenovan Game, ki bo naš glavni nadzornik programa. Razred Game bo imel funkcijo `__init__` za inicializacijo igre in funkcijo `mainloop` za izvedbo animacije.

Nastavitev naslova okna in izdelava platna

V prvem delu funkcije `__init__` bomo nastavili naslov okna in ustvarili platno. Kot boste videli, je ta del kode podoben kodi, ki smo jo napisali za igro Odbij! v poglavju 13. Odprite urejevalnik in vnesite naslednjo kodo, nato pa shranite datoteko kot `stickmangame.py`. Poskrbite, da jo shranite v imenik, ki ga imamo ustvarjenega v poglavju 15 (imenovan `stickman`).

```
from tkinter import *
import random
import time

class Game:
    def __init__(self):
        self.tk = Tk()
        self.tk.title("Mr. Stickman drvi proti izhodu")
        self.tk.resizable(0, 0)
        self.tk.wm_attributes("-topmost", 1)
        self.canvas = Canvas(self.tk, width=500, height=500, \
                             highlightthickness=0)
        self.canvas.pack()
        self.tk.update()
        self.canvas_height = 500
        self.canvas_width = 500
```

V prvi polovici tega programa (od `from tkinter import *` do `self.tk.wm_attributes`), ustvarimo `tk` objekt in nato nastavimo naslov okna s `self.tk.title` na ("Mr. Stickman drvi proti izhodu"). Okno smo fiksirali (zato ni mogoče spremeniti njegove velikosti) s klicem funkcije `resizable` in nato premaknemo okno pred vsa druga okna s funkcijo `wm_attributes`.

Nato ustvarimo platno v vrstici `self.canvas = Canvas`, in pokličemo `pack` in `update` funkciji za posodobitev `tk`-objekta. Nazadnje ustvarimo dve spremenljivki `canvas_height` in `canvas_width` - višino in širino platna.

Opomba: Backslash ali obrnjena poševnica (\) v vrstici `self.canvas = Canvas` je samo zato, da predolgo vrstico razdeli v dve vrstici. Ni potrebna, vendar je tu vnesena zaradi boljše čitljivosti, saj bi bila vrstica predolga.

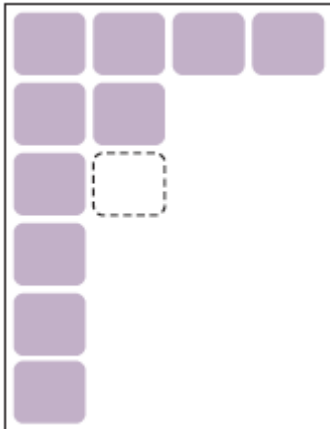
Končanje funkcije `_init_`

Zdaj vnesite preostanek funkcije `__init__` v `stickmangame.py` datoteko, ki ste jo pravkar ustvarili. Ta koda bo naložila sliko ozadja in jo nato prikazala na platnu:

```
self.tk.update()
self.canvas_height = 500
self.canvas_width = 500
(1) self.bg = PhotoImage(file="background.png")
(2) w = self.bg.width()
    h = self.bg.height()
(3) for x in range(0, 5):
(4)     for y in range(0, 5):
(5)         self.canvas.create_image(x * w, y * h, \
            image=self.bg, anchor='nw')
(6) self.sprites = []
    self.running = True
```

Pri (1) ustvarimo spremenljivko `bg`, ki vsebuje `PhotoImage` objekt - slikovna datoteka ozadja, imenovana `background.png`, ustvarjena v 15. poglavju. Nato z začetkom v (2) shranimo širino in višino slike v spremenljivkah `w` in `h`. Funkciji `width` in `height` razreda `PhotoImage` vrmeta velikost naložene slike.

V nadaljevanju prideta dve zanki. Da bi razumeli, kaj delata, si predstavljajte da imate majhen kvadratni gumijasti žig, blazinico za črnilo in velik kos papirja. Kako boste zapolnili papir s kvadratnim žigom? No, lahko bi samo naključno pritiskali žig dokler list ne bi bil zapolnjen. Rezultat bi bil nered in nekaj časa bi trajalo, vendar bi bila stran vseeno zapolnjena. Lahko pa bi začeli žigosanje navzdol po strani v stolpcu in se nato pomaknili nazaj na vrh in ponovno žigosali navzdol v naslednjem stolpcu, kot prikazano na sliki.



Ozadje, ki smo ga ustvarili v prejšnjem poglavju je naš žig. Vemo, da je platno široko 500 in visoko 500 pik in da smo ustvarili ozadje slike s 100 pikami. To nam pove, da potrebujemo pet stolpcev in pet vrstic za zapolnitev zaslona s slikami. Zanko (3) uporabljamo za izračun preko stolpcev in zanko (4) za izračun po vrsticah navzdol.

Pri (5) množimo prvo spremenljivko zanke `x` s širino slike (`x * w`), da določimo položaj proti desni in potem množimo v drugi zanki spremenljivko `y` z višino slike (`y * h`), da določimo, kako daleč navzdol

je treba narisati. Za izris slike na zaslon s pomočjo teh koordinat uporabljamo funkcijo `create_image` objekta `canvas` (`self.canvas.create_image`).

Nazadnje, z začetkom v (6) ustvarimo spremenljivko `sprite`, ki ima prazen seznam in `running`, ki vsebuje Boolovo vrednost `True`. Ti spremenljivki bomo uporabili kasneje v kodi naše igre.

Izdelava glavne zanke - funkcija `mainloop`

Za animacijo igre bomo uporabili funkcijo `mainloop` v razredu `Game`. Ta funkcija je zelo podobna glavni zanki (ali zanki `animation`), ki smo jo ustvarili za igro `Odbij!` v poglavju 13. Tukaj je:

```
        for x in range(0, 5):
            for y in range(0, 5):
                self.canvas.create_image(x * w, y * h, image=self.bg,
anchor='nw')
        self.sprites = []
        self.running = True

    def mainloop(self):
(1)     while 1:
(2)         if self.running == True:
(3)             for sprite in self.sprites:
(4)                 sprite.move()
(5)                 self.tk.update_idletasks()
                self.tk.update()
                time.sleep(0.01)
```



Pri (1) ustvarimo `while` zanko, ki bo trajala do zaprtja okna. Nato, pri (2), primerjamo spremenljivko `running` s `True`. V kolikor je, se sprehodimo s `for` zanko (3) po seznamu sličic (`self.sprites`) in jih s funkcijo `move` premaknemo pri (4). (Seveda moramo še ustvariti sličice, torej ta koda še ne bi nič naredila, če bi program pognali sedaj.)

Zadnje tri vrstice pri (5) ukažejo `tk` objektu, da se ponovno izriše in da počaka eno stotinko, kot smo naredili v igri `Odbij!` v poglavju 13.

Za zagon te kode, dodajte še spodnji vrstici (upoštevajte, da ni nobenega zamika) in shranite datoteko.

```
g = Igra()
g.mainloop()
```

OPOMBA: Prosim, da to kodo dodate na dnu. Prav tako se prepričajte, da so vaše slike v isti mapi kot datoteka igre. Če ste v poglavju 15 ustvarili imenik `stickman` in shranili vse vaše slike tam, bi morala biti datoteka za to igro tudi tam.

Ta koda ustvari objekt razreda `Game` in ga shrani kot spremenljivko `g`. Nato kličemo glavno zanko – funkcijo `mainloop` v novem objektu za izris zaslona.

Ko ste program shranili, ga v `IDLE` izvedite tako, da izberete `Run`, `Run Modul`. V ozadju se prikaže platno zapolnjeno s sliko ozadja.

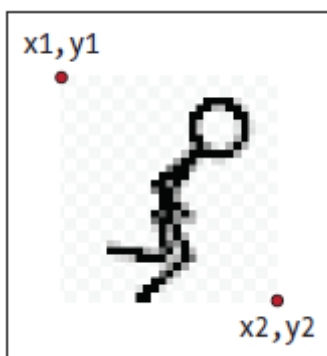


Dodali smo prijetno ozadje za našo igro in ustvarili animacijsko zanko, ki nam bo izrisovala sličice (ko jih bomo ustvarili).

Izdelava razreda Coords

Zdaj bomo ustvarili razred, ki ga bomo uporabili za določitev položaja nečesa na našem zaslonu. Ta razred bo shranil zgornje leve (x1 in y1) in spodnje desne (x2 in y2) koordinate katerekoli komponente naše igre.

Takole bi lahko zapisali položaj sličice s koordinatami:



Poklicali bomo naš nov razred Coords, ki bo vseboval le `__init__` funkcijo, kjer posredujemo štiri parametre (x1, y1, x2, in y2). Tukaj je koda, ki jo želimo dodati (postavite jo takoj za razredom Game):

```
class Coords:
    def __init__(self, x1=0, y1=0, x2=0, y2=0):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
```

Upoštevajte, da je vsak parameter shranjen kot spremenljivka objekta z istim imenom (x1, y1, x2 in y2). Objekte tega razreda bomo kmalu uporabili.

Preverjanje trkov

Ko vemo, kako shraniti položaj sličic, potrebujemo način, da ugotovimo, če je ena sličica trčila v drugo. Ko Mr. Stickman skače po zaslonu, lahko udari v katero od platform. Za lažje reševanje bomo ta problem razbili na dva manjša problema: preverili bom, če sličice trčijo navpično in če trčijo vodoravno. Nato rešitvi združimo in vidimo, če sta se sličici zadeli v katerikoli smeri!

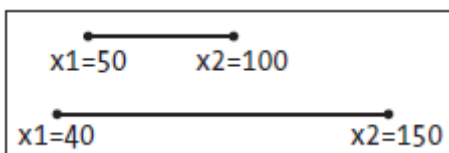
Trk sličic vodoravno

Najprej bomo ustvarili funkcijo `within_x`, da ugotovimo, ali je en niz x koordinat (`x1` in `x2`) prešel preko drugega niza x koordinat (`spet`, `x1` in `x2`). Obstaja več kot en način, da to storite, vendar tukaj je preprost pristop, ki ga lahko dodate tik pod `Coords` razred:

```
def within_x(co1, co2):
(1) if co1.x1 > co2.x1 and co1.x1 < co2.x2:
(2)     return True
(3) elif co1.x2 > co2.x1 and co1.x2 < co2.x2:
(4)     return True
(5) elif co2.x1 > co1.x1 and co2.x1 < co1.x2:
(6)     return True
(7) elif co2.x2 > co1.x1 and co2.x2 < co1.x2:
(8)     return True
(9) else:
(10)    return False
```

Funkcija `within_x` sprejme parametra `co1` in `co2` objektov `Coords`. Pri (1), preverimo, ali je najbolj levi položaj koordinate prvega objekta (`co1.x1`) med najbolj levim (`co2.x1`) in najbolj desnim položajem (`co2.x2`) koordinate drugega predmeta. Če je, vrnemo `True`.

Oglejmo si primer na dveh črtah s prekrivajočimi x koordinatami, da bomo lažje razumeli. Vsaka črta se začne pri `x1` in konča pri `x2`.



Prva črta v tem diagramu (`co1`) se začne pri 50 (`x1`) in konča pri 100 (`x2`). Druga črta (`co2`) se začne pri 40 in konča pri 150. V tem primeru, ker je položaj `x1` prve črte med pozicijami `x1` in `x2` druge črte, bi bil prvi `if` stavek `True` za ti dve koordinati.

Z `elif` pri (3), preverimo, če je najbolj desni položaj prve črte (`co1.x2`) med najbolj levim (`co2.x1`) in najbolj desnim položajem (`co2.x2`) druge črte. Če je, vrnemo `True` pri (4). Dva `elif` pri (5) in (6) sta skoraj enaka: preverjata najbolj levi in najbolj desni druge črte (`co2`) proti prvi (`co1`).

Če noben pogoj od `if` stavkov ni izpolnjen, pridemo do `else` pri (7) in vrnemo `False` (8). To bi dejansko pomenilo: "Ne, oba objekta se z vodoravnimi koordinatami ne križata med seboj."

Če si želite ogledati primer delovanja funkcije, pogledajte na črti na diagramu. Položaja `x1` in `x2` prve črte sta 40 in 100, položaja `x1` in `x2` druge črte sta 50 in 150. Kaj se zgodi, ko pokličemo funkcijo `within_x`, ki smo jo zapisali:

```
>>> c1 = Coords(40, 40, 100, 100)
>>> c2 = Coords(50, 50, 150, 150)
>>> print(within_x(c1, c2))
True
```

Funkcija vrne True. To je prvi korak k temu, da ugotovimo, če se je ena sličica zadela druge. Na primer, ko bomo ustvarili razrede za Mr. Stickmana in za platforme, bomo lahko ugotovili, če se njihove x koordinate križajo.

Ni res dobra programska praksa z veliko if in elif stavki, ki vrnejo isto vrednost. Da bi rešili ta problem, lahko skrajšamo funkcijo within_x, tako da obkrožimo vsak pogoj z oklepaji in jih povežemo s ključno besedo ali. Funkcija bi potem izgledala takole:

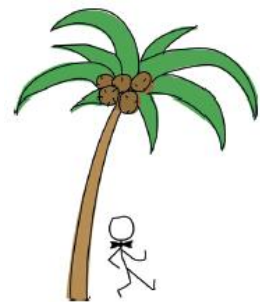
```
def within_x(co1, co2):
    if (co1.x1 > co2.x1 and co1.x1 < co2.x2) \
        or (co1.x2 > co2.x1 and co1.x2 < co2.x2) \
        or (co2.x1 > co1.x1 and co2.x1 < co1.x2) \
        or (co2.x2 > co1.x1 and co2.x2 < co1.x1):
        return True
    else:
        return False
```

Če želite zapisati if stavek v več vrsticah, da ne pride do zapisa v res dolgo vrstico, se uporabi obrnjena poševnica (\), kot je prikazano zgoraj.

Trk sličic navpično

Prav tako moramo vedeti, če se sličice zadenejo navpično. Funkcija within_y je zelo podobna funkciji within_x. Ustvarimo jo tako, da preverimo, če je y1 prve koordinate med y1 in y2 druge koordinate, nato pa obratno. Tukaj je funkcija, ki jo je potrebno dodati (dajte jo pod funkcijo within_x) - tokrat bomo napisali kodo s krajšo različico (in z več if, elif stavki):

```
def within_y(co1, co2):
    if (co1.y1 > co2.y1 and co1.y1 < co2.y2) \
        or (co1.y2 > co2.y1 and co1.y2 < co2.y2) \
        or (co2.y1 > co1.y1 and co2.y1 < co1.y2) \
        or (co2.y2 > co1.y1 and co2.y2 < co1.y1):
        return True
    else:
        return False
```



Sestavimo skupaj: naša končna koda za preverjanje trka

Ko smo ugotovili, če se en nabor x koordinat križa z drugim in naredili enako za y koordinate, lahko napišemo funkcije, da ugotovimo, če je sličica zadela drugo in s katere strani. To bomo storili s funkcijami collided_left, collided_right, collided_top in collided_bottom.

Funkcija collided_left

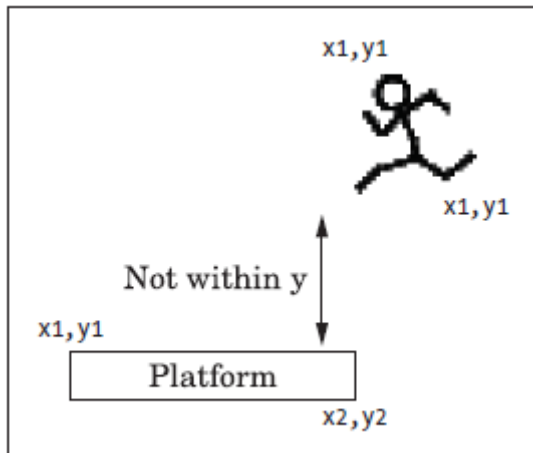
Tukaj je koda za funkcijo collided_left, ki jo lahko dodate pod dvema within funkcijama, ki smo ju pravkar ustvarili:

```
(1) def collided_left(co1, co2):
(2)     if within_y(co1, co2):
(3)         if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
```

```
(4)         return True
(5)     return False
```

Ta funkcija nam pove, če je leva stran (vrednost x1) prvega objekta zadela drugi objekt.

Funkcija sprejme dva parametra: co1 (prvi objekt koordinat) in co2 (drugi objekt koordinat). Kot vidite pri (1), preverimo, če sta dva objekta trčila po višini, z uporabo funkcije `within_y` pri (2). Navsezadnje nima smisla preverjati, če je Mr. Stickman zadel platformo, če je nad njo:



Pri (3), pogledamo, če je leva koordinata prvega objekta (`co1.x1`) dosegla položaj `x2` drugega objekta (`co2.x2`) - to je manj ali enako položaju `x2`. Preverimo tudi, da ni prečkal položaja `x1`. Če je zadel s strani, vrnemo `True` pri (4). Če nobeden od if pogojev ni izpolnjen, vrnemo `False` pri (5).

Funkcija `collided_right`

Funkcija `collided_right` izgleda podobno kot `collided_left`:

```
def collided_right(co1, co2):
(1) if within_y(co1, co2):
(2)     if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
(3)         return True
(4) return False
```

Kot pri `collided_left` preverimo, če so koordinate `y` v območju trka z uporabo funkcije `within_y` pri (1). Nato preverimo, če je vrednost `x2` prvega med položajema `x1` in `x2` drugega objekta pri (2) in vrnemo `True` (3), če je. V nasprotnem primeru vrnemo `False` (4).

Funkcija `collided_top`

Funkcija `collided_top` je zelo podobna dvema funkcijama, ki smo ju ravno dodali.

```
def collided_top(co1, co2):
(1) if within_x(co1, co2):
(2)     if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
         return True
     return False
```

Razlika je v tem, da tokrat preverimo, če se koordinate križajo vodoravno, z uporabo funkcije `within_x` pri (1). Nato, pri (2), preverimo, če je najvišji položaj prve koordinate (`co1.y1`) prešel položaj

y2 drugega objekta, ne pa tudi položaja y1. Če je tako, vrnemo True (pomeni, da je vrh prve koordinate zadel drugo koordinato).

Funkcija `collided_bottom`

Seveda ste slutili, da bo ena od teh štirih funkcij nekoliko drugačna. Tukaj je funkcija `collided_bottom`:

```
def collided_bottom(y, co1, co2):
(1)     if within_x(co1, co2):
(2)         y_calc = co1.y2 + y
(3)         if y_calc >= co2.y1 and y_calc <= co2.y2:
(4)             return True
(5)     return False
```

Ta funkcija sprejme dodaten parameter `y`, vrednost, ki jo dodamo položaju `y` prve koordinate. Pri (1), preverimo, če se koordinate križajo vodoravno (kot smo naredili pri `collided_top`). Nato dodamo vrednost parametra `y` prvemu objektu `y2` in shranimo rezultat v spremenljivko `y_calc` pri (2). Če je pri (3) novoizračunana vrednost med `y1` in `y2` druge koordinate, vrnemo True (4), ker je dno koordinate `co1` doseglo vrh koordinate `co2`. Vendar, če nobeden od pogojev ni izpolnjen, vrnemo False (5).

Dodatni parameter `y` potrebujemo, ker bi Mr. Stickman lahko padel s platforme. Za razliko od drugih funkcij trka, moramo preveriti, če bi trčil na dno, namesto, če je že. Če pade s platforme in leti po zraku, naša igra ne bi bila zelo realistična; tako pri hoji preverimo, če je naletel na nekaj na levi ali desni strani. Ko preverjamo pod njim, gledamo, če bi trčil s platformo; sicer pada naprej!

Izdelava razreda `Sprite`

Starševski razred za naše sličice bomo imenovali `Sprite`. Ta razred bo imel dve funkciji: `move`, za premikanje sličic in `coords` za vračilo položaja sličice. Tukaj je koda za `Sprite` razred.

```
class Sprite:
(1)     def __init__(self, game):
(2)         self.game = game
(3)         self.endgame = False
(4)         self.coordinates = None
(5)     def move(self):
(6)         pass
(7)     def coords(self):
(8)         return self.coordinates
```

Funkcija `__init__` razreda `Sprite`, ki je definirana pri (1), sprejme en parameter: `game`. Ta parameter bo objekt `game`. Potrebujemo `ga`, da bodo vse sličice, ki jih ustvarimo, lahko dostopale do seznama drugih sličic v igri. Parameter `game` shranimo kot objektno spremenljivko pri (2).

Pri (3) shranimo spremenljivko `endgame`, ki jo bomo uporabili za označitev konca igre. (Trenutno je nastavljen na False.) Zadnja spremenljivka `coordinates` pri (4) je nastavljena na prazno (None).

Funkcija `move`, ki je definirana pri (5), v tem staršu ne izvaja ničesar, zato tu damo le `pass` (6).

Funkcija `coords` (7) enostavno vrne objektno spremenljivko `coordinates` pri (8).

Tako ima naš razred `Sprite` funkcijo `move`, ki ne počne ničesar, in `coords` funkcijo, ki vrne koordinate. Ne zveni zelo uporabno, ali ne? Vendar pa vemo, da bodo vsi razredi, ki imajo starša `Sprite`, vedno

imeli funkciji `move` in `coords`. Tako v glavni zanki lahko pri vseh sličicah kličemo `move` in ne bo prišlo do napake. Zakaj ne? Ker ima vsak objekt `Sprite` to funkcijo.

Opomba: Razredi s funkcijami, ki ne delajo veliko, so dejansko precej pogosti pri programiranju. Na nek način so nekakšno zagotovilo, da imajo vsi otroci razreda enako funkcionalnost, tudi če v nekaterih primerih funkcije v razredih otrok ne delajo ničesar.

Dodajanje platform

Zdaj bomo dodali platforme. Poklicali bomo naš razred za objekte platform `PlatformSprite`, ki bo otrok razred `Sprite`. Funkcija `__init__` za ta razred bo imela parametre `game` (kot ga ima tudi starš `Sprite`), kot tudi `image`, `x` in `y` položaje ter širino in višino slike. Tukaj je koda za razred `PlatformSprite`:

```
(1) class PlatformSprite(Sprite):
(2)     def __init__(self, game, photo_image, x, y, width, height):
(3)         Sprite.__init__(self, game)
(4)         self.photo_image = photo_image
(5)         self.image = game.canvas.create_image(x, y, \
            image=self.photo_image, anchor='nw')
(6)         self.coordinates = Coords(x, y, x + width, y + height)
```

Ko definiramo razred `PlatformSprite` (1), mu damo en parameter: ime matičnega razreda (`Sprite`). `__init__` (2) ima sedem parametrov: `self`, `game`, `photo_image`, `x`, `y`, `width` in `height`.

Pri (3) pokličemo funkcijo `__init__` starševskega razreda `Sprite` s pomočjo `self` in `game` kot vrednosti parametrov, ker poleg `self` `Sprite` funkcija `__init__` sprejme le še en parameter: `game`.

Na tej točki, bi pri ustvarjenju objekta `PlatformSprite`, ta imel vse spremenljivke od svojega starševskega razreda (`game`, `endgame` in `coordinates`), preprosto zato, ker smo poklicali funkcijo `__init__` v `Sprite`.

Pri (4) shranimo parameter `photo_image` kot objektno spremenljivko, in pri (5) uporabimo spremenljivko `canvas` iz objekta `game`, da narišemo sliko na zaslonu s `create_image`.

Končno ustvarimo objekt `Coords` s parametri `x` in `y` kot prva dva argumenta. Tema dvema prištejemo še širino in višino za druga dva parametra pri (6).

Čeprav je spremenljivka `coordinates` nastavljena na `None` v starševskem razredu `Sprite`, smo jo spremenili v našem otroškem razredu `PlatformSprite` na dejansko lokacijo.

Dodajanje objekta platform

Dodajmo platformo v igro, da vidimo, kako izgleda. Spremenite zadnji dve vrstici datoteke (`stickmangame.py`), kot sledi:

```
(1) g = Game()
(2) platform1 = PlatformSprite(g, PhotoImage(file="platform1.png"), \
    0, 480, 100, 10)
(3) g.sprites.append(platform1)
(4) g.mainloop()
```

Kot lahko vidite, se vrstice (1) in (4) nista spremenili, pri (3) pa ustvarimo objekt razreda PlatformSprite, tako da ga mu pošljemo spremenljivko game (g), skupaj z objektom PhotoImage (ki uporablja prvo naših platform slik, platform1.png). Prav tako ga damo na svoj položaj (0 levo in 480 navzdol, blizu dna platna), skupaj z višino in širino slike (100 pik širine in 10 pik višine). To sličico dodamo na seznam sličic sprites (3).

Če zdaj poženetite igro, bi morali videti ozadje s platformo, na primer:



Dodajanje skupine platform

Dodajmo še ostale platforme. Vsaka platforma bo imela različne položaje x in y, tako da bodo razpršene po zaslonu. Tukaj je koda:

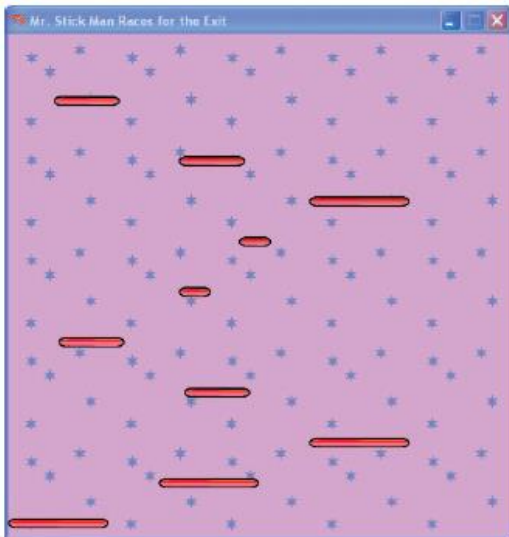
```
g = Game ()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.png"), 0, 480,
100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.png"), 150, 440,
100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.png"), 300, 400,
100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.png"), 300, 160,
100, 10)
platform5 = PlatformSprite(g, PhotoImage(file="platform2.png"), 175, 350,
66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.png"), 50, 300,
66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.png"), 170, 120,
66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.png"), 45, 60, 66,
10)
platform9 = PlatformSprite(g, PhotoImage(file="platform3.png"), 170, 250,
32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.png"), 230, 200,
32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
```

```

g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
g.mainloop ()

```

Ustvarimo veliko objektov PlatformSprite, ki jih shranimo kot spremenljivke platform1, platform2, platform3 in tako naprej do platform10. Nato dodamo vsako platformo seznamu sprites, ki smo ga ustvarili v razred Game. Če bi igrali zdaj, bi moralo izgledati takole:



Ustvarili smo osnovo naše igre! Zdaj smo pripravljeni na dodajanje našega glavnega lika, Mr. Stickmana.

Kaj ste se naučili

V tem poglavju ste ustvarili razred Game in narisali ozadje na zaslon kot nekakšno tapeto. Naučili ste se ali je vodoravni ali navpični položaj znotraj omejitve drugih vodoravnih ali navpičnih pozicij z ustvarjanjem funkcije withi_x in within_y. Nato ste uporabili te funkcije za ustvarjanje novih funkcij, da ugotovite, če je objekt trčil z drugim. Te funkcije bomo uporabili v naslednjem poglavju, ko bomo animirali Mr. Stickmana in moramo zaznati, če trči s platformo, ko se premika po platnu.

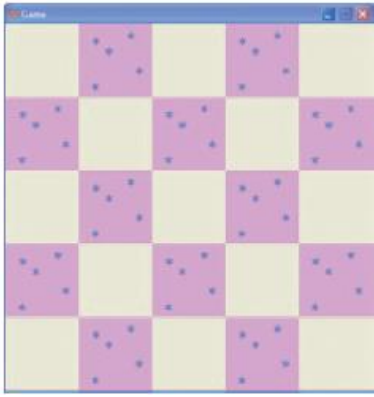
Ustvarili smo tudi starševski razred Sprite in njegov prvi otroški razred PlatformSprite, s katerim smo narisali platforme na platno.

Vaje

Z naslednjimi primeri lahko preizkusite različne slike ozadja. Preverite odgovore na <https://nostarch.com/pythonforkids>.

1: Šahovnica

Poskusite spremeniti razred Game tako, da bo ozadje izrisano kot šahovnica:



2: Dvobarvna šahovnica

Ko ugotovite, kako ustvariti šahovnico, poskusite z uporabo dveh izmenjujočih slik. Narišite še eno ozadje (z uporabo grafičnega programa) in nato spremenite razred Game, tako da prikazuje šahovnico z dvema različnima slikama.

3: Knjižna polica in svetilka

Za tapeto lahko uporabite različne slike ozadja, da bo igra bolj zanimiva. Ustvarite kopijo slike ozadja in nato na njej narišite knjižno polico. Lahko narišete tudi mizo s svetilko na njej ali okno. Nato slike razmestite po zaslonu s spreminjanjem razreda Game tako, da se naložijo (in prikažejo) tri ali štiri različna ozadja.

17. poglavje: Izdelava Mr. Stickmana

V tem poglavju bomo ustvarili **glavni lik** naše igre. To bo najzahtevnejše kodiranje do sedaj, ker mora g. Stickman **teči levo in desno, skočiti, se ustavi**, ko pride na platformo in **pasti**, ko pride čez rob platforme. Uporabili bomo **povezave na dogodke** za levo in desno smerno tipko za tek v levo in desno smer, za skok pa bomo uporabili preslednico.

Inicializacija glavnega lika

Funkcija `__init__` za naš nov razred glavnega lika bo videti zelo podobna drugim razredom naše igre doslej. Začnimo s poimenovanjem našega novega razred: `StickFigureSprite`. Kot pri platformi ima ta razred starševski razred: `Sprite`.

```
class StickFigureSprite(Sprite):
    def __init__(self, game):
        Sprite.__init__(self, game)
```

Ta koda izgleda kot tista za razred `PlatformSprite` v poglavju 16 razen, da ne uporabljamo dodatnih parametrov (razen `self` in `game`). Razlog je, da bo v nasprotju s `PlatformSprite` v igri uporabljen le en objekt `StickFigureSprite`.

Nalaganje slik Mr. Stickmana

Ker imamo na zaslonu veliko platform različnih velikost, slike platforme posredujemo kot parameter funkcije `__init__` (kot bi rekli, "Poslušaj `PlatformSprite`, uporabi to sliko, ko se narišeš na zaslonu. "). Ker obstaja le ena glavna figura, je ni smiselno naložiti zunaj in jo nato prenesti kot parameter. Razred `StickFigureSprite` bo vedel, kako naložiti lastne slike.

Naslednjih nekaj vrstic funkcije `__init__` opravi to delo: naložijo se po tri leve slike (ki jih bomo uporabili za animacijo figure, ki teče levo) in tri desne slike (uporabljene za figuro, ki teče desno). Te slike moramo naložiti zdaj, ker jih ne želimo nalagati vsakič posebej, ko se prikaže na zaslonu (to bi predolgo trajalo in naša igra bi tekla počasi).

```
class StickFigureSprite(Sprite):
    def __init__(self, game):
        Sprite.__init__(self, game)
(1)    self.images_left = [
        PhotoImage(file="stickman-L1.png"),
        PhotoImage(file="stickman-L2.png"),
        PhotoImage(file="stickman-L3.png")
    ]
(2)    self.images_right = [
        PhotoImage(file="stickman-R1.png"),
        PhotoImage(file="stickman-R2.png"),
        PhotoImage(file="stickman-R3.png")
    ]
(3)    self.image = game.canvas.create_image(200, 470, \
        image=self.images_left[0], anchor='nw')
```



Ta koda naloži vsako od treh levih slik, ki jih bomo uporabili za animacijo teka v levo, in tri desne slike, ki jih bomo uporabili za animiranje teka v desno. Pri (1) in (2) ustvarimo spremenljivke objekta `images_left` in `images_right`. Vsaka vsebuje seznam objektov `PhotoImage`, ki smo jih ustvarili v

poglavju 15, s sličicami teka v levo in desno. Prvo sliko pripravimo pri (3) z `images_left [0]` z uporabo funkcije `platna: create_image` na položaju (200, 470), ki postavlja sliko na sredino zaslona na dno platna. Funkcija `create_image` vrne številko, ki identificira sliko na platnu. Ta identifikator shranimo v spremenljivko objekta `image` za kasnejšo uporabo.

Nastavitev spremenljivk

Naslednji del funkcije `__init__` nastavlja nekaj več spremenljivk, ki jih bomo kasneje uporabljali v tej kodi.

```
self.images_right = [  
    PhotoImage(file="stick-R1.gif"),  
    PhotoImage(file="stick-R2.gif"),  
    PhotoImage(file="stick-R3.gif")  
]  
self.image = game.canvas.create_image(200, 470, \  
    image=self.images_left[0], anchor='nw')  
(1) self.x = -2  
(2) self.y = 0  
(3) self.current_image = 0  
(4) self.current_image_add = 1  
(5) self.jump_count = 0  
(6) self.last_time = time.time()  
(7) self.coordinates = Coords()
```

Pri (1) in (2) sta objektni spremenljivki `x` in `y`, ki hranita hitrost premikanje vodoravno in navpično. Uporabili ju bomo pri funkciji `move`.

Kot ste se naučili v poglavju 13, za animacijo z modulom `tkinter` dodamo vrednost položaju `x` ali `y`, da objekt premikamo po platnu. Z nastavitvijo `x` na `-2` in `y` na `0`, odštejemo `2` od položaja `x` in ničesar ne dodamo navpičnemu položaju, da se slička premakne v levo.

Opomba: Ne pozabite, da negativna številka `x` pomenijo premik v levo na platnu, pozitivna `x` številka pomenijo premik desno. Negativna številka `y` pomeni, da se slička premakne navzgor in pozitivna številka `y` pomeni premik navzdol.

Pri (3) ustvarimo objektno spremenljivko `current_image` za shranjevanje indeksa trenutne slike, prikazane na zaslону. Naš seznam levo obrnjenih slik, `images_left`, vsebuje `stickman-L1.png`, `stickman-L2.png` in `stickman-L3.png`. To so indeksni položaji `0`, `1`, in `2`.

Pri (4) bo spremenljivka `current_image_add` vsebovala številko, ki jo dodamo indeksnemu položaju, shranjenem v `current_image`, da dobimo naslednji indeksni položaj. Če se kaže slika na indeksni poziciji `0`, dodamo `1`, da dobimo naslednjo sliko na indeksni poziciji `1` in nato še enkrat dodajte `1`, da dobite končno sliko iz seznama z indeksom `2`. (uporabo spremenljivke za animacijo boste videli v naslednjem poglavju.)

Spremenljivka `jump_count` pri (5) je števec, ki ga bomo uporabili med skokom. Spremenljivka `last_time` bo zapisala zadnjo spremembo figure. Trenutni čas shranjujemo s časovno funkcijo časovnega modula pri (6).

Pri (7) nastavimo objektno spremenljivko `coordinates` na objekt razreda `Coords`, brez nastavljenih začetnih parametrov (`x1`, `y1`, `x2` in `y2` so vsi `0`). Za razliko od ploščadi se koordinate glavnega lika spreminjajo, zato jih bomo nastavljali kasneje.

Povezovanje s tipkami

V zadnjem delu funkcije `__init__` so povezovalne funkcije za tipke, ki se poženejo ob pritisku na tipko.

```
self.jump_count = 0
self.last_time = time.time()
self.coordinates = Coords()
game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
game.canvas.bind_all('<space>', self.jump)
```

Povežujemo `<KeyPress-Left>` s funkcijo `turn_left`, `<KeyPress-Right>` s funkcijo `turn_right` in `<space>` s funkcijo `jump`. Zdaj moramo ustvariti te funkcije, da bomo lahko premikali glavno figuro.

Obračanje glavne figure levo in desno

Funkcije `turn_left` in `turn_right` poskrbita, da glavna figura ne skače in nato nastavi vrednost objektne spremenljivke `x`, da ga premikata levo in desno. (Če lik skoči, naša igra ne dovoljuje, da bi spremenil smer v zraku.)

```
game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
game.canvas.bind_all('<space>', self.jump)
```

```
(1) def turn_left(self, evt):
(2)     if self.y == 0:
(3)         self.x = -2

(4) def turn_right(self, evt):
(5)     if self.y == 0:
(6)         self.x = 2
```



Python pokliče funkcijo `turn_left`, ko igralec pritisne levo smerno tipko in prenese objektu podatek o dogodku kot parameter. Ta predmet se imenuje event object in parametru damo ime `evt`.

Opomba: Event object za nas trenutno ni pomemben, vendar ga moramo vključiti kot parameter naših funkcij (pri (1) in (4)), da ne dobimo napake, ker Python pričakuje, da bo tam. Event object vsebuje podatke o položaju `x` in `y` miške (mouse event), kodo tipke (keyboard event) in druge informacije. Za to igro nobena od teh informacij ni uporabna, zato ta parameter lahko varno prezremo.

Če želimo ugotoviti, ali figura skoči, preverimo vrednost objektne spremenljivke `y` pri (2) in (5). Če vrednost ni 0, je figura v skoku. V primeru, če je vrednost `y` enaka 0, nastavimo `x` na -2 za gibanje v levo (3) ali pa nastavimo na 2 za gibanje v desno (6). Vrednosti -1 ali 1 bi povzročili, da se lik premika prepočasi. (Ko bo igrice delovala, poskusite spremeniti to vrednost, da vidite razliko v hitrosti.)

Ustvarjanje skokov

Funkcija `jump` je zelo podobna `turn_left` in `turn_right` funkcijama.

```
def turn_right(self, evt):
    if self.y == 0:
        self.x = 2
```

```

def jump(self, evt):
(1)     if self.y == 0:
(2)         self.y = -4
(3)         self.jump_count = 0

```

Ta funkcija ima parameter evt (objekt dogodka), ki ga lahko prezremo, ker nam niso potrebne informacije o dogodku. Če se izvaja ta funkcija, vemo, da smo pritisnili preslednico.

Ker hočemo našo figuro, da skoči le, če še ni skočila, pri (1) preverimo, če je y enak 0. Če figura ne skače, pri (2) nastavimo y na -4 (premikamo jo navpično navzgor na zaslonu) in nastavimo jump_count na 0 pri (3). jump_count bomo uporabili, da glavni lik ne bo stalno skakal. Namesto tega mu bomo dovolili, da skoči le določeno število krat in ga potem spet spustimo, kakor da ga povleče gravitacija. To kodo bomo dodali v naslednjem poglavju.

Kaj imamo do sedaj

Oglejmo si definicije razredov in funkcij, ki jih imamo sedaj v naši igri in kje morajo biti v vaši datoteki.

Na vrhu programa bi morali imeti uvoze modulov, ki jim sledita razreda Game in Coords. Razred Game se bo uporabljala za glavnega nadzornika naše igre in objekti razreda Coords se uporabljajo za položaje stvari v naši igri (kot so platforme in Mr. Stickman):

```

from tkinter import *
import random
import time
class Game:
...
class Coords:
...

```

Nato morate imeti funkciji within (ki povesta, ali so koordinate ene sličice "znotraj" istega območja druge sličice), starševski razred Sprite (ki je starševski razred vseh sličic v naši igri), razred PlatformSprite in začetek razreda StickFigureSprite. PlatformSprite je bil uporabljen za ustvarjanje objektov platform, na katere naša glavna figura skače, StickFigureSprite pa uporabimo za glavnega igralca v naši igri:

```

def within_x(co1, co2):
...
def within_y(co1, co2):
...
def collided_left(co1, co2):
...
def collided_right(co1, co2):
...
def collided_top(co1, co2):
...
def collided_bottom(y, co1, co2):
...
class Sprite:
...
class PlatformSprite(Sprite):
...
class StickFigureSprite(Sprite):
...

```


Na koncu programa morate imeti kodo, ki ustvari vse objekte v naši igri: objekt igra in platforme. Zadnja vrstica je, kjer kličemo funkcijo mainloop.

```
g = Game ()
platform1 = PlatformSprite (g, PhotoImage (file="platform1.png"), \
0, 480, 100, 10)
...
g.sprites.append(platform1)
...
g.mainloop ()
```

Če je vaša koda nekoliko drugačna, ali imate težave pri delovanju, lahko vedno preskočite do konca poglavja 18, kjer boste našli celotno kodo.

Kaj ste se naučili

V tem poglavju smo začeli delati na razredu za našo glavno figuro. Trenutno smo ustvarili objekt tega razreda, ki razen nalaganja slik še ne naredi drugega. Ta razred vsebuje funkcije za odziv na tipke (ko igralec pritisne levo ali desno smerno tipko ali preslednico).

V naslednjem poglavju bomo končali igro. Napisali bomo funkcije za razred StickFigureSprite za prikaz in animiranje slike in jo premikali po zaslonu. Dodali bomo tudi izhod (vrata), ki jih Mr. Stickman skuša doseči.

18. poglavje: Dokončanje igre Mr. Stickman

V prejšnjih treh poglavjih smo razvijali našo igro: Mr. Stickman drvi proti izhodu. Ustvarili smo grafiko in nato napisal kodo za dodajanje slike ozadja, platform in glavnega lika. V tem poglavje, bomo zapolnili manjkajoče dele za animacijo glavnega lika in dodali vrata.

Celotno kodo za igro boste našli na koncu tega poglavja. Če se izgubite ali se zmedete pri pisanju nekaterih delov kode, jo primerjajte s svojo, da vidite, kaj bi lahko šlo narobe.

Animiranje glavnega igralca

Doslej smo ustvarili osnovni razred za naš glavni lik, nalaganje slik, ki jih bomo uporabljali, in nekatere funkcije. Če poženet program na tej točki, pa se bo zgodilo zelo malo.

Zdaj bomo dodali preostale funkcije v razred `StickFigureSprite`, ki smo ga ustvarili v poglavju 17: `animate`, `move` in `coords`. Funkcija `animate` bo uporabila različne slike za izris, `move` bo določila kam se mora lik premakniti in `coord` bo vrnila aktualni položaj. (Za razliko od platform, moramo za glavni lik stalno obnavljati položaj.)



Ustvarjanje funkcije `Animate`

Najprej bomo dodali funkcijo `animate`, ki jo bomo morali preverjati za gibanje in ustrezno spremeniti sliko.

Preverjanje gibanja

Slike glavnega lika ne želimo spreminjati prehitro, ker gibanje ne bo izgledalo realistično. Če se slike spreminjajo prehitro, ni pravega učinka animacije.

Prva polovica funkcije `animate` preveri, če se glavni lik giblje levo ali desno in nato uporabi spremenljivko `last_time` za spremembo trenutne slike. Ta spremenljivka nam bo pomagal nadzirati hitrost naše animacije. Funkcija bo za `jump` funkcijo, ki smo jo dodali razredu `StickFigureSprite` v poglavju 17.

```
def jump(self, evt):
    if self.y == 0:
        self.y = -4
        self.jump_count = 0

def animate(self):
(1)     if self.x != 0 and self.y == 0:
(2)         if time.time() - self.last_time > 0.1:
(3)             self.last_time = time.time()
(4)             self.current_image += self.current_image_add
(5)             if self.current_image >= 2:
(6)                 self.current_image_add = -1
(7)             if self.current_image <= 0:
(8)                 self.current_image_add = 1
```

V stavku if pri (1), preverimo, če x ni enak 0, da preverimo, če se lik premika (levo ali desno) in preverimo, če je y enak 0, da ugotovimo, če je lik v skoku. Če so pogoji izpolnjeni, moramo animirati naš lik; če ne, mirno stoji, zato ni potrebe po risanju. Če se lik ne giblje, gremo ven iz funkcije in preostala koda se ne izvaja.

Pri (2), izračunamo čas od zadnjega klica funkcije animate, tako da od trenutnega časa odštejemo zadnji čas z uporabo time.time (). Ta izračun se uporablja za odločitev o naslednjem izrisu sličice. Če je rezultat večji od desetinke sekunde (0.1), nadaljujemo z blokom kode pri (3). Nastavimo last_time spremenljivko - v bistvu ponastavimo štoparico za naslednjo spremembo slike.

Pri (4) dodamo vrednost objektne spremenljivke current_image_add spremenljivki current_image, ki shrani indeksni položaj trenutno prikazane slike. Ne pozabite, da smo ustvarili current_image_add v funkciji __init__ v Poglavju 17. Ko se prvič pokliče funkcija animate je vrednost spremenljivke že nastavljena na 1.







Pri (5), preverimo, če je vrednost indeksnega položaja v current_image je večja ali enaka 2, in če je, spremenimo vrednost current_image_add na -1 pri (6). Postopek je podoben pri (7), ko dosežemo 0, moramo znova začeti štetje, kar naredimo pri (8).

OPOMBA: Če imate težave z ugotovitvijo, kako to kodo zamakniti je tukaj namig: na začetku (1) je 8 presledkov in 20 pri (8).

Da bi vam pomagali razumeti, kaj se dogaja v tej funkciji, si predstavljajte, da imate na tleh v vrsti zaporedje barvnih blokov. Prst premaknete iz enega bloka na drugega in vsak blok, ki ga vaš prst kaže (1, 2, 3, 4 in tako naprej) ima številko (spremenljivka current_image). Število za koliko blokov se vaš prst premakne (kaže na en blok naenkrat) je številka shranjena v spremenljivki current_image_add. Ko se prst premakne za en blok naprej, dodate vsakič 1 in ko pride do konca se začne pomikati nazaj, odšteva se 1 (dodaja se -1).

Koda, ki smo jo dodali k naši funkciji animate izvede to opravilo, le da imamo namesto barvnih blokov tri slike za vsako smer shranjene v seznamu. Indeksne pozicije teh slik so 0, 1 in 2. Pri animiranju slike glavnega lika, ko pridemo do zadnje slike, začnemo šteti navzdol in ko pridemo do prve slike, ponovno začnemo šteti navzgor. Kot rezultat, ustvarimo sliko tekača.

Naslednje kaže, kako se premikamo po seznamu slik, z indeksnimi položaji, ki jih izračunamo v funkciji animate.

Position 0	Position 1	Position 2	Position 1	Position 0	Position 1
Counting up	Counting up	Counting up	Counting down	Counting down	Counting up
					

Spreminjanje slike

V naslednji polovici animirane funkcije spreminjamo trenutno prikazano sliko z izračunanim indeksnim položajem.

```
def animate(self):
    if self.x != 0 and self.y == 0:
        if time.time() - self.last_time > 0.1:
            self.last_time = time.time()
            self.current_image += self.current_image_add
            if self.current_image >= 2:
                self.current_image_add = -1
            if self.current_image <= 0:
                self.current_image_add = 1
(1) if self.x < 0:
(2)     if self.y != 0:
(3)         self.game.canvas.itemconfig(self.image, \
            image=self.images_left[2])
(4)     else:
(5)         self.game.canvas.itemconfig(self.image, \
            image=self.images_left[self.current_image])
(6) elif self.x > 0:
(7)     if self.y != 0:
(8)         self.game.canvas.itemconfig(self.image, \
            image=self.images_right[2])
(9)     else:
(10)        self.game.canvas.itemconfig(self.image, \
            image=self.images_right[self.current_image])
```

Pri (1), če je x manjši od 0, se lik premika levo in Python se premakne v blok kode prikazane pri (2) do (5), kar preveri, če y ni enak 0 (kar pomeni, da lik skače). Če y ni enak 0 (se lik premika navzgor ali navzdol – z drugimi besedami, skače), uporabimo funkcijo `itemconfig` in spremenite prikazano sliko na zadnjo sliko v našem seznamu levo obrnjenih slik pri (3) (`images_left[2]`). Ker lik skače, uporabimo sliko v razkoraku, da bo animacija bolj realistična:



Če lik ne skače (to pomeni, da je y enak 0), se izvaja `else`, ki se začne pri (4) z uporabo funkcije `itemconfig` spremenimo prikazano sliko na kateremkoli indeksnem položaju v spremenljivki `current_image`, kot je prikazano v kodi pri (5).

Pri (6), pogledamo, če se lik premika desno (x je večji od 0), Python začne izvajati blok prikazan med (7) in (10). Ta koda je zelo podobna prvemu bloku in ponovno preverja, če lik skače, da izriše ustrezno sliko, tokrat iz seznama `images_right`.

Pozicija glavnega lika

Ker bomo morali ugotoviti, kje na zaslonu je lik (ker se premika okoli), se bo funkcija `Coords` razlikovala od drugih funkcij razreda `Sprite`. Uporabili bomo `coords` funkcijo platna in nato uporabili te vrednosti za nastavitve `x1`, `y1` in `x2`, `y2` spremenljivke `coordinates`, ki smo jo ustvarili v funkciji `__init__` na začetku Poglavlja 17. Tu je koda, ki jo lahko dodate po funkciji `animate`:

```
def coords(self):
(1)     xy = list(self.game.canvas.coords(self.image))
(2)     self.coordinates.x1 = xy[0]
(3)     self.coordinates.y1 = xy[1]
(4)     self.coordinates.x2 = xy[0] + 27
(5)     self.coordinates.y2 = xy[1] + 30
        return self.coordinates
```

Ko smo v poglavju 16 ustvarili razred `Game`, je bila ena od objektovnih spremenljivk `canvas`. Pri (1) uporabljamo `coords` funkcijo tega platna s `self.game.canvas.coords` za določitev `x` in `y` položaja trenutne slike. Ta funkcija uporablja številko shranjeno v spremenljivki `image` kot identifikator za sliko na platnu.

Seznam shranimo v spremenljivki `xy`, ki zdaj vsebuje dve vrednosti: položaj `x` zgoraj-levo, shranjeno kot `x1` pri (2) in položaj `y` zgoraj-levo, shranjeno kot spremenljivka `y1` pri (3).

Ker so vse slike lika, ki smo jih ustvarili, 27 pik široke in 30 pik visoke, lahko določimo spremenljivki `x2` in `y2` z dodajanjem `x` (4) in `y` (5).

V zadnji vrstici funkcije vrnemo objektno spremenljivko `coordinates`.

Premikanje glavnega lika

Zadnja funkcija razreda `StickFigureSprite`, `move`, je odgovorna za dejansko premikanje lika po zaslonu. Povedati nam mora tudi, če se je lik v kaj zaletel.

Začetek funkcije `move`

Tukaj je koda za prvi del funkcije `move` - to pride za funkcijo `coords`:

```
def move(self):
(1)     self.animate()
(2)     if self.y < 0:
(3)         self.jump_count += 1
(4)         if self.jump_count > 20:
(5)             self.y = 4
(6)     if self.y > 0:
(7)         self.jump_count -= 1
```



Pri (1) pokličemo funkcijo `animate`, ki smo jo ustvarili prej v tem poglavju in spremeni trenutno prikazano sliko, če je potrebno. Pri (2) preverimo, če je vrednost `y` manjša od 0. Če je, vemo, da lik skače, ker ga negativna vrednost premika navzgor. (Ne pozabite, da je 0 na vrhu platna in spodnji del platna je 500.)

Pri (3) prištejemo 1 v `jump_count`, pri (4) pa rečemo, če je vrednost v `jump_count` večja od 20, moramo spremeniti `y` na 4, da bi lik začel padati (5).

Pri (6) pogledamo, če je vrednost `y` večja od 0 (pomeni, da lik pada), in če je, odštejemo 1 od `jump_count`, da zmanjšujemo dolžino skoka (pristajamo).

V naslednjih nekaj vrsticah funkcije `move` pokličemo funkcijo `coords`, ki nam pove, kje je naš lik na zaslonu in shranimo rezultat v spremenljivko `co`. Nato ustvarimo spremenljivke `left`, `right`, `top`, `bottom` in `falling`. To bomo uporabili v nadaljevanju funkcije.

```
if self.y > 0:
    self.jump_count -= 1

co = self.coords()
left = True
right = True
top = True
bottom = True
falling = True
```

Upoštevajte, da je vsaka spremenljivka nastavljena na boolean vrednost `True`. Te bomo uporabili za preverjanje, če je lik zadel ob kaj ali če pada.

Ali je lik zadel vrh ali dno platna?

Naslednji del funkcije `move` preveri, ali je naš lik zadel spodnji ali zgornji del platna. Tukaj je koda:

```
bottom = True
falling = True

(1) if self.y > 0 and co.y2 >= self.game.canvas_height:
(2)     self.y = 0
(3)     bottom = False
(4) elif self.y < 0 and co.y1 <= 0:
(5)     self.y = 0
(6)     top = False
```

Če lik pada po zaslonu, bo `y` večji kot 0, zato se moramo prepričati, da še ni zadel dna platna (sicer bo izginil s spodnjega dela zaslona). Za to, pri (1) preverimo, če je njegov položaj `y2` (spodnji del slike) večji ali enaka spremenljivki `canvas_height` objekta `game`. Če je, nastavimo vrednost `y` na 0 pri (2), da se padanje ustavi in nato nastavimo spremenljivko `bottom` na `False` pri (3), ki pove preostali kodi, da ne potrebujemo več preverjati, če je lik zadel dno.

Postopek ugotavljanja, če je lik zadel vrh zaslona, je zelo podoben. Pri (4) najprej preverimo, če lik skače (`y` je manjši od 0), potem pogledamo, če je njegov položaj `y1` manjši ali enak 0, kar pomeni, da je zadel vrh platna. Če sta oba pogoja izpolnjena, `y` nastavimo na 0 in ustavimo skok. Spremenljivko `top` nastavimo na `False` pri (6), da ostali kodi povemo, da ni več potrebno preverjati, če je bil zadel vrh.

Ali je lik zadel stranico platna?

Sledimo skoraj enakemu postopku kot v prejšnji kodi, da ugotovimo, če je lik zadel levo ali desno stran platna, kot sledi:

```
elif self.y < 0 and co.y1 <= 0:
    self.y = 0
    top = False
if self.x > 0 and co.x2 >= self.game.canvas_width:
```

```

self.x = 0
right = False
elif self.x < 0 and co.x1 <= 0:
self.x = 0
left = False

```

Koda pri (1) temelji na dejstvu, da lik teče desno, če je x večji od 0. Prav tako vemo, da zadane desno stran zaslona, če je položaj x2 (co.x2) večji ali enaka širini platna, shranjeno v game_width. Če sta obe izjavi resnični, nastavimo x na 0 (zaustavitev lika) in nastavimo spremenljivko right na False pri (3).

Trkanje z drugimi sličicami

Ko smo ugotovili, če je lik zadel robove zaslona, moramo preveriti, če je na zaslonu naletel na kaj drugega. Naslednjo kodo uporabimo, da se sprehodimo po sličicah in vidimo, če je lik zadel katerokoli od njih.

```

elif self.x < 0 and co.x1 <= 0:
self.x = 0
left = False
(1) for sprite in self.game.sprites:
(2)     if sprite == self:
(3)         continue
(4)     sprite_co = sprite.coords()
(5)     if top and self.y < 0 and collided_top(co, sprite_co):
(6)         self.y = -self.y
(7)         top = False

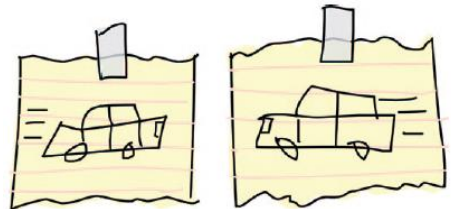
```

Pri (1), gremo skozi seznam sličic. Pri (2) preverimo, če gre za glavni lik (kot bi rekli: "če je ta sličica enaka kot jaz"), ni potrebe po ugotavljanju trčenja s samim seboj. Skočimo na naslednjo sličico.

Nato dobimo koordinate sličice, tako da pokličemo njeno funkcijo coords pri (4) in shranimo rezultat v spremenljivko sprite_co.

Nato koda pri (5) preveri naslednje:

- Slika lika ni zadela vrha platna (spremenljivka top je še vedno True).
- Lik skače (vrednost y je manjša od 0).
- vrh lika je trčil s sličico v seznamu (z uporabo collided_top funkcije, ki smo jo ustvarili v poglavju 16).



Če so vsi ti pogoji resnični, želimo, da se lik začne spuščati, tako da pri (6) obrnemo vrednost y z uporabo minus znaka (-). Spremenljivka top se nastavi na False pri (7), ker ko lik zadane vrh, ni potrebno preverjati trka.

Trk od spodaj

Naslednji del zanke preveri, če je spodnji del našega lika ob kaj udaril:

```

if top and self.y < 0 and collided_top(co, sprite_co):
self.y = -self.y
top = False
(1) if bottom and self.y > 0 and collided_bottom(self.y, \
co, sprite_co):

```

```

(2)         self.y = sprite_co.y1 - co.y2
(3)         if self.y < 0:
(4)             self.y = 0
(5)         bottom = False
(6)         top = False

```

Obstajajo tri podobna preverjanja pri (1): če je spremenljivka `bottom` še nastavljena, če lik še pada (`y` je večji od 0) in če je spodnji del našega junaka udaril ob drugo sličico. Če vsi trije pogoji držijo, odštejemo spodnjo `y` vrednost (`y2`) lika od zgornje `y` vrednosti sličice (`y1`) pri (2). To se morda zdi čudno, zato pogledajmo, zakaj to naredimo.

Predstavljajte si, da je naš lik padel s platforme. Pada po zaslonu, vsakič za 4 pike, ko se izvrši `mainloop` in pride na primer do 3 pike nad platformo. Recimo, da je spodnji del lika (`y2`) na poziciji 57 in vrh platforme (`y1`) je na položaju 60. V tem primeru bi `collided_bottom` funkcija vrnila `True`, ker bo njena koda dodala vrednost `y` (kar je 4) spremenljivki `y2` lika, kar pomeni 61.

Vendar ne želimo, da bi se Mr. Stickman ustavil tako, ko se zazdi, da bo zadel platformo ali dno zaslona, ker bi to bilo tako, kot da bi bil z velikim odskokom in se ustavil v zraku, nekaj centimetrov nad tlemi. To je lahko zanimiv trik, vendar v naši igri ne bo videti v redu. Namesto tega, če liku odštejemo vrednost `y2` (od 57) od vrednosti `y1` platforme (od 60) dobimo 3, kolikor je do pravilnega pristanka na vrhu platforme.

Pri (3) se prepričamo, da izračun ne povzroči negativnega števila; če je, `y` nastavimo na 0 pri (4). (Če pustimo številko negativno, bi lik znova skočil, česar pa ne želimo v tej igri.)

Nazadnje nastavimo `top` in `bottom` vrednosti na `False`, tako ni več potrebe po preverjanju, če je lik trčil pri na vrhu ali dnu z drugo sličico.

Še enkrat preverimo `bottom`, da ugotovimo, če je lik padel s platforme. Tu je koda:

```

if self.y < 0:
    self.y = 0
    bottom = False
    top = False

if bottom and falling and self.y == 0 \
    and co.y2 < self.game.canvas_height \
    and collided_bottom(1, co, sprite_co):
    falling = False

```

Pet preverjanj mora biti resničnih, da bi spremenljivko `falling` nastavili na `False`:

- Še vedno moramo preveriti, če je `bottom` nastavljeno na `True`.
- Preveriti moramo, če naj bi lik še padal (`falling` je nastavljeno na `True`).
- Lik še ne pada (`y` je 0).
- Spodnji rob sličice ni zadel dna zaslona (je manj kot višina platna).
- Lik je zadela vrh platforme (`collided_bottom` vrne `True`).

Potem nastavimo `falling` spremenljivko na `False`.

Preverjanje levo in desno

Preverili smo, če lik zadane sličico spodaj ali zgoraj. Zdaj moramo preveriti, če je zadel levo ali desno stran s to kodo:

```
        if bottom and falling and self.y == 0 \
            and co.y2 < self.game.canvas_height \
            and collided_bottom(1, co, sprite_co):
            falling = False
(1)        if left and self.x < 0 and collided_left(co, sprite_co):
(2)            self.x = 0
(3)            left = False
(4)        if right and self.x > 0 and collided_right(co, sprite_co):
(5)            self.x = 0
(6)            right = False
```

Pri (1) pogledamo, če bi morali še vedno iskati trke levo (levo je še nastavljeno na True) in če se lik giblje v levo (x je manj kot 0). Preverjamo tudi, če lik trči s sličico z uporabo funkcije `collided_left`. Če so ti trije pogoji izpolnjeni, nastavimo x na 0 pri (2) (da se lik ustavi) in nastavimo left na False pri (3), da ni več potrebno preverjanje trkov na levi strani.

Koda je podobna za trke na desni, kot je prikazano pri (4). Nastavimo x na 0 pri (5) in right na False pri (6), da ustavimo preverjanje trkov na desni strani.

Zdaj bi morala s preverjanji trkov v vseh štirih smereh, naša koda za for zanko biti taka:

```
for sprite in self.game.sprites:
    if sprite == self:
        continue
    sprite_co = sprite.coords()
    if top and self.y < 0 and collided_top(co, sprite_co):
        self.y = -self.y
        top = False

    if bottom and self.y > 0 \
        and collided_bottom(self.y, co, sprite_co):
        self.y = sprite_co.y1 - co.y2
        if self.y < 0:
            self.y = 0
        bottom = False
        top = False

    if bottom and falling \
        and self.y == 0 \
        and co.y2 < self.game.canvas_height \
        and collided_bottom(1, co, sprite_co):
        falling = False

    if left and self.x < 0 and collided_left(co, sprite_co):
        self.x = 0
        left = False

    if right and self.x > 0 \
        and collided_right(co, sprite_co):
        self.x = 0
        right = False
```

Dodati moramo le še nekaj vrstic za funkcijo `move`, kot sledi:

```
    if right and self.x > 0 \
        and collided_right(co, sprite_co):
```

```

        self.x = 0
        right = False
(1)    if falling and bottom \
        and self.y == 0 and co.y2 < self.game.canvas_height:
(2)        self.y = 4
(3)    self.game.canvas.move(self.image, self.x, self.y)

```

Pri (1) preverimo, če sta spremenljivki `falling` in `bottom` nastavljeni na `True`. Če je tako, smo preverili vse sličice v seznamu brez trčenja na dno.

Končni pregled v tej vrstici ugotovi, če je spodnji del našega lika manjši od višine platna - to pomeni, da je nad tlemi (spodnji del platna). Če lik ni trčil z ničemer nad tlemi, stoji v zraku, zato bi moral začeti padati (z drugimi besedami, padel je s platforme). Da bi ga odstranili s konca katerekoli platforme, postavimo `y` na 4 pri (2).

Pri (3) premikamo sliko po zaslonu glede na vrednosti, ki smo jih določili v spremenljivkah `x` in `y`. Dejstvo, da smo preverili sličice glede trkov lahko pomeni, da smo nastavili obe spremenljivki na 0, ker je lik trčil na levi in z dnom. V tem primeru klic funkcije `move` ne bo naredil ničesar.

Morda je Mr. Stickman odšel z roba platforme. Če se to zgodi, bo `y` nastavljen na 4 in lik bo padel navzdol.

Huh, to je bila dolga funkcija!

Testiranje našega glavnega lika

Ko smo ustvarili razred `StickFigureSprite`, ga sedaj preizkusimo z dodajanjem naslednjih dveh vrstic tik pred klicem funkcije `mainloop`.

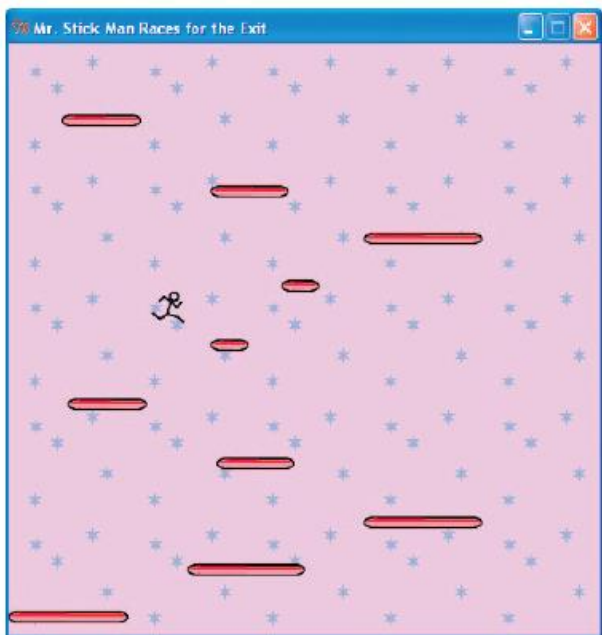
```

(1) sf = StickFigureSprite(g)
(2) g.sprites.append(sf)
    g.mainloop()

```

Pri (1) ustvarimo objekt `StickFigureSprite` in ga nastavimo kot spremenljivko `sf`. Kot pri platformah, dodamo novo spremenljivko na seznam sličic, shranjenih v objektu `game` pri (2).

Zdaj zaženite program. Ugotovili boste, da lahko Mr. Stickman teče, skoči s platforme na platformo in pade!



Vrata!

Edina stvar, ki nam še manjka, so izhodna vrata. Končali bomo z izdelavo razreda sprite za vrata, dodali kodo za odpiranje vrat in dodali našemu programu objekt door.

Ustvarjanje razreda DoorSprite

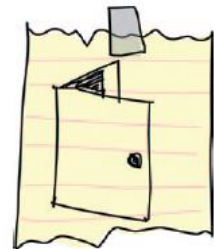
Uganili ste - ustvariti moramo še en razred: DoorSprite. Tukaj je začetek kode:

```
class DoorSprite(Sprite):
(1) def __init__(self, game, photo_image, x, y, width, height):
(2)     Sprite.__init__(self, game)
(3)     self.photo_image = photo_image
(4)     self.image = game.canvas.create_image(x, y, \
        image=self.photo_image, anchor='nw')
(5)     self.coordinates = Coords(x, y, x +(width / 2), y + height)
(6)     self.endgame = True
```

Kot je prikazano pri (1), ima funkcija `__init__` razreda DoorSprite parameter `self`, objekt `game`, objekt `photo_image`, `x` in `y` koordinate ter širino in višino slike.

Pri (2), pokličemo `__init__` kot v drugih naših razredih sprite.

Pri (3) shranimo parameter `photo_image` z uporabo objektne spremenljivke z istim imenom, kot smo storili s PlatformSprite. Ustvarimo prikaz slike s funkcijo `create_image` in shranimo identifikacijsko številko v `image` pri (4).



Pri (5) nastavimo koordinate DoorSprite na `x` in `y` parametra (ki postaneta položaja vrat `x1` in `y1`), nato pa izračunamo položaja `x2` in `y2`. Položaj `x2` izračunamo z dodajanjem polovice širine (spremenljivka `width` deljena z 2) parametru `x`. Na primer, če je `x=10` (`x1` koordinata je tudi 10), in širina je 40, bi bila koordinata `x2=30` (10 plus polovica 40).

Zakaj uporabiti ta izračun? Ker, za razliko od platforme, kjer želimo, da Mr. Stickman preneha teči takoj, ko trči s platformo, ga tu želimo ustaviti pred vrati. (Ne bo videti dobro, če se bo Mr. Stickman ustavil šele za vrati!) To boste videli v igri, ko boste odigrali in ga pripeljali do vrat.

Za razliko od položaja x1 je položaj y1 enostaven za izračun. Spremenljivki height dodamo le parameter y, in to je to.

Nazadnje, pri (6) nastavimo spremenljivko endgame na True. To pomeni, da se igra konča, ko glavni lik pride do vrat.

Ugotavljanje stanja vrat

Zdaj moramo spremeniti kodo v razredu StickFigureSprite funkcijo move, ki ugotavlja, kdaj lik trči na levi ali desni strani. Tukaj je prva sprememba:

```
if left and self.x < 0 and collided_left(co, sprite_co):
    self.x = 0
    left = False
    if sprite.endgame:
        self.game.running = False
```

Preverjamo, če ima sličica, s katero je lik trčil, spremenljivko endgame nastavljeno na True. Če je, nastavimo spremenljivko running na False in vse se ustavi - prišli smo do konca igre.

Enake vrstice bomo dodali h kodi, ki preverja trk na desni. Tukaj je koda:

```
if right and self.x > 0 and \
    collided_right(co, sprite_co):
    self.x = 0
    right = False
    if sprite.endgame:
        self.game.running = False
```

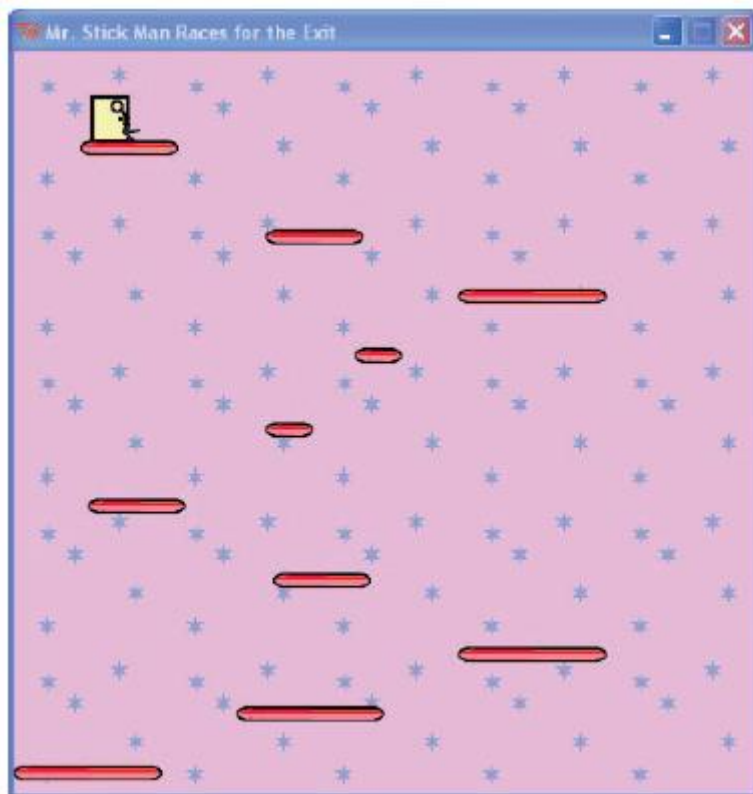
Dodajanje objekta door

Naš zadnji dodatek k igralni kodi je objekt door. To bomo dodali pred glavno zanko. Tik pred izdelavo objekta glavnega lika, ustvarili bomo objekt door in ga dodali na seznam sličic. Tukaj je koda:

```
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, PhotoImage(file="door1.png"), 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()
```

Ustvarimo objekt door z uporabo spremenljivke objekta game, g, ki ji sledi PhotoImage (slika vrat, ki smo jo ustvarili v poglavju 15). Parametre x in y smo nastavili na 45 in 30, da bi vrata postavili na platformo blizu vrha zaslona in nastavili width in height na 40 in 35. Objekt door dodamo na seznam sprites, tako kot pri vseh drugih sličicah v igri.

Rezultat lahko vidite, ko Mr. Stickman pride do vrat. Ustavi se pred vrati in ne zraven njih, kot je prikazano tukaj:



Končana igra

Celotne kode igre je zdaj nekaj več kot 200 vrstic. Spodaj je celotna koda igre. Če imate težave pri delovanju z vašo kodo, primerjanje vse funkcije (in vsak razred) v tem seznamu in pogledjte, kje ste se zmotili.

```

from tkinter import *
import random
import time

class Game:
    def __init__(self):
        self.tk = Tk()
        self.tk.title("Mr. Stickman drvi proti izhodu")
        self.tk.resizable(0, 0)
        self.tk.wm_attributes("-topmost", 1)
        self.canvas = Canvas(self.tk, width=500, height=500,
highlightthickness=0)
        self.canvas.pack()
        self.tk.update()
        self.canvas_height = 500
        self.canvas_width = 500
        self.bg = PhotoImage(file="background.png")
        w = self.bg.width()
        h = self.bg.height()
        for x in range(0, 5):
            for y in range(0, 5):
                self.canvas.create_image(x * w, y * h, image=self.bg,
anchor='nw')
        self.sprites = []
        self.running = True

```

```

def mainloop(self):
    while 1:
        if self.running == True:
            for sprite in self.sprites:
                sprite.move()
            self.tk.update_idletasks()
            self.tk.update()
            time.sleep(0.01)

class Coords:
    def __init__(self, x1=0, y1=0, x2=0, y2=0):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

def within_x(co1, co2):
    if (co1.x1 > co2.x1 and co1.x1 < co2.x2) \
        or (co1.x2 > co2.x1 and co1.x2 < co2.x2) \
        or (co2.x1 > co1.x1 and co2.x1 < co1.x2) \
        or (co2.x2 > co1.x1 and co2.x2 < co1.x1):
        return True
    else:
        return False

def within_y(co1, co2):
    if (co1.y1 > co2.y1 and co1.y1 < co2.y2) \
        or (co1.y2 > co2.y1 and co1.y2 < co2.y2) \
        or (co2.y1 > co1.y1 and co2.y1 < co1.y2) \
        or (co2.y2 > co1.y1 and co2.y2 < co1.y1):
        return True
    else:
        return False

def collided_left(co1, co2):
    if within_y(co1, co2):
        if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
            return True
    return False

def collided_right(co1, co2):
    if within_y(co1, co2):
        if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
            return True
    return False

def collided_top(co1, co2):
    if within_x(co1, co2):
        if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
            return True
    return False

def collided_bottom(y, co1, co2):
    if within_x(co1, co2):
        y_calc = co1.y2 + y
        if y_calc >= co2.y1 and y_calc <= co2.y2:
            return True
    return False

class Sprite:
    def __init__(self, game):

```

```

        self.game = game
        self.endgame = False
        self.coordinates = None
    def move(self):
        pass
    def coords(self):
        return self.coordinates

class PlatformSprite(Sprite):
    def __init__(self, game, photo_image, x, y, width, height):
        Sprite.__init__(self, game)
        self.photo_image = photo_image
        self.image = game.canvas.create_image(x, y, image=self.photo_image,
anchor='nw')
        self.coordinates = Coords(x, y, x + width, y + height)

class StickFigureSprite(Sprite):
    def __init__(self, game):
        Sprite.__init__(self, game)
        self.images_left = [
            PhotoImage(file="stickman-L1.png"),
            PhotoImage(file="stickman-L2.png"),
            PhotoImage(file="stickman-L3.png")
        ]
        self.images_right = [
            PhotoImage(file="stickman-R1.png"),
            PhotoImage(file="stickman-R2.png"),
            PhotoImage(file="stickman-R3.png")
        ]
        self.image = game.canvas.create_image(200, 470,
image=self.images_left[0], anchor='nw')
        self.x = -2
        self.y = 0
        self.current_image = 0
        self.current_image_add = 1
        self.jump_count = 0
        self.last_time = time.time()
        self.coordinates = Coords()
        game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        game.canvas.bind_all('<space>', self.jump)

    def turn_left(self, evt):
        if self.y == 0:
            self.x = -2

    def turn_right(self, evt):
        if self.y == 0:
            self.x = 2

    def jump(self, evt):
        if self.y == 0:
            self.y = -4
            self.jump_count = 0

    def animate(self):
        if self.x != 0 and self.y == 0:
            if time.time() - self.last_time > 0.1:
                self.last_time = time.time()
                self.current_image += self.current_image_add
                if self.current_image >= 2:

```

```

        self.current_image_add = -1
        if self.current_image <= 0:
            self.current_image_add = 1
    if self.x < 0:
        if self.y != 0:
            self.game.canvas.itemconfig(self.image,
image=self.images_left[2])
        else:
            self.game.canvas.itemconfig(self.image,
image=self.images_left[self.current_image])
    elif self.x > 0:
        if self.y != 0:
            self.game.canvas.itemconfig(self.image,
image=self.images_right[2])
        else:
            self.game.canvas.itemconfig(self.image,
image=self.images_right[self.current_image])

    def coords(self):
        xy = list(self.game.canvas.coords(self.image))
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + 27
        self.coordinates.y2 = xy[1] + 30
        return self.coordinates

    def move(self):
        self.animate()
        if self.y < 0:
            self.jump_count += 1
            if self.jump_count > 20:
                self.y = 4
        if self.y > 0:
            self.jump_count -= 1

        co = self.coords()
        left = True
        right = True
        top = True
        bottom = True
        falling = True

        if self.y > 0 and co.y2 >= self.game.canvas_height:
            self.y = 0
            bottom = False
        elif self.y < 0 and co.y1 <= 0:
            self.y = 0
            top = False

        if self.x > 0 and co.x2 >= self.game.canvas_width:
            self.x = 0
            right = False
        elif self.x < 0 and co.x1 <= 0:
            self.x = 0
            left = False

        for sprite in self.game.sprites:
            if sprite == self:
                continue
            sprite_co = sprite.coords()
            if top and self.y < 0 and collided_top(co, sprite_co):

```



```

        self.y = -self.y
        top = False

    if bottom and self.y > 0 and collided_bottom(self.y, co,
sprite_co):
        self.y = sprite_co.y1 - co.y2
        if self.y < 0:
            self.y = 0
        bottom = False
        top = False

    if bottom and falling and self.y == 0 and co.y2 <
self.game.canvas_height and collided_bottom(1, co, sprite_co):
        falling = False

    if left and self.x < 0 and collided_left(co, sprite_co):
        self.x = 0
        left = False
        if sprite.endgame:
            self.game.running = False

    if right and self.x > 0 and collided_right(co, sprite_co):
        self.x = 0
        right = False
        if sprite.endgame:
            self.game.running = False

    if falling and bottom and self.y == 0 and co.y2 <
self.game.canvas_height:
        self.y = 4

    self.game.canvas.move(self.image, self.x, self.y)

class DoorSprite(Sprite):
    def __init__(self, game, photo_image, x, y, width, height):
        Sprite.__init__(self, game)
        self.photo_image = photo_image
        self.image = game.canvas.create_image(x, y, image=self.photo_image,
anchor='nw')
        self.coordinates = Coords(x, y, x +(width / 2), y + height)
        self.endgame = True

g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.png"), 0, 480,
100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.png"), 150, 440,
100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.png"), 300, 400,
100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.png"), 300, 160,
100, 10)
platform5 = PlatformSprite(g, PhotoImage(file="platform2.png"), 175, 350,
66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.png"), 50, 300,
66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.png"), 170, 120,
66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.png"), 45, 60, 66,
10)

```

```

platform9 = PlatformSprite(g, PhotoImage(file="platform3.png"), 170, 250,
32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.png"), 230, 200,
32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, PhotoImage(file="door1.png"), 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()

```

Kaj ste se naučili

V tem poglavju smo zaključili igro Mr. Stickman drvi proti izhodu. Ustvarili smo razred za naš glavnega lika in napisali funkcije, da bi ga premikali po zaslonu in ga animirali pri premikanju (spreminjanje iz ene slike v drugo, da bi dalo iluzijo teka). Uporabili smo osnovno ugotavljanje trkov, da bi vedeli, kdaj je zadel levo ali desno stran platna ali trčil z drugimi sličicami, kot so platforme ali vrata. Dodali smo tudi ugotavljanje dotika vrha ali dna zaslona ter padca s platforme. Ugotoviti smo morali tudi kdaj Mr. Stickman pride do vrat in se igra konča.



Vaje

Za izboljšanje igre je več možnosti. V tem trenutku je to zelo preprosto, zato lahko dodamo kodo, da postane igra profesionalna in zanimiva za igro. Poskusite dodati naslednje lastnosti in nato preverite svojo kodo na <https://nostarch.com/pythonforkids>.

1: "Zmagal si!"

Kot besedilo »Game Over« v igri Odbij!, ki smo jo končali v poglavju 14, dodajte besedilo »Zmagal si!«, ko lik doseže vrata, tako da igralci vidijo, da so zmagali.

2: Animacija vrat

V poglavju 15 smo ustvarili dve sliki za vrata: ena odprta in ena zaprta. Ko g. Stickman pride do vrat, naj se slika vrat spremeni v odprta vrata, Mr. Stickman bi moral izginiti in slika vrat se vrne na zaprta vrata. To bo izgledalo kot, da se za Mr. Stickmanom zaprejo vrata. To lahko naredite s spremembo razreda DoorSprite in razreda StickFigureSprite.

3: Premične platforme

Poskusite dodati nov razred imenovan `MovingPlatformSprite`. Te platforme se morajo premikati iz ene strani na drugo, zaradi česar je Mr. Stickmanu težje priti do vrha.

Zaključna beseda - kako naprej

V tem sprehodu po Pythonu ste se naučili nekaj osnovnih programskih postopkov, da boste sedaj veliko lažje delali z drugimi programskimi jeziki. Medtem ko je Python neverjetno uporaben, pa ni rečeno, da je en jezik vedno najboljše orodje za vsako nalogo. Zato se ne bojte preizkusite druge načine programiranja vašega računalnika. Tukaj si bomo ogledali nekaj možnosti za programiranje iger in grafike ter nato še nekaj najbolj pogosto uporabljenih programskih jezikov.

Igre in grafično programiranje

Če se želite več ukvarjati z igrkami ali grafičnim programiranjem, imate odprtih veliko možnosti. Na primer:

- BlitzBasic (<http://www.blitzbasic.com/>), ki uporablja posebno različico programskega jezika BASIC, ki je zasnovana posebej za igre
- Adobe Flash, vrsta animacijske programske opreme, ki je zasnovana, da teče v brskalniku in ima lasten programski jezik ActionScript (<http://www.adobe.com/devnet/actionscript.html>)
- Alice (<http://www.alice.org/>), 3D programsko okolje (samo za Microsoft Windows in Mac OS X)
- Scratch (<http://scratch.mit.edu/>), orodje za razvoj iger
- Unity3D (<http://unity3d.com/>), še eno orodje za ustvarjanje iger

Spletno iskanje bo odkrilo veliko virov za pomoč pri začetku s katero od od teh možnosti.

Po drugi strani pa, če bi se radi še naprej igrali s Pythonom, lahko uporabite PyGame, modul Python, ki je namenjen razvoju iger. Raziščimo to možnost.

PyGame

PyGame Reloaded (pgreloaded ali pygame2) je različica PyGame ki deluje s Python 3 (starejše različice delujejo samo s Python 2). Dobra možnost za začetek branja je tutorial, ki je na voljo na <https://www.pygame.org/wiki/pgreloaded>.

Pisanje igre s PyGame je malo bolj zapleteno kot z uporabo tkinterja. Na primer, v poglavju 12 smo prikazali sliko z uporabo tkinter s to kodo:

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
myimage = PhotoImage(file='c:\\test.png')
canvas.create_image(0, 0, anchor=NW, image=myimage)
```

Enako kodo s PyGame (naložite datoteko .bmp in ne datoteka .png) bi bila videti takole:

```
import sys
import time
import pygame
import pygame.sdl.constants as constants
import pygame.sdl.image as image
import pygame.sdl.video as video
(1) video.init()
(2) img = image.load_bmp("c:\\test.bmp")
```

```

(3) screen = video.set_mode(img.width, img.height)
(4) screen.fill(pygame2.Color(255, 255, 255))
(5) screen.blit(img, (0, 0))
(6) screen.flip()
(7) time.sleep(10)
(8) video.quit()

```

Po uvozu pygame2 modulov pokličemo funkcijo `init` na video modulu PyGame (1), ki je podoben klicu ustvarjanja platna in nato `pack` v primeru `tkinter`. BMP sliko naložimo s funkcijo `load_bmp` pri (2), nato pa ustvarimo zaslonski objekt z uporabo funkcije `set_mode`, ki sprejme širino in višino naložene slike kot parametre (3). Z naslednjo (neobvezno) vrstico, obrišemo zaslon tako, da ga napolnimo z belo barvo pri (4) in nato uporabimo funkcijo `Blit` zaslonskega objekta, da prikažemo sliko pri (5). Parametri za to funkcijo so objekt `img` in `Tuple`, ki vsebuje položaj, kjer želimo prikazati sliko (0 desno, 0 navzdol).

PyGame uporablja `off-screen buffer` (vmesni pomnilnik znan tudi kot `double buffer`). `Off-screen buffer` je tehnika, ki se uporablja za risanje grafike na območju pomnilnika računalnika, ki ni viden, in ga nato v celoti prekopira v vidni zaslon. Vmesnik `off-screen` zmanjša učinek utripanja, če se na zaslonu izrisuje veliko različnih predmetov. Kopiranje iz `off-screen` medpomnilnika na vidni zaslon se izvede z ukazom `flip` pri (6).

Na koncu še počakamo 10 sekund pri (7), ker se za razliko od `tkinter` platna, zaslon takoj zapre, če ga ne bomo ustavili tako. Pri (8) počistimo za seboj `video.quit`, tako da bo PyGame pravilno zaprt. PyGame omogoča še veliko več, tule je le primer, da vidite, kako izgleda.

Programski jeziki

Če vas zanimajo drugi programski jeziki, nekateri trenutno priljubljeni so Java, C / C ++, C #, PHP, Objective-C, Perl, Ruby in JavaScript. Na kratko si bomo ogledali te jezike in pogledali, kako bi izgledal program "Pozdravljen svet" (kot v Pythonu v poglavju 1). Upoštevajte, da noben teh jezikov ni posebej namenjen začetnikom in večina se bistveno razlikuje od Pythona.

Java

Java (<http://www.oracle.com/technetwork/java/index.html>) je zmerno zapleten programski jezik z velikim vgrajenimi knjižnicami modulov (imenovanih paketov). Veliko brezplačne dokumentacije je na voljo na spletu. V večini operacijskih sistemov lahko uporabljate Java. Java je tudi jezik, ki se uporablja v mobilnih telefonih Android.

Tukaj je primer Hello World v Java:

```

public class HelloWorld {
    public static final void main(String[] args) {
        System.out.println("Hello World");
    }
}

```

C/C++

C (<http://www.cprogramming.com/>) in C++ sta zapletena programska jezika, ki se uporabljata za vse operacijske sisteme. Našli boste brezplačne in komercialne različice. Oba jezika (čeprav morda C++

več kot C) imata strmi učni krivulji. Ugotovili boste, da morate ročno kodirati nekatere funkcije, ki jih Python že ponuja (npr. uporabo pomnilnika za shranjevanje objektov). Veliko komercialnih iger in konzolnih iger je programiranih v kakšni obliki C ali C++.

Tukaj je primer Pozdravljen svet v C:

```
#include <stdio.h>
int main()
{
    printf("Pozdravljen svet \n");
}
```

Primer v C++ bi bil lahko takšen:

```
#include <iostream>
int main()
{
    std::cout << "Pozdravljen svet\n";
    return 0;
}
```

C#

C# (<http://msdn.microsoft.com/en-us/vstudio/hh388566/>), izgovori se "Si šarp" je zmerno zapleten programski jezik za Windows, ki je zelo podoben Javi. Malo je lažji kot C in C++.

Tukaj je primer Pozdravljen svet v C#:

```
public class Hello
{
    public static void Main()
    {
        System.Console.WriteLine("Pozdravljen svet");
    }
}
```

PHP

PHP (<http://www.php.net/>) je programski jezik za spletne strani. Potrebovali boste spletni strežnik (programska oprema, ki se uporablja za objavljanje spletnih strani v spletnih brskalnikih) z nameščenim PHP. Vsa zahtevana programska oprema je prosto dostopna za vse glavne operacije sisteme. Za delo s PHP-jem se boste morali naučiti HTML (preprost jezik za izdelavo spletnih strani). Brezplačne PHP vodiče najdete na <http://php.net/manual/en/tutorial.php> in HTML tutorial na <http://www.w3schools.com/html>.

Stran v obliki HTML, ki prikazuje "Pozdravljen svet" v brskalniku, izgleda tako:

```
<html>
  <body>
    <p> Pozdravljen svet </p>
  </body>
</html>
```

Stran PHP, ki bo naredila isto stvar, bi lahko izgledala takole:

```
<? php
echo "Pozdravljeni svet\n";
?>
```

Objective-C

Objective-C (<http://classroomm.com/objective-c/>) je zelo podoben C (dejansko je razširitev programskega jezika C) in se najpogosteje uporabljajo na računalnikih Apple. To je programski jezik za iPhone in iPad.

Tukaj je primer Pozdravljen svet v Objective-C:

```
#import <Foundation/Foundation.h>
int main(int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Hello World");
    [pool drain];
    return 0;
}
```

Perl

Programski jezik Perl (<http://www.perl.org/>) je na voljo brezplačno za vse večje operacijske sisteme. Običajno se uporablja za razvijanje spletnih strani (podobno kot PHP).

Tukaj je primer Hello World v Perlu:

```
print("Pozdravljeni svet\n");
```

Ruby

Ruby (<http://www.ruby-lang.org/>) je brezplačen programski jezik na voljo v vseh večjih operacijskih sistemih. Večinoma se uporablja za ustvarjanje spletnih strani, posebej z uporabo framework Ruby on Rails. (Framework je zbirka knjižnic, ki podpirajo razvoj posebnih vrst aplikacij.)

Tukaj je primer Pozdravljen svet v Rubyju:

```
puts "Pozdravljen svet"
```

JavaScript

JavaScript (<https://developer.mozilla.org/en-US/docs/Web/JavaScript>) je programski jezik, ki se običajno uporablja znotraj spletnih strani, vse bolj se uporablja tudi za igranje iger. Sintaksa je v osnovi podobna Javi, vendar je morda malo lažje začeti z JavaScript-om. (Lahko ustvarite preprosto HTML stran, ki vsebuje JavaScript program in ga poženete v brskalniku brez potrebe po lupini, ukazni vrstici ali čem podobnem.) Dober kraj za začetek učenja JavaScript je lahko Codecademy na <http://www.codecademy.com/>.

Primer "Pozdravljen svet" v JavaScriptu je odvisen od tega ali ga poganjate v brskalniku ali v lupini. V lupini je takšen:

```
print('Pozdravljen svet');
```

V brskalniku je lahko videti tako:

```
<html>
  <body>
    <script type="text/javascript">
      alert("Pozdravljen svet");
    </script>
  </body>
</html>
```

Zaključna beseda

Ne glede na to, ali se držite Pythona ali se odločite, da preizkusite drugo programsko orodje (in obstaja jih veliko več, kot je navedenih tukaj), morate še vedno uporabljati postopke, ki ste jih odkrili v tej knjigi. Tudi, če ne nadaljujete z računalniškim programiranjem, vam bo razumevanje nekaterih temeljnih idej pomagalo pri raznih dejavnostih, bodisi v šoli ali kasneje pri delu.

Vso srečo in uživajte v programiranju!



Dodatek - rezervirane besede v Pythonu

Ključne besede v Pythonu (in v večini programskih jezikov) so besede, ki imajo poseben pomen. Uporabljajo se kot del samega programskega jezika, zato se ne smejo uporabljati za nič drugega. Če na primer skušate uporabiti ključne besede kot spremenljivke ali jih uporabite na napačen način, boste dobili iz konzole Python čudno (včasih smešno, včasih zmedeno) sporočilo napakah.

Tu opisujemo vse ključne besede v Pythonu.

and

Ključna beseda `and` se uporablja za združitev dveh izrazov v izjavi (kot v `if` stavku), kot bi rekli, da sta oba izraza izpolnjena. Tukaj je primer:

```
if starost > 10 and starost < 20:
    print('Pozor, najstnik!!!!')
```

Ta koda pomeni, da mora biti vrednost spremenljivke `starost` več kot 10 in manj kot 20, da se sporočilo natisne.

as

Ključno besedo, ki jo lahko uporabite, če želite drugo ime za uvožen modul. Recimo, da ste imeli modul z zelo dolgim imenom:

```
i_am_a_python_module_that_is_not_very_useful
```

Bilo bi zelo neprijetno, da bi morali vnesti to ime modula vsakič, ko ga želite uporabiti:

```
import i_am_a_python_module_that_is_not_very_useful
i_am_a_python_module_that_is_not_very_useful.do_something()
I have done something that is not useful.
i_am_a_python_module_that_is_not_very_useful.do_something_else()
I have done something else that is not useful!!
```

Namesto tega lahko modulu pri uvozu daste novo, krajše ime in potem preprosto uporabite to novo ime (podobno kot vzdevek), in sicer:

```
import i_am_a_python_module_that_is_not_very_useful as notuseful
notuseful.do_something()
I have done something that is not useful.
notuseful.do_something_else()
I have done something else that is not useful!!
```

assert

`Assert` je ključna beseda, ki pravi, da mora biti koda resnična. To je drug način za lovljenje napak in težav s kodo, običajno v bolj naprednih programih (zato je ne uporabljamo v tej knjigi). Tukaj je preprosta trditev:

```
>>> mynumber = 10
>>> assert mynumber < 5
```

```
Traceback (most recent call last):
File "<pyshell#1>", line 1, in <module>
assert a < 5
AssertionError
```

V tem primeru trdimo, da je vrednost spremenljivke mynumber manj kot 5. Ni, zato Python prikaže napako (imenovano AssertionError).

break

Ključna beseda break se uporablja za ustavitev nekaterih kod. Uporabite jo lahko znotraj zanke, kot je ta:

```
starost = 10
for x in range(1, 100):
    print('štejem %s' % x)
    if x == starost:
        print('konec štetja')
        break
```

Ker je spremenljiva starost nastavljena na 10, bo ta koda izpisala:

```
štejem 1
štejem 2
štejem 3
štejem 4
štejem 5
štejem 6
štejem 7
štejem 8
štejem 9
štejem 10
konec štetja
```

Ko vrednost spremenljivke x doseže 10, koda natisne besedilo "konec štetja" in nato zaključi zanko.

class

Ključna beseda Class se uporablja za določitev vrste objekta, kot je vozilo, žival ali oseba. Razredi imajo lahko funkcijo, imenovano `__init__`, ki se uporablja za nastavljanje lastnosti in nastavitev, ki jih objekt potrebuje, ko je ustvarjen. Objekt razreda Avto, bi lahko imel nastavljivo barvo:

```
class Avto:
    def __init__(self, barva):
        self.barva = barva

avto1 = Avto('rdeča')
avto2 = Avto('modra')
print(avto1.barva)
rdeča
print(avto2.barva)
modra
```

continue

Ključna beseda `continue` je način »preskakovanja« na naslednji element v zanki, tako da se preostala koda v zanki ne izvaja. Za razliko od `break` ne skočimo iz zanke, ampak samo nadaljujemo z naslednjim elementom. Na primer, če bi imeli seznam predmetov in bi želeli preskočiti elemente, ki se začnejo z `b`, bi lahko uporabili naslednjo kodo:

```
(1) >>> my_items = ['apple', 'aardvark', 'banana', 'badger', 'clementine',  
'camel']  
(2) >>> for item in my_items:  
(3)         if item.startswith('b'):  
(4)             continue  
(5)             print(item)  
apple  
aardvark  
clementine  
camel
```

Ustvarimo naš seznam predmetov pri (1), nato pa uporabimo `for` zanko za sprehod po elementih pri (2). Če se predmet začne s črko `b` pri (3), nadaljujemo z naslednjim elementom pri (4). V nasprotnem primeru pri (5) natisnemo element.

def

Ključna beseda `def` se uporablja za definiranje funkcije. Na primer, ustvarite funkcijo za pretvorbo več let v enakovredno število minut:

```
>>> def minute(let):  
        return let * 365 * 24 * 60  
>>> minute(10)  
5256000
```

del

Funkcija `del` se uporablja za odstranitev nečesa. Na primer, če imate seznam stvari, ki si jih želite za svoj rojstni dan v svojem dnevniku, potem se pri enem premislite in ga prečrtate ter dodate nekaj drugega:

daljinsko voden avto

novo kolo

računalniška igra

robot

V Pythonu bi prvotni seznam izgledal takole:

```
zelim_si = ["daljinsko voden avto", "novo kolo", "računalniška  
igra"]
```

Računalniško igro lahko odstranite z `del` in indeksom predmeta, ki ga želite izbrisati. Nato lahko dodate nov element s funkcijo `add`:

```
del zelim_si[2]  
zelim_si.append("robot")
```

In nato natisnite nov seznam:

```
print(zelim_si)
["daljinsko voden avto", "novo kolo", "robot"]
```

elif

Ključna beseda elif se uporablja kot del stavka if. Primer si oglejte pri opisu ključne besede if.

else

Ključna beseda else se uporablja kot del stavka if. Primer si oglejte pri opisu ključne besede if.

except

Ključna beseda except se uporablja za lovljenje napak v kodi. To običajno uporabljamo v precej zapletenih programih, zato je ne uporabljamo v tej knjigi.

finally

Ključna beseda finally se uporablja, da se v primeru napake, izvede določena koda (ponavadi se počisti nered, ki jo del kode naredi). Ta ključna beseda v tej knjigi ni uporabljena, ker je za bolj zahtevne programe.

for

Ključno besedo for se uporablja za zanke, ki se morajo izvesti določeno število krat. Tukaj je primer:

```
for x in range(0, 5):
    print('x je %s' % x)
```

Ta for zanka izvede blok kode petkrat, kar ima za posledico naslednji rezultat:

```
x je 0
x je 1
x je 2
x je 3
x je 4
```

from

Ko uvozite modul, lahko uvozite le del, ki ga potrebujete z uporabo ključne besede from. Na primer, v 4. poglavju je bil predstavljen modul turtle z razredom Pen, ki smo ga uporabljali za objekt Pen (platno, na katerem se premika želva). Tako smo uvozili celoten modul turtle in nato uporabili razred Pen:

```
import turtle
t = turtle.Pen()
```

Lahko uvozimo le razred Pen in ga potem uporabimo neposredno (brez sklicevanja na modul turtle):

```
from turtle import Pen
t = Pen()
```

Ob pregledu programa si lahko za uvoz na začetku programa pripravite le funkcije in razrede, ki jih uporabljate (nekateri moduli so lahko izredno obsežni). Če se odločite za to, ne boste mogli uporabljati delov modulov, ki jih niste uvozili.

Na primer, časovni modul ima funkcije, imenovane localtime in gmtime. Če uvozite samo localtime in poskusite uporabljati gmtime, boste dobili napako:

```
>>> from time import localtime
>>> print(localtime())
(2007, 1, 30, 20, 53, 42, 1, 30, 0)
>>> print(gmtime())
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'gmtime' is not defined
```

Sporočilo o napaki 'gmtime' is not defined pomeni, da Python ne ve ničesar o funkciji gmtime, ker je niste uvozili.

Če v določenem modulu obstaja več funkcij, ki jih želite uporabiti in se ne želite povezati z njimi z uporabo imena modula (na primer, time.localtime ali time.gmtime), lahko uvozite vse v modulu z zvezdico (*), tako:

```
>>> from time import *
>>> print(localtime())
(2007, 1, 30, 20, 57, 7, 1, 30, 0)
>>> print(gmtime())
(2007, 1, 30, 13, 57, 9, 1, 30, 0)
```

To uvozi vse funkcije iz časovnega modula in kličete lahko le imena funkcij brez imen modulov.

global

Ideja o območju uporabe spremenljivk je bila predstavljena v poglavju 7. Območje se nanaša na *vidnost* spremenljivke. Če je spremenljivka določena zunaj funkcije, je ponavadi vidna tudi znotraj funkcije. Po drugi strani pa, če je spremenljivka definirana znotraj funkcije, je običajno ni mogoče videti zunaj te funkcije. Ključna beseda global je ena od izjem tega pravila. Spremenljivko, ki je definirana kot globalna je mogoče videti povsod. Tukaj je primer:

```
>>> def test():
    global a
    a = 1
    b = 2
```

Kaj mislite se bo zgodilo, ko pokličete print (a) in potem print (b), po klicu funkcije test? Prvi bo deloval, drugi pa bo prikazal sporočilo o napaki:

```
>>> test()
>>> print(a)
1
>>> print(b)
```

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined

Spremenljivka a je bila spremenjena v globalno znotraj funkcije, tako da je vidna, tudi ko je funkcija končana. Spremenljivka b je vidna le znotraj funkcije. (Morate uporabiti ključno besedo global pred nastavljanjem vrednosti vaše spremenljivke.)

if

If je ključna beseda uporabljena za odločanje na podlagi pogojev. To lahko uporabite tudi z drugimi ključnimi besedami else in elif (drugače, če). If stavek je na primer: "Če je nekaj res, potem opravi nekaj." Tu je primer:

```
(1) if cena_igrace > 1000:  
(2)     print('Igrača je predraga')  
(3) elif cena_igrace > 100:  
(4)     print('Ta igrača je draga')  
(5) else:  
(6)     print('Lahko si privoščim to igračo')
```

Ta if stavek pravi, da če je cena igrače več kot 1000 pri (1), prikaže sporočilo, da je predraga pri (2); drugače, če je cena igrače več kot 100 pri (3), prikažite sporočilo, da je draga pri (4). Če nobeden od teh pogojev ni izpolnjen pri (5), bi moralo prikazati sporočilo »Lahko si privoščim to igračo« pri (6).

import

Ključna beseda import pove Pythonu naj naloži modul, da se ga lahko uporabi. Na primer, naslednja koda pove, da Python uporablja modul sys:

```
import sys
```

in

Ključna beseda in se uporablja v izrazih, če želite vedeti, ali je določen element v zbirki. Na primer, ali je številka 1 v seznamu številok?

```
>>> if 1 in [1,2,3,4]:  
>>>     print('število je v seznamu')  
število je v seznamu
```

Tukaj vidimo, kako ugotoviti, ali so hlače na seznamu oblačil:

```
>>> seznam_obleke = ['kratke hlače', 'spodnje hlače', 'boksarice', 'dolge  
hlače']  
>>> if "hlače" in seznam_obleke:  
     print("hlače so na seznamu")  
else:  
     print("hlače niso na seznamu")  
hlače niso na seznamu
```

is

Ključna beseda is je nekoliko podobna operatorju enakosti (==), če želimo povedati, da sta dve stvari enaki (na primer 10 == 10 je resnično, in 10 == 11 je napačna). Vendar pa obstaja temeljna razlika

med is in ==. Če primerjate dve stvari, == lahko vrne True, medtem ko is vrne False (čeprav menite, da so stvari enake). To je napredni programski koncept zato se mi v tej knjigi držimo uporabe ==.

lambda

Ključna beseda lambda se uporablja za ustvarjanje anonimnih ali inline funkcij. Ta ključna beseda se uporablja v naprednejših programih in mi o njej v tej knjigi ne razpravljamo.

not

Če je nekaj res, potem ključna beseda not naredi izjavo napačno. Na primer, če ustvarimo spremenljivko x in jo nastavimo na vrednost True, nato pa natisnemo vrednost te spremenljivke z uporabo not, dobimo naslednji rezultat:

```
>>> x = True
>>> print(not x)
False
```

To se ne zdi zelo koristno, dokler ne začnete uporabljati ključne besede not v izjavah. Na primer, če želimo ugotoviti, ali predmet ni na seznamu:

```
>>> seznam_obleke = ['kratke hlače', 'spodnje hlače', 'boksarice', 'dolge hlače']
>>> if 'hlače' not in seznam_obleke:
    print('Res morate kupiti ene hlače')
Res morate kupiti ene hlače
```

or

Ključna beseda or se uporablja za združitev dveh pogojev v izjavi (v if stavkih), da bi rekli, da je vsaj eden od pogojev resničen. Tukaj je primer:

```
if dino == 'Tyrannosaurus' or dino == 'Allosaurus':
    print('Mesojedci')
elif dino == 'Ankylosaurus' or dino == 'Apatosaurus':
    print('Rastlinojedci')
```

V tem primeru, če spremenljivka dino vsebuje Tyrannosaurus ali Allosaurus, program natisne Mesojedci, sicer če je Ankylosaurus ali Apatosaurus, program natisne Rastlinojedci.

pass

Včasih, ko razvijate program, želite napisati le del kode, da preizkusite stvari. Težava je v tem, da ne morete imeti izjave if brez bloka kode, ki jo je treba zagnati, če je izraz v stavku if resničen. Prav tako ne morete imeti for zanke brez bloka kode. Naslednja koda deluje čisto v redu:

```
>>> starost = 15
>>> if starost > 10:
    print('starejši od 10')
starejši od 10
```

Ampak, če ne dodate kode za if stavek, boste prejeli sporočilo o napaki:

```
>>> starost = 15
>>> if starost > 10:
File "<stdin>", line 2
```

^

IndentationError: expected an indented block

To je sporočilo o napaki, ki ga Python prikaže, ko želi imeti blok kode po izjavi neke vrste (v IDLE tega nite ne boste mogli vnesti). V takih primerih lahko uporabite ključno besedo `pass`, da napišete izjavo, vendar ne zagotovite bloka kode.

Recimo, da želite ustvariti `for` zanko z `if` stavkom v njej. Morda se še niste odločili, kaj vnesti v `if` stavek. Uporabite lahko `pass` in koda še vedno deluje (tudi če to ne naredi ravno tistega, kar hočete).

Tukaj je naša če izjava, tokrat z uporabo ključne besede `pass`:

```
>>> starost = 15
>>> if starost > 10:
    pass
```

Naslednja koda prikazuje drugo uporabo ključne besede `pass`:

```
>>> for x in range(0, 7):
>>>     print('x je %s' % x)
>>>     if x == 4:
>>>         pass
x je 0
x je 1
x je 2
x je 3
x je 4
x je 5
x je 6
```

Python še vedno vsakič preverja, če spremenljivka `x` vsebuje vrednost 4, ko izvede blok kode v zanki, vendar ne stori ničesar, zato bo natisnil vsako številko v razpon od 0 do 7.

Kasneje lahko dodate kodo v blok za `if` stavek in zamenjati `pass` z nečim drugim, na primer `break`:

```
>>> for x in range(0, 7):
>>>     print('x je %s' % x)
>>>     if x == 4:
>>>         break
x je 0
x je 1
x je 2
x je 3
x je 4
```

Ključna beseda `pass` se najpogosteje uporablja, ko ustvarite funkcijo, vendar še ne želite napisati kode za to funkcijo.

raise

Ključno besedo `raise` uporabite za povzročitev napake. To se zdi čudno, ampak v naprednem programiranju je lahko dejansko zelo koristno. (Tega ne uporabljamo v tej knjigi.)

return

Ključna beseda `return` se uporablja za vračilo rezultata iz funkcije. Ustvarite lahko funkcijo za izračun števila sekund od svojega rojstnega dne:

```
def starost_v_sekundah(starost_v_letih):  
    return starost_v_letih * 365 * 24 * 60 * 60
```

Ko pokličete to funkcijo, lahko dodelite vrnjeno vrednost drugi spremenljivki ali jo natisnete:

```
>>> sekund = starost_v_sekundah(9)  
>>> print(sekund)  
283824000  
>>> print(starost_v_sekundah(12))  
378432000
```

try

Ključna beseda `try` začne blok kode, ki se konča s ključnima besedama `except` in `finally`. Skupaj se `try/except/finally` uporabljajo za obvladovanje napak v programu, na primer, da se prepričate, da program prikaže koristno sporočilo uporabniku, namesto neprijaznega Pythonovega sporočila o napaki. Te ključne besede se ne uporabljajo v tej knjigi.

while

Ključna beseda `while` je podobna `for`, razen da se zanka ponavlja, dokler je pogoj resničen. Pri uporabi `while` je treba biti previden, da se zanka ne ponavlja neskončno (to se imenuje neskončna zanka).

Tukaj je primer:

```
>>> x = 1  
>>> while x == 1:  
    print('živijo')
```

Če boste zagnali to kodo, se bo stalno ponavljala ali vsaj do zaprtja okna Python ali pritiska `ctrl-C`, da prekinete. Naslednja koda pa bo natisnila "živijo" devetkrat (vsakič, ko dodate 1 spremenljivki `x`, dokler `x` ni 10 ali več).

```
>>> x = 1  
>>> while x < 10:  
    print('živijo')  
    x = x + 1
```

with

Ključna beseda `with` se uporablja pri objektih na podoben način kot `try` in `finally`. Ta ključna beseda ni uporabljena v tej knjigi.

yield

Ključna beseda `yield` je podobna `return`, le da se uporablja s posebnim razredom objekta, imenovanim generator. Generatorji ustvarjajo vrednosti sproti (kar bi drugače rekli, da ustvari vrednosti na zahtevo), tako da se v tem pogledu tako obnaša funkcija `range`. Ta ključna beseda se ne uporablja v tej knjigi.

Slovar

Ko se prvič lotite programiranja, boste naleteli na nove izraze, ki vam nič ne pomenijo. To pomanjkanje razumevanja lahko upočasnijo napredovanje. Vendar obstaja preprosta rešitev za ta problem!

Ta slovarček vam bo pomagal v tistih trenutkih, ko vas nova beseda ali izraz ustavi. Tu boste našli definicije številnih programskih izrazov uporabljenih v tej knjigi. Če naletite na besedo, ki je ne razumete, le pogledajte sem.

animacija Proces prikaza zaporedja slik dovolj hitro, da izgleda, da se nekaj giblje.

blok Skupina računalniških izjav v programu.

Boolean Tip oziroma vrednost podatka, ki je lahko resnična ali napačna. (v Pythonu, True ali False, z velikim T in F.)

dimenzije V kontekstu grafičnega programiranja dvo- ali tridimenzionalno se nanaša na prikazovanje slik na računalniškem monitorju. Dvodimenzionalna (2D) grafika so ravne slike na zaslonu in imajo širino in višino, podobno kot stare risanke na televiziji. Tridimenzionalna (3D) grafika so slike na zaslonu, ki imajo širino, višino in videz globine - vrsta grafike, kjer liki v računalniški igri izgledajo bolj naravno.

dogodek (event) Nekaj, kar se zgodi, ko program teče. Lahko je to klik na miško, pritisk na tipko, dogodek ob določenem času...

funkcija (function) Ukaz v programskem jeziku, ki je običajno zbirka izjav, ki nekaj izvajajo.

horizontalno (vodoravno) Leva in desna smer na zaslonu (x koordinate).

identifikator Številka, ki v programu nekaj označuje. Na primer v Pythonovem modulu tkinter se identifikator uporablja za oštevilčenje elementov na platnu.

imenik (directory) ali mapa Lokacija skupine datotek na pogonu v vašem računalnik.

inicijalizacija Se nanaša na nastavitve začetnega stanja objekta (to je nastavitve spremenljivk v objektu, ko je prvič ustvarjen).

instanca Instanca razreda - z drugimi besedami: objekt.

izjema (exception) Vrsta napake, ki se lahko pojavi med delovanjem programa.

izvršitev (execute) Zagon nekaj kode, programčka, funkcije ali dela kode

klic (call) Zažene kodo v funkciji. Ko uporabimo funkcijo, pravimo do jo "kličemo".

klik pritisk na enega od gumbov miške, da pritisnete gumb na zaslonu, izberete možnost menija in tako naprej.

ključna beseda (keyword) Posebna beseda, ki jo uporablja programski jezik. Ključne besede imenujemo tudi rezervirane besede, kar v bistvu pomeni, da jih ne morete uporabiti za kaj drugega (na primer, ne morete uporabite ključno besedo za ime spremenljivke).

koordinate Položaj piksla na zaslonu. To je običajno opisano kot število pikslov na zaslonu v desno (x) in število pikslov navzdol (y).

lupina (shell) Pri uporabi računalnikov je lupina vmesnik ukazne vrstice. V tej knjigi se "lupina Python" nanaša na IDLE.

modul Skupina funkcij in spremenljivk.

namestitev (installation) Postopek kopiranja datotek programske opreme na svoj računalnik, tako da je aplikacija na voljo za uporabo.

napaka (error) Ko gre kaj narobe s programom v računalniku, je to napaka. Ko programirate s Python-om, lahko pride do raznovrstnih sporočil kot odgovor na napako. Če nepravilno vnesete kodo, boste morda videli IndentationError.

navpično (vertical) smer navzgor in navzdol na zaslonu (y koordinata).

niz (string) Zbirka alfanumeričnih znakov (črk, števil, ločil in presledkov).

null Odsotnost vrednosti (v Python, imenovana tudi None).

okvir (frame) Ena iz niza slik, ki sestavljajo animacijo.

objekt (object) Specifičen primer razreda. Ko ustvarite objekt razreda, Python rezervira nekaj pomnilnika vašega računalnika za podatke člana tega razreda.

območje uporabe (scope) Del programa, kjer je lahko spremenljivka "vidna" (ali uporabljena). (Spremenljivka znotraj funkcije morda ni vidna za kodo izven funkcije.)

operator Element v računalniškem programu, ki se uporablja za matematiko ali za primerjavo vrednosti.

otrok (child) Ko govorimo o razredih, opisujemo odnose med razredi kot pri starših in otrocih. Otroški razred podeduje značilnosti svojega starševskega razreda.

parameter Vrednost, uporabljena s funkcijo, ko jo kličete ali ko ustvarite objekt (pri klicanju funkcije Python `__init__` na primer). Parametri se včasih imenujejo argumenti.

pixel (pika) Ena točka na zaslonu računalnika - najmanjša točka, ki jo je računalnik zmožen narisati.

platno (canvas) Površina zaslona za risanje. Canvas je razred ki ga ponuja modul tkinter.

podatek (data) Običajno se nanaša na podatke, ki jih shranjuje in obdeluje računalnik.

pogoj (condition) Izraz v programu, ki je podoben vprašanju. Pogoji imajo lahko vrednost true ali false.

pogovorno okno (dialog) Pogovorno okno je običajno majhno okno aplikacije, ki vsebuje nekatera pojasnila, kot so opozorilo ali sporočilo o napaki ali prošnja po odgovoru na vprašanje. Na primer, ko se odločite za odpiranje datoteke, se običajno pojavi pogovorno okno za izbiro datoteke.

pomnilnik (memory) Naprava ali komponenta v računalniku, ki se uporablja za začasno shranjevanje informacije.

program Zbirka ukazov, ki računalniku pove, kaj storiti.

programska oprema (software) Zbirka računalniških programov.

prosojnost (transparency) Pri grafičnem programiranju je del slike, ki se ne prikaže, kar pomeni, da se vidi skozi.

razred (class) Opis ali opredelitev vrste stvari. V programerskem jeziku je razred zbirka funkcij in spremenljivk.

sintaksa Razporeditev in vrstni red besed v programu.

sličica (sprite) lik ali predmet v računalniški igri.

slika (image) Slika na zaslonu računalnika.

spremenljivka (variable) Nekaj, kar se uporablja za shranjevanje vrednosti. Spremenljivka je kot nalepka za informacije, shranjene v pomnilniku računalnika. Spremenljivke niso trajno vezane na določeno vrednost, zato ime "spremenljivka", kar pomeni, da se lahko spremeni.

starš (parent) Pri sklicevanju na razrede in objekte je starševski razred tisti razred od katerega podedujete funkcije in spremenljivke. Z drugimi besedami, otroški razred podeduje značilnosti njegovega starševskega razreda. Ko ne govorimo o Pythonu, je starš oseba, ki vam pove, da si pred spanjem očistite zobe.

stopinja Enote za merjenje kotov.

šestnajstiški (hexadecimal) Način predstavljanja števil, še posebej pri programiranju. Šestnajstiška števila imajo osnovo 16 in vsebujejo števke od 0 do 9 in nato A, B, C, D, E in F.

trk (collision) V računalniških igrah, ko se en lik v igri dotakne drugega lika ali predmeta na zaslonu.

uvoz (import) V Python-u uvoz omogoča, da je modul na valjo za uporabo.

vdelava (embed) Zamenja vrednosti v nizu. Spremenjene vrednosti so včasih imenovane lokacije (placeholder).

zanka (loop) Ponovljen ukaz ali niz ukazov.

FOR KIDS AGED 10+ (AND THEIR PARENTS)

REAL PROGRAMMING. REAL EASY.

ILLUSTRATIONS BY MIRAN LIPOVAČA



Python is a powerful, expressive programming language that's easy to learn and fun to use! But books about learning to program in Python can be kind of dull, gray, and boring, and that's no fun for anyone.

Python for Kids brings Python to life and brings you (and your parents) into the world of programming. The ever-patient Jason R. Briggs will guide you through the basics as you experiment with unique (and often hilarious) example programs that feature ravenous monsters, secret agents, thieving ravens, and more. New terms are defined; code is colored, dissected, and explained; and quirky, full-color illustrations keep things on the lighter side.

Chapters end with programming puzzles designed to stretch your brain and strengthen your understanding. By the end of the book you'll have programmed two complete games: a clone of the famous Pong and "Mr. Stick Man Races for the Exit"—a platform game with jumps, animation, and much more.

**PYTHON RUNS ON WINDOWS,
OS X, LINUX, OLPC LAPTOPS,
AND EVEN RASPBERRY PI!**

As you strike out on your programming adventure, you'll learn how to:

- Use fundamental data structures like lists, tuples, and maps
- Organize and reuse your code with functions and modules
- Use control structures like loops and conditional statements
- Draw shapes and patterns with Python's turtle module
- Create games, animations, and other graphical wonders with tkinter

Why should serious adults have all the fun? *Python for Kids* is your ticket into the amazing world of computer programming.

ABOUT THE AUTHOR

Jason R. Briggs has been a programmer since the age of eight, when he first learned BASIC on a Radio Shack TRS-80. He has written software professionally as a developer and systems architect and served as Contributing Editor for *Java Developer's Journal*. His articles have appeared in *JavaWorld*, *ONJava*, and *ONLamp*. *Python for Kids* is his first book.



www.nostarch.com

THE FINEST IN
GEEK ENTERTAINMENT™

ISBN: 978-1-59327-407-8



9 781593 274078

\$34.95 (\$36.95 CDN)



6 89145 74076 9

SHOULDER
PROGRAMMING KNOW HOW PYTHON