Podloge za stručno usavršavanje učitelja osnovnih škola
za domenu
Računalno razmišljanje i programiranje

03
Kornjačina grafika, ponavljane niza naredbi – programske petlje, definiranje vlastitih funkcija

Uz dozvolu izdavača korišteni su sadržaji iz priručnika:

Leo Budin, Predrag Brođanac, Zlatka Markučič
Smiljana Perić, Dejan Škvorc, Magdalena Babić

Računalno razmišljanje i programiranje u Pythonu
Element, Zagreb, 2017

Kornjača crta, modul `turtle`

U prethodnom poglavlju upoznali smo više osnovnih programskih funkcija koje obavljaju određene zadatke. Sve se one nalaze u osnovnom dijelu *Pythona* te često kažemo da su to ugrađene funkcije. Uz osnovne funkcije u *Pythonu* postoje skupine funkcija koje pomažu pri programiranju određenih problema. Takve su skupine funkcija smještene u tzv. **module**.

Tako u *Pythonu* postoji modul `turtle` koji sadrži funkcije za izradu crteža na zaslonu računala. Crtanje se obavlja u posebnom prozoru koji se automatski otvara prvim pozivom neke funkcije iz tog modula.

Turtle je engleski naziv za kornjaču. Naša zamišljena kornjača djeluje kao mali robot koji se kreće po zaslonu računala. Robot nosi olovku koju može spuštati i podizati. Kada je olovka spuštena, kornjača će pri kretanju ostaviti iza sebe trag, a kada je olovka podignuta kornjača se pomiče bez ostavljanja traga[1]. Olovka ostavlja trag na zaslonu računala, slično kao što olovka ostavlja trag na papiru te nastaje crtež. U nastavku programa koristit ćemo naziv kornjača, a ne olovka.

Mogućnost korištenja (importiranje) pojedine funkcije iz nekog modula mora se zatražiti posebnom naredbom koja ima sljedeći oblik:

```
>>> from ime_modula import ime_funkcije
```

Primjerice, želimo li se iz modula `turtle` koristiti funkcijama za kretanje kornjače unaprijed i unazad, funkcije `forward()` i `back()` te funkcije za podizanje i spuštanje olovke, funkcije `penup()` i `pendown()`, upotrijebit ćemo četiri puta naredbu za importiranje funkcije modula:

```
>>> from turtle import forward
>>> from turtle import back
>>> from turtle import penup
>>> from turtle import pendown
```

Ovakav način importiranja funkcija može postati nepraktičan ako je potrebno uključiti veliki broj funkcija iz nekog modula. Osim toga, vrlo često, a posebice kod korištenja modula `turtle`, nećemo točno moći predvidjeti koje ćemo funkcije trebati. U takvim situacijama možemo zatražiti importiranje svih funkcija nekog modula. Pisat ćemo:

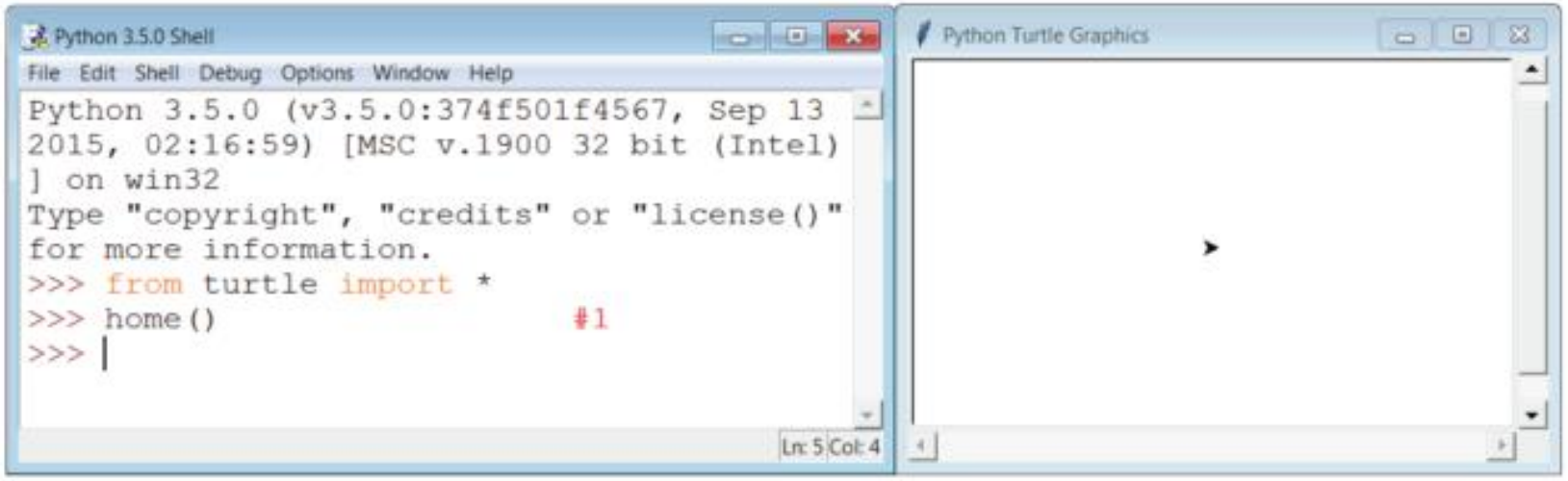
```
>>> from turtle import *
```

Znak zvjezdice `*` označava sve funkcije tog modula. Napišemo li nakon toga u interaktivnom sučelju neku naredbu kojom se poziva kornjačina funkcija,aktivirat će se modul `turtle` i otvorit će se novi prozor s naslovom `Python Turtle Graphics`.

To je **treći prozor** u našem Python okruženju.

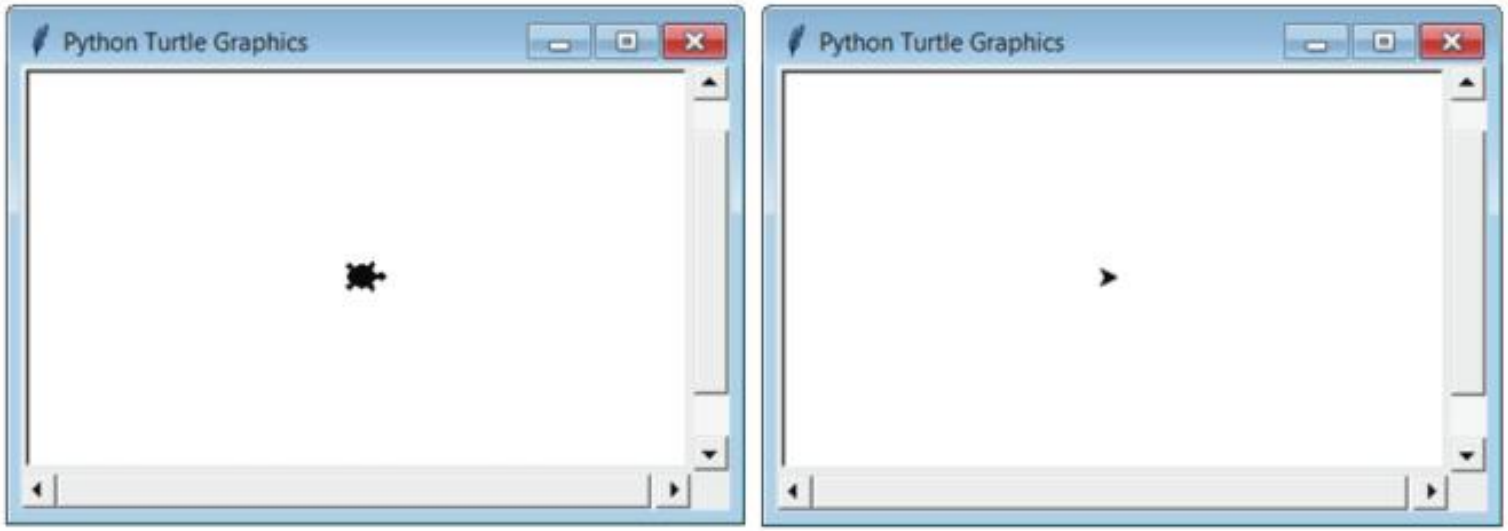
Funkicije `home()`, `reset()`, `shape()`

Prva funkcija koju ćemo upoznati je funkcija `home()`. Ona postavlja kornjaču u početni položaj. Kao što vidimo na slici 3.1., u sredini otvorenog prozora pojavila se strelica usmjerena udesno. Ta strelica predstavlja olovku koju nosi kornjača.



Slika 3.1. *Interaktivni i grafički prozor*

Prikaz položaja olovke strelicom mnogo je praktičniji od oblika kornjače pa se taj oblik uobičajeno rabi i pojavljuje pri otvaranju prozora. No, kako bi se ilustrirala povezanost s pričom o kornjači, moguće je olovku pretvoriti u stilizirani oblik kornjače posebnom naredbom `shape('turtle')`, slika 3.2. Naredbom `shape('classic')` kornjači se vraća oblik strelice.



Slika 3.2. *Prikaz olovke u obliku kornjače i u obliku strelice*

Osim funkcije `home()`, na raspolaganju nam je i funkcija `reset()` koja za razliku od funkcije `home()` briše sve što smo nacrtali do toga trenutka te postavlja kornjaču u početni položaj.

Osnovne funkcije kornjače

Jedinica pomaka (koraci) `d` je broj točaka, odnosno *piksela* zaslona računala za koje se kornjača treba pomaknuti. Na zaslonu računala slika se sastoji od točkica za koje se kod nas prema engleskom nazivu *pixel* (engl. *picture element* – slikovni element, element slike) udomaćio naziv **piksel**.

`fd(d)` ili `forward(d)` – funkcija kojom kornjaču pomičemo unaprijed za zadani broj koraka *m*

`bk(d)` ili `back(d)` – funkcija kojom kornjaču pomičemo unazad za zadani broj koraka *m*

`pu()` ili `up()` odnosno `penup()` – funkcija kojom podižemo kornjaču

`pd()` ili `down()` odnosno `pendown()` – funkcija kojom spuštamo kornjaču.

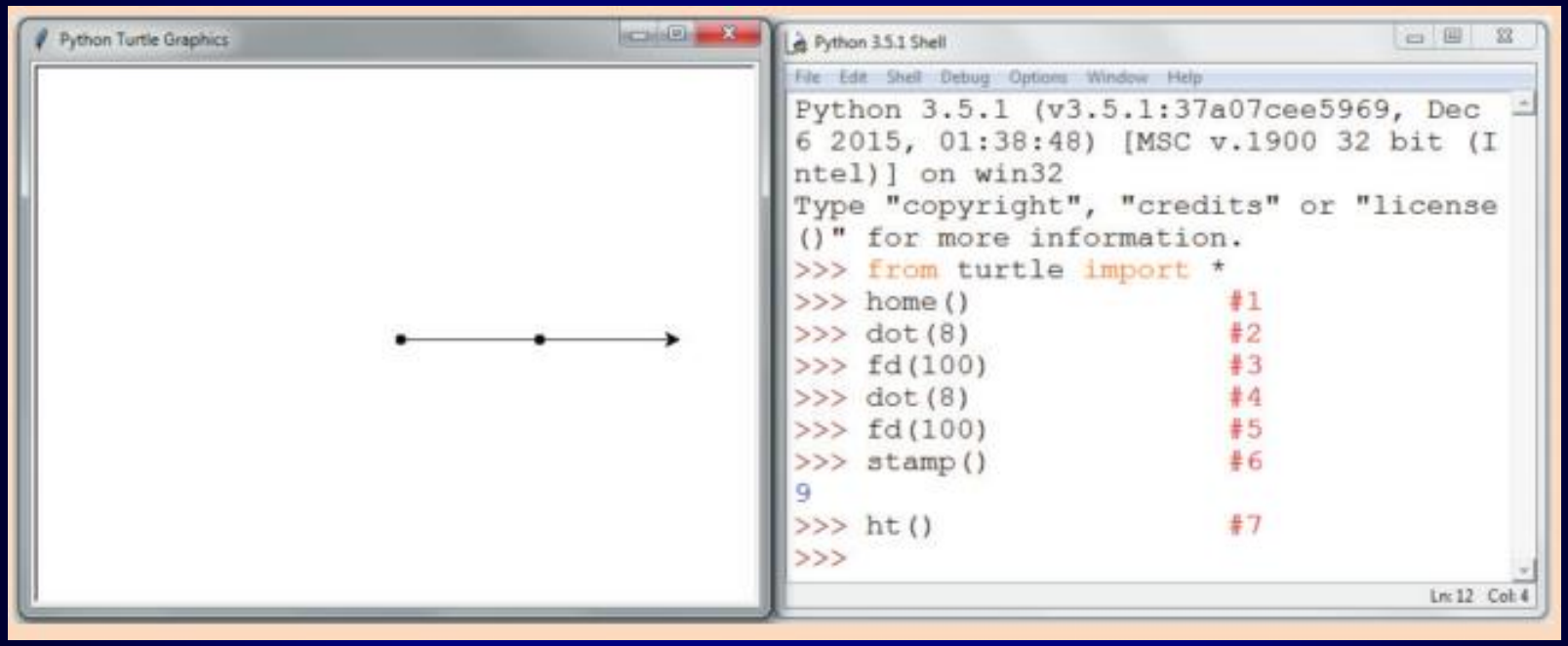
Pri kretanju kornjača ostavlja trag ako je spuštena, no ponekad će nam trebati da na pojedinom mjestu kornjača ostavi svoj otisak poput pečata ili nacrta točku određene veličine. U tom slučaju ćemo rabiti funkcije:

`stamp()` koja ostavlja otisak na mjestu na kojem se ona trenutno nalazi

`dot(vel)` koja na mjestu na kojem se kornjača nalazi crta točku veličine `vel` piksela.

Sliku kornjače možemo sakriti naredbom `hideturtle()` ili `ht()`. Tom ćemo se funkcijom koristiti kako bi nam na zaslonu bio prikazan samo crtež koji smo nacrtali, bez strelice za označavanje položaja kornjače. Želimo li kornjaču ponovno prikazati kako bismo nastavili s crtanjem, napisat ćemo naredbu `showturtle()` ili `st()`.

Uporabom tih funkcija možemo u nizom naredbi u interaktivnom sučelju nacrtati sljedeću sliku:



Primijetimo da se nakon naredbe `stamp()` u interaktivnom sučelju pojavio neki broj. Taj je broj rezultat djelovanja funkcije `stamp()` te nam može poslužiti za brisanje otiska funkcijom `clearstamp()`. U ovom trenutku taj ćemo broj zanemariti.

Ponavljanje niza naredbi

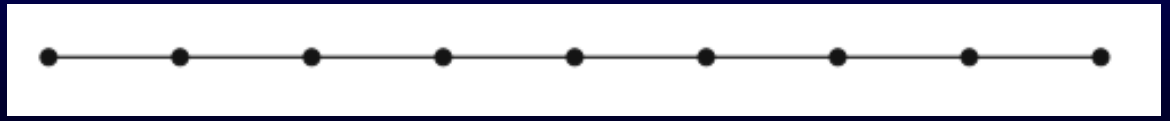
U mnogim slikovnim prikazima pojavit će se potreba za ponavljanjem pojedinih dijelova crteža. Pokazat ćemo to na jednostavnom primjeru. Od ovog primjera nadalje nećemo više na našim slikama prenositi cijele prozore (interaktivni i grafički), već ćemo prikazivati samo njihove važne dijelove.

Želimo nacrtati dužinu duljine 400 piksela s tim da na svakih 50 piksela nacrtamo točku veličine 6 piksela.

Taj ćemo crtež izraditi sljedećim nizom naredbi napisanih u interaktivnom sučelju:

```
>>> reset()
>>> pu(); bk(200); pd()
>>> dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> fd(50); dot(6)
>>> ht()
```

Dobiveni slikovni prikaz podsjeća na brojevni pravac. Kasnije ćemo pokazati da se kornjačom može nacrtati i brojevni pravac s označenim pojedinim točkama.



Niz naredbi iz interaktivnog prozora možemo prepisati u editor i pohraniti kao program. To će biti naš prvi program u ovom poglavlju i dobit će naziv `program_3_1.py`.

```
#Crtanje crte s točkama - program_3_1.py
from turtle import *
reset()
pu(); bk(200); pd()
dot(6)
fd(50); dot(6)
fd(50); dot(6)
fd(50); dot(6)
fd(50); dot(6)
fd(50); dot(6)
fd(50); dot(6)
fd(50); dot(6)
fd(50); dot(6)
ht()
```

Izvođenjem ovog programa dobivamo jednaki crtež.

Bilo bi jako dobro da ponavljanje možemo kraće zapisati.

Ponavljanje petljom `for`

Ponavljanje određenog slijeda naredbi može se obaviti programskom petljom. U programskom jeziku *Python*, jedna od naredbi za realizaciju programske petlje je naredba `for`. Koristeći se naredbom `for` u interaktivnom sučelju, možemo napisati programsku petlju na sljedeći način:

```
>>> for i in range(4):                                    #1
        print('Ovaj se ispis ponavlja četiri puta')

Ovaj se ispis ponavlja četiri puta
Ovaj se ispis ponavlja četiri puta
Ovaj se ispis ponavlja četiri puta
Ovaj se ispis ponavlja četiri puta
>>>
```

Naredba (#1) određuje da se naredba koja slijedi obavlja četiri puta. Nakon što napišemo dvotočku i pritisnemo *(Unos)* treptajuća će se oznaka postaviti za određeni broj mjesta udesno, tj. naznačit će uvlaku. Uobičajeno je da se uvlaka (engl. *indent*) sastoji od četiriju znakovnih mjesta, što je i postavljeno početnim postavkama u editoru programskog jezika *Python*. Sve naredbe koje se trebaju ponavljati, odnosno naredbe u petlji koje nazivamo blokom naredbi, moraju biti napisane s uvlakom. U našem primjeru radi se samo o jednoj naredbi `print()`. Nakon što u interaktivnom sučelju dvaput pritisnemo *(Unos)*, naredba `print()` obavit će se četiri puta jer smo u parametru funkcije `range()`, koja se nalazi unutar naredbe `for` naveli broj 4. Naime, varijabla s imenom `i` uzastopno će na svakom početku ponavljanja petlje poprimati redom četiri vrijednosti i to: `0`, `1`, `2`, `3`. Ime te varijable koju nazivamo varijablom petlje, možemo proizvoljno odabrati.

Vrijednosti varijable `i` ne moramo rabiti u naredbama petlje ako nam nisu potrebne, ali u mnogim primjenama one nam mogu biti korisne.

Pogledajmo promjenu varijable `i` tijekom izvođenje naredbe **for**:

```
>>> for i in range(4):
        print('U {0}. prolazu petlje, i je {1}'.format(i + 1, i))          #2

U 1. prolazu petlje, i je 0
U 2. prolazu petlje, i je 1
U 3. prolazu petlje, i je 2
U 4. prolazu petlje, i je 3
>>>
```

Ispisni string u naredbi (#2) oblikovali smo tako da na mjesto `{0}` ispisujemo vrijednost `i + 1`, a na mjestu `{1}` vrijednost varijable `i`. Uvijek trebamo imati na umu da sva brojenja u *Pythonu* počinju vrijednošću `0`. Prema tome, najveća vrijednost varijable `i` je za jedan manja od broja ponavljanja petlje. Utvrdimo to još jednim primjerom:

```
>>> n = 15                                                                  #3
>>> for x in range(n):                                                      #4
        print(x, end=' ')                                                   #5

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Naredbom (#3) varijabli `n` pridijelili smo vrijednost koja će odrediti broj ponavljanja naredbe unutar naredbe (#4). Varijablu petlje nazvali smo imenom `x`. Promjenom parametra `end` u naredbi (#5) odredili smo da se nakon obavljenog ispisa funkcijom `print()` ne prelazi u novi red. Na taj smo način sve vrijednosti umjesto u petnaest redova ispisali u jednom redu.

Na temelju gornjeg opisa naš ćemo `program_3_1.py` napisati uporabom petlje **for**.

U novom programu ne moramo osam puta napisati naredbe `fd(50)` i `dot(6)` već će se one u petlji obaviti osam puta. Program bi mogao izgledati ovako:

```
#Crtanje crte s točkama u petlji program_3_2.py
from turtle import                                    #1
reset()                                               #2
pu(); bk(200); pd()                                   #3
dot(6)                                                #4
for i in range(8):                                    #5
    fd(50)                                            #6
    dot(6)                                            #7
ht()                                                  #8
```

*Utvrdimo da se iz ovog teksta programa može jednostavno ustanoviti kojim će redom Python izvoditi naredbe. On počinje od naredbe (#1) i zatim redom do naredbe (#4) kojom će nacrtati početnu točku. Zatim Python nailazi na naredbu (#5) koja određuje da će se naredbe pisane s uvlakom izvoditi osam puta. Te naredbe s uvlakom (#6), (#7) čine blok naredbi petlje **for**. Tek nakon što se osam puta izvedu naredbe petlje (#6), (#7), obavit će se naredba (#8) koja opet počinje u prvom stupcu.*

Nakon izvođenja zadnje naredbe Python obustavlja izvođenje i u interaktivnom sučelju se pojavljuje prompt >>>.

Oblici programskih funkcija

U *Pythonu*, a slično je i u drugim programskim jezicima, može se više programskih naredbi smišljenih za obavljanje nekog određenog posla objediniti u jednu cjelinu. Takve se cjeline nazivaju **programskim funkcijama** ili kraće samo **funkcijama**. Kod zadavanja imena funkcija vrijede ista pravila kao i za imena varijabli.

Dosad smo upoznali neke od ugrađenih funkcija *Pythona*: `int()`, `str()`, `input()` i `print()`, te neke od funkcija koje se nalaze u modulu `turtle`. Te se funkcije donekle međusobno razlikuju.

Kod nekih se funkcija unutar zagrada iza njihova imena nalaze imena varijabli koje nazivamo *parametrima funkcije*. Tako su u funkcijama `fd(m)` i `dot(d)` varijable `m` i `d` parametri funkcije. Kada te funkcije rabimo ("pozivamo") unutar programa, onda na mjesta parametara moramo pisati konkretne vrijednosti ili imena varijabli u kojima se nalaze konkretne vrijednosti. Te vrijednosti zvat ćemo **argumentima**. Posao koji funkcija obavlja ovisit će o vrijednostima ulaznih argumenata. Tako će poziv funkcije `fd(50)` pomaknuti kornjaču za 50 piksela, dok će poziv `fd(20)` pomaknuti kornjaču za 20 piksela.

Kod nekih drugih funkcija unutar zagrada ništa ne piše (par zagrada je prazan) i funkcija uvijek obavlja jednak posao. Primjerice, takve su funkcije `home()` i `reset()`.

Dosad spomenute funkcije samo djeluju na ponašanje kornjače u njezinu prozoru i ne daju nam nikakvu povratnu vrijednost. Kažemo da nam one ne vraćaju ništa. Postoje i funkcije koje će svojim obavljanjem proizvesti neki nama koristan rezultat koji možemo dalje rabiti u programu. Kažemo da nam takve funkcije vraćaju rezultat. Tako će funkcija `input()` vratiti string koji smo utipkali, dok će funkcija `int()` ulazni argument tipa `str` pretvoriti u cijeli broj i vratiti ga kao tip `int`. U 2. smo poglavlju vidjeli takve primjere.

Definiranje vlastitih funkcija koje nemaju ulazne parametre

Niz naredbi programa `program_3_2.py` prvo će obrisati cijeli grafički prozor, pomaknuti kornjaču za 200 piksela ulijevo, staviti početnu točku i zatim u petlji `for` nacrtati osam crta duljine 50 piksela s točkama veličine 6 piksela na njihovu kraju. Na kraju će sakriti znak kornjače. Navedene naredbe možemo napisati tako da one čine jednu cjelinu, tj. **funkciju**. U interaktivnom sučelju tu ćemo funkciju napisati na sljedeći način:

```
>>> from turtle import                                    #1
>>> def crta_točka():                                     #2
        reset()
        pu(); bk(200); pd()                               #3
        dot(6)
        for i in range(8):
            fd(50)
            dot(6)
        ht()                                              #4

>>>
```

Prije pisanja funkcije moramo naznačiti da ćemo se koristiti funkcijama modula `turtle` (**#1**).

Definiranje funkcije počinje naredbom (**#2**) koja čini **zaglavlje funkcije**. Zaglavlje započinje ključnom riječi `def` iza koje pišemo odabrano ime funkcije. Ime se odabire po istim pravilima koja vrijede za pisanje imena varijabli. Iza imena moramo staviti zagrade i na kraju znak dvotočke. Našu prvu funkciju nazvat ćemo `crta_točka()`. Uočimo da funkcija nema parametara tj. unutar zagrada ništa ne piše.

Kada nakon te dvotočke pritisnemo tipku *(Unos),* neće se pojaviti prompt >>>, već će se novi red upisivati uvučeno za četiri znakovna mjesta. Sve sljedeće naredbe funkcije pišu se jedna ispod druge s uvlakom od četiri znaka. Te naredbe tvore blok naredbi ili **tijelo funkcije**. U ovom se našem primjeru tijelo funkcije sastoji od devet naredbi, od kojih su tri napisane u istom redu (#3).

Nakon zadnje naredbe funkcije (#4), ostavljamo prazan redak tj. dvaput uzastopce pritisnemo tipku *(Unos)* te će se pojaviti znak >>> i tada je funkcija spremna za uporabu. Sada sučelje možemo rabiti na uobičajeni način. Našu novu funkciju možemo pozivati na isti način kao i ugrađene funkcije. Pogledajmo:

```
>>> crta_točka()
```

Nakon tog poziva funkcije otvorit će se grafički prozor i nacrtat će se crta s točkama.

Ako nakon toga funkcijom `reset()` pobrišemo crtu, možemo ju opet nacrtati ponovnim pozivom funkcije `crta_točka()`.

Zagrade iza imena funkcija su važne. Naime, napišemo li ime funkcije bez zagrada, dobit ćemo samo podatak o tome gdje je funkcija smještena u radnom spremniku računala, što nam sada nije važno.

```
>>> crta_točka
<function crta_točka at 0x02588030>
```

Znamo već da će gašenjem računala, odnosno zatvaranjem interaktivnog prozora *Pythona*, nestati naša funkcija. Isto tako, znamo da ju možemo sačuvati, ako ju definiramo u editoru i pohranimo unutar programa koji bi mogao izgledati ovako:

```
#Program za crtanje crte s točkama - program_3_3.py

from turtle import *                                          #1

def crta_točka():                                             #2
    reset()
    pu(); bk(200); pd()
    dot(6)
    for i in range(8):
        fd(50)
        dot(6)
    ht()                                                      #3

crta_točka()                                                  #4
```

Jednako kao u interaktivnom sučelju naredba (#1) povezuje program i funkcije modula `turtle`. Definicija funkcije (#2) počinje kao i u interaktivnom sučelju ključnom riječi `def`, a naredbe unutar funkcije koje su napisane s uvlakom određuju što će funkcija raditi. Unutar funkcije nalazi se i petlja `for` čiji je blok naredbi dodatno uvučen. Naredba (#3) je zadnja naredba funkcije.

Mi ćemo poziv funkcije `crta_točka()` obaviti našom jedinom naredbom "glavnog" programa (#4), pišući je od početka reda. Pisanjem naredbe od početka reda, *Python* zaključuje da ta naredba više ne pripada tijelu funkcije, već se radi o naredbi koju je moguće odmah izvesti. Ta će naredba pozivom funkcije `crta_točka()` pokrenuti posao koji ta funkcija radi – iscrtavanje crte s točkama.

Definiranje vlastitih funkcija s ulaznim parametrima

U programu `program_3_3.py` *funkcija* `crta_točka()` *pri svakom pozivu obavlja jednak posao: crta devet točaka veličine 6 piksela koje su spojene crtama duljine 50 piksela.*

U našoj funkciji možemo uočiti sljedeće vrijednosti:

- *broj crta između točaka iznosi 8*
- *razmak između točaka iznosi 50*
- *veličina točke izražene u pikselima iznosi 6.*

Uočimo da je broj točaka za jedan veći od broja crta koje ih spajaju – točaka ima devet, a spojnih crta osam.

Korisno bi bilo kada bismo mogli funkciju pripremiti tako da opisane vrijednosti možemo mijenjati i prikazivati crte s različitim brojem točaka, s različitim veličinama točaka i različitim razmacima između točaka. To bismo mogli postići tako da na mjesta gdje su zapisane vrijednosti napišemo imena varijabli i da omogućimo da se tim varijablama prilikom poziva funkcije dodijele odgovarajuće željene vrijednosti.

Kako bismo to mogli učiniti, uvedimo sljedeće varijable:

- `n` – *broj crta između točaka*
- `m` – *razmak između točaka*
- `d` – *veličina točaka izražena u pikselima.*

Imena varijabli napisat ćemo pri definiciji funkcije na ona mjesta gdje su pisale pojedine vrijednosti. Kao što smo već rekli, te varijable zovemo parametrima *funkcije, a njihova imena pišemo unutar zagrada iza imena funkcije. Tako ćemo na osnovi funkcije iz programa* `program_3_3.py` *napisati novi program s novim oblikom funkcije:*

```
#Program za crtanje crte s točkama - program_3_4_a.py

from turtle import *                                          #1

def crta_točka(n, m, d):                                      #2
    reset()
    pu(); bk(n * m // 2); pd()                                #3
    dot(d)
    for i in range(n):
        fd(m)
        dot(d)
    ht()                                                      #4

crta_točka(8, 50, 6)                                          #5
```

Prilikom pokretanja ovog programa, Python će prvo izvesti naredbu (#1) te će preskočiti naredbe od (#2) do (#4) i početi izvoditi prvu naredbu na koju naiđe koja počinje u prvom stupcu. U našem primjeru to je naredba (#5) kojom se poziva na izvođenje funkcije `crta_točka(8, 50, 6)`.

Redoslijed parametara određen pri definiciji funkcije vrlo je važan. Pri pozivu funkcije na prvom mjestu treba zapisati argument koji određuje broj crta `n`, argument napisan na drugom mjestu odredit će razmak između točaka `m`, a argument napisan na trećem mjestu odredit će veličinu točaka `d`. Isto tako je važno da broj argumenata bude jednak broju parametara. Ako to ne uvažimo, Python će nam javiti pogrešku:

```
>>> točka_crta(8, 20)
Traceback (most recent call last):
     File "<pyshell#9>", line 1, in <module>
        crta_točka(20, 20)
TypeError: crta_točka() missing 1 required positional argument: 'd'
```

Kao argumente možemo napisati i imena varijabli u kojima smo prethodno pohranili vrijednosti argumenata. Tako možemo pisati:

```
>>> veličina = 8
>>> broj = 10
>>> razmak = 20
>>> crta_točka(broj, razmak, veličina)
```

Moramo znati da imena varijabli koje smo uveli pri definiciji funkcije vrijede samo unutar funkcije i zovemo ih **lokalnim varijablama**. Pri pozivu funkcije tim će lokalnim varijablama biti pripisane vrijednosti argumenata iz poziva. Unutar funkcije te se vrijednosti mogu rabiti i može ih se i mijenjati, ali nakon napuštanja funkcije one više ne postoje i ne može ih se dohvaćati izvan funkcije. U poglavlju 4. upoznat ćemo kako vratiti odabrane vrijednosti u dio programa koji je funkciju pozvao.

Funkcije s unaprijed zadanim vrijednostima parametara (opcijski parametri)

Pretpostavimo da pri uporabi funkcije `crta_točka()` želimo uglavnom crtati crtu kod koje je razmak između dviju točaka 50 piksela i debljina točke 6 piksela, a želimo mijenjati samo broj točaka.

S druge strane želimo ostaviti mogućnost da ponekad i promijenimo razmak između točaka te debljinu točaka. U *Pythonu* se to lako može ostvariti. Odabranim argumentima u zaglavlju funkcije ćemo pridružiti inicijalnu ili početnu vrijednost (engl. *default value*). To ćemo načiniti korištenjem znaka pridruživanja. Želimo li da razmak između točaka bude 50 piksela, a veličina točke 6 piksela, treba napisati `m=50` i `d=6`. U pravilima pisanja preporučuje se da ovdje iznimno ispred i iza znaka pridruživanja ne stavljamo razmak.

Program s ovako definiranom funkcijom može izgledati ovako:

```
# Funkcija s opcijskim parametrima - program_3_4_b.py

from turtle import *                                          #1

def crta_točka(n, m=50, d=6):                                 #2
    reset()
    pu(); bk(n * m // 2); pd()                                #3
    dot(d)
    for i in range(n):
        fd(m)
        dot(d)
    ht()                                                      #4

crta_točka(8)                                                 #5
```

U naredbi #5 naveden je samo jedan argument!

Nakon izvođenja programa `program_3_4_b.py` dobit ćemo jednaku sliku kao i izvođenjem programa `program_3_4_a.py`. Kao i dosad u interaktivnom sučelju možemo višekratno pozivati funkciju `crta_točka()`, ali ovaj puta s različitim argumentima.

```
>>> crta_točka(7)                          #6
>>> crta_točka(5,80)                       #7
>>> crta_točka(10,20,8)                    #8
>>> crta_točka(10, m=20, d=6)              #9
>>> crta_točka(8, d=10)                   #10
```

U naredbi (#6) izostavili smo argumente za parametre `m` i `d` pa će oni poprimiti unaprijed postavljene vrijednosti. Poziv funkcije u naredbi (#7) ima dva argumenta. Prvi argument povezuje se s parametrom `n`, a drugi argument s parametrom `m`. Parametar `d` imat će unaprijed postavljenu vrijednost argumenta. U naredbi (#8) navedene su vrijednosti argumenata za sva tri parametra, što znači da će obje unaprijed postavljene vrijednosti argumenata biti promijenjene. Poziv funkcije u naredbi (#9) napisan je malo drukčije od poziva u naredbi (#8). Ovdje su nove vrijednosti argumenata za parametre `m` i `d` napisane tako da smo vrijednosti pridružili imenima parametara. Oba poziva funkcije obavit će jednak posao. Način pridruživanja vrijednosti argumenata omogućuje nam da pri određivanju vrijednosti argumenata napišemo samo one koji želimo mijenjati. Tako u pozivu funkcije u naredbi (#10) "preskačemo" parametar `m` i samo parametru `d` pridjeljujemo novu vrijednost argumenta. Vrijednost parametra `m` poprimit će osnovnu vrijednost.

Prema tome, oni parametri kojima je u definiciji funkcije postavljena osnovna vrijednost argumenta pri pozivu funkcije se mogu, ali i ne moraju navoditi. U pozivu funkcije oni se pojavljuju kao argumenti samo onda kada postavljene osnovne vrijednosti želimo mijenjati. Dakle, imamo mogućnost (opciju) da ih pri pozivu funkcije uopće ne spominjemo. Zbog toga su oni dobili i naziv **opcijski parametri**.

Oblikovanje programa

U prethodnim smo se primjerima uvjerili da *Python* počinje izvoditi one naredbe koje započinju u prvom stupcu teksta. U našim programima to je bila najprije naredba za uvođenje funkcija iz modula `turtle`. Ostale naredbe koje definiraju vrijednosti argumenata i pozivaju funkcije u tim programima čine **glavni dio** programa. Definicija funkcije ne pripada tom glavnom dijelu programa. Njezine će se naredbe obavljati onda kada to zatraži glavni program. Funkcije djeluju kao mali programi. U nekim se programskim jezicima funkcije stoga nazivaju **potprogramima**.

Kod složenijih programa definirat ćemo veći broj funkcija, a i glavni dio programa sastojat će se od većeg broja naredbi. Ako se pri pisanju takvih programa ne pridržavamo nekog ujednačenog stila pisanja, imat ćemo poteškoća pri razumijevanju čak i vlastitih programa.

Napišimo program u kojem ćemo vrijednosti argumenata upisati prilikom izvođenja programa. Na početku programa odredit ćemo koliko puta želimo ponoviti crtanje.

```
#Program s upisom argumenata - program_3_5.py

from turtle import *                                                    #1

def crta_točka(m, d=50, p=6):
    reset()                                                             #2
    pu(); bk(d * m // 2); pd()
    dot(p)
    for i in range(m):
        fd(d)
        dot(p)
    ht()                                                                #3

k = int(input('Upiši broj ponavljanja k = '))                           #4
for i in range(k):
    n = int(input('Upiši broj odsječaka n = '))
    m = int(input('Upiši razmak između točaka m = '))
    d = int(input('Upiši promjer točke d = '))
    crta_točka(n, m, d)                                                 #5
    input('Pritisni tipku (Unos) za nastavak')                          #6
print('Zadano je {} ponavljanja!'.format(k))
print('Kraj programa')                                                  #7
```

Nakon što se izvede naredba (#1), izvest će se naredba (#4), sljedeća naredba koja je napisana počevši od prvog stupca. Dalje će se redom izvoditi naredbe sve do naredbe (#5) kojom se poziva funkcija. Tim pozivom počinju se izvoditi naredbe funkcije počevši s naredbom (#2). Nakon izvođenja zadnje naredbe funkcije (#3), izvođenje se nastavlja naredbom (#6) tj. prvom naredbom iza poziva funkcije. Kada se obavi i zadnja naredba (#7) glavnog programa, program završava što vidimo po tome da se u interaktivnom sučelju pojavljuje znak prompt >>>.

Namjerno su pobrkana imena varijabli !
Pri pozivu funkcije važan je redoslijed a ne ime !

Pokazalo se vrlo korisnim program podijeliti na manje cjeline i definirati funkcije koje imaju jedinstvene zadaće. U nekim se programskim jezicima i naredbe glavnog programa oblikuju kao jedna posebna funkcija koja se obično naziva `main()` (engl. *main* – glavni). Tako napisani programi izgledat će ujednačeno i pregledno.

Kako bismo usvojili standardne načine pisanja programa, iako to u *Pythonu* nije nužno činiti, naše ćemo programe oblikovati tako da definiramo i glavnu funkciju programa `main()`. U našem će slučaju glavni program u pravilu imati samo jednu naredbu tj. sastojat će se samo od poziva glavne funkcije `main()`.

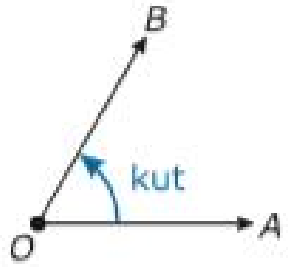
Program 3_5 preoblikovat ćemo ovako:

```
#Crtanje više crta različitih duljina s točkama program_3_6.py

from turtle import *                                                    #1

def crta_točka(m, d=50, p=6):                                           #2
    reset()
    pu(); bk(d * m // 2); pd()
    dot(p)
    for i in range(m):
        fd(d)
        dot(p)
    ht()

def main():                                                             #3
    n = int(input('broj'))
    d = int(input('duljina'))
    p = int(input('promjer'))
    crta_točka(n, d, p)                                                 #4
    print('Kraj programa')                                              #5

main()                                                                  #6
```

Nakon što započne izvođenje naredbom (#1) program će pronaći tek jednu jedinu naredbu koju treba izvesti – naredbu (#6) kojom se poziva funkcija `main()` (#3). Unutar funkcije `main()` *Python* će izvoditi redom naredbe do naredbe (#4) kojom se poziva funkcija `crta_točka()` (#2). Nakon povratka iz funkcije `crta_točka()` nastavlja se izvođenje naredbi funkcije `main()`. Kada naredbom (#5) završi izvođenje funkcije `main()`, trebala bi se izvoditi naredba iza poziva funkcije. Međutim, iza poziva funkcije `main()` više nema ni jedne naredbe te će *Python* okončati izvođenje programa

Kornjača može mijenjati smjer kretanja

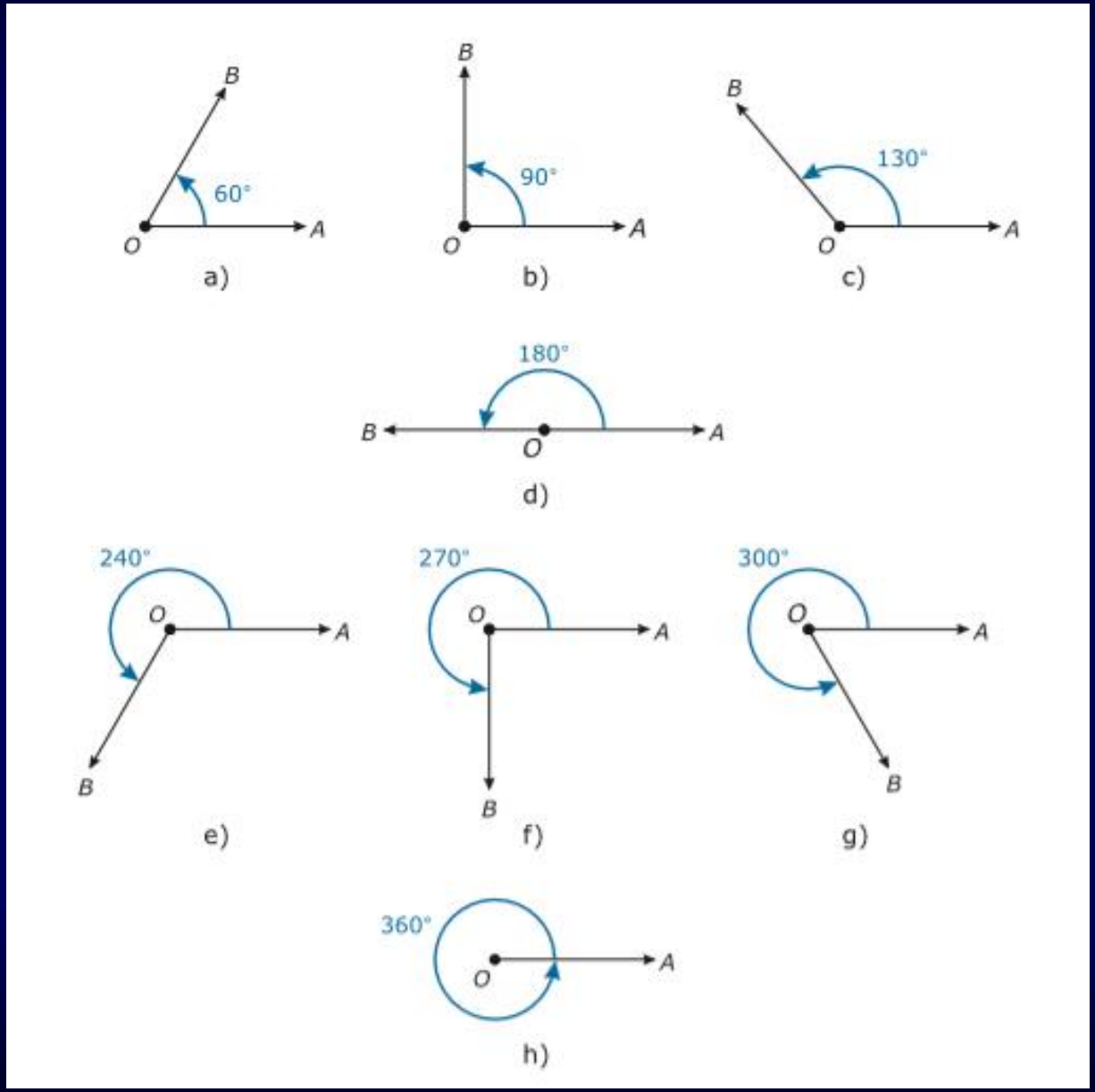
Dva polupravca koje polaze iz iste točke, a nisu istog smjera čine **kut**.



Slika 3.6. *Kut*

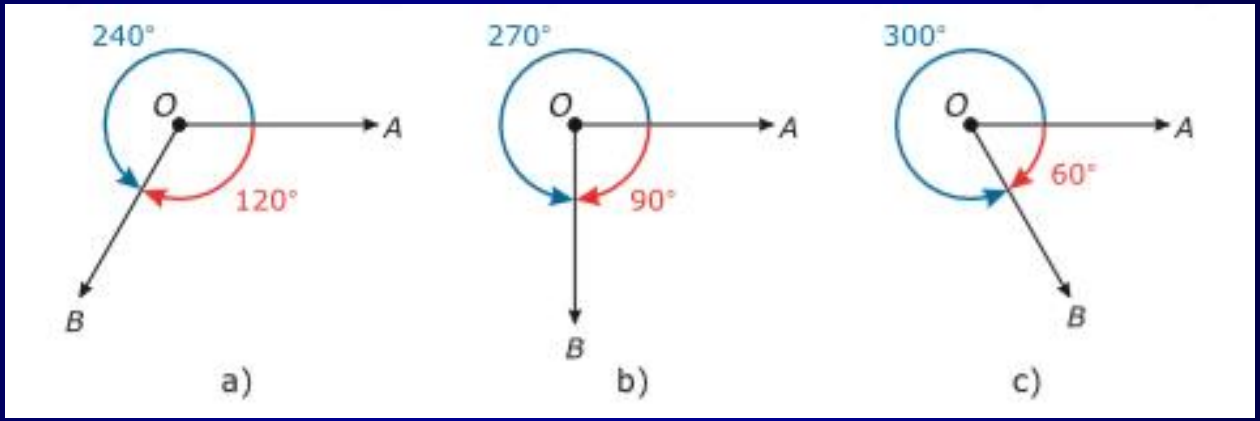
Točka *O* na slici 3.6. je **vrh** kuta. Iz vrha polaze dva polupravca *OA* i *OB*. Možemo zamisliti da je kut *OAB* nastao tako da se na početku polupravac *OB* poklapao s polupravcem *OA* i da se nakon toga rotira oko vrha *O* kao što rotiraju kazaljke na satu. Uobičajeno je da pri mjerenju veličine kuta za pozitivni smjer uzimamo rotaciju obrnutu od rotacije kazaljke na satu. Veličinu rotacije, odnosno **veličinu** kuta, mjerit ćemo u **stupnjevima** pri čemu rotacija za puni krug iznosi 360°. Tako se velika kazaljka sata u jednom satu okrene za 360°.

Različite veličine kutova izražene stupnjevima



Funkcije `left(kut)` *i* `right(kut)`

Do istog položaja polupravca možemo doći na dva načina: rotacijom obrnuto od kazaljke na satu ili rotacijom u smjeru kazaljke na satu.



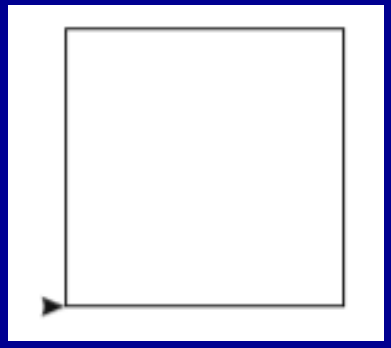
U modulu `turtle` postoje dvije funkcije za rotiranje kornjače:

- funkcija `left(kut)` ili kraće `lt(kut)` obavlja rotaciju kornjače ulijevo (obrnuto od smjera kazaljke na satu) za veličinu kuta `kut`
- funkcija `right(kut)` ili kraće `rt(kut)` obavlja rotaciju kornjače udesno (u smjeru kazaljke na satu) za veličinu kuta `kut`.

Ove dvije funkcije nadopunjuju našu zbirku funkcija koje smo dosad naučili, pa možemo stvarati nove i složenije crteže.

Pogledajmo kako se sa samo dvije funkcije može nacrtati kvadrat. Podsjetimo se, kvadrat ima četiri jednake stranice i četiri jednaka kuta od 90°. Prema tome, kvadrat bismo mogli nacrtati sljedećim nizom naredbi:

```
>>> reset()
>>> fd(150); lt(90)
>>> fd(150); lt(90)
>>> fd(150); lt(90)
>>> fd(150); lt(90)
```



S obzirom na to da se naredbe `fd(150); lt(90)` ponavljaju četiri puta, možemo ih obaviti u petlji:

```
>>> reset()
>>> for i in range(4):
        fd(150); lt(90)

>>>
```

Program za crtanje pravilnih mnogokuta

```
#Crtanje pravilnih mnogokuta - program_3_8.py

from turtle import *

def pravilni_mnogokut(n, a):
    kut = 360 // n                                    #1
    for i in range(n):                                #2
        fd(a)                                         #3
        lt(kut)                                       #4

def main():
    home()
    n = int(input('Upiši broj stranica n = '))        #5
    a = int(input('Upiši duljinu stranice a = '))     #6
    pravilni_mnogokut(n, a)                           #7
    ht()

main()
```

Funkcija `pravilni_mnogokut()` (#7) je jednostavna. Naredbom (#1) izračunava se kut zakreta kornjače. Zatim se u petlji, koja se ponavlja `n` puta (#2), crta stranica duljine `a` (#3) te se kornjača zakreće za `kut` stupnjeva ulijevo (#4). Tako će kornjača nakon `n` zakreta načiniti puni kut.

U glavnoj funkciji `main()` možemo odabrati broj stranica, odnosno broj kutova mnogokuta i duljinu stranice (#5) i (#6).

Funkcija `divmod()`

Funkcija `pravilni_mnogokut()` *(#7) djelovat će bez pogreške kada je 360 djeljivo sa* `n`*, tj. ako je ostatak dijeljenja* `360//n` *jednak nuli. Je li broj djeljiv sa* `n` *možemo provjeriti s pomoću operatora za ostatak cjelobrojnog dijeljenja* `%`*. U ovom će nam primjeru dobro doći ugrađena funkcija* `divmod()` *koja nam zamjenjuje operatore za cjelobrojno dijeljenje i računanje ostatka cjelobrojnog dijeljenja. Tako možemo pisati:*

```
>>> divmod(9, 2)                                    #8
(4, 1)
>>> divmod(9, 9)                                    #9
(1, 0)
```

Izvođenjem funkcije dobivamo uređeni par u kojem je prva vrijednost cjelobrojni količnik, a druga vrijednost ostatak pri cjelobrojnom djeljenju.

Primjenom funkcije `divmod(360, n)` provjerit ćemo za koje će mnogokute funkcija `pravilni_mnogokuti()` djelovati bez pogreške

```
>>> for n in range(3, 13):
        kut, ostatak = divmod(360, n)
        print('n = {}, kut = {}, ostatak = {}'.format(n, kut, ostatak))

n = 3, kut = 120, ostatak = 0
n = 4, kut = 90, ostatak = 0
n = 5, kut = 72, ostatak = 0
n = 6, kut = 60, ostatak = 0
n = 7, kut = 51, ostatak = 3
n = 8, kut = 45, ostatak = 0
n = 9, kut = 40, ostatak = 0
n = 10, kut = 36, ostatak = 0
n = 11, kut = 32, ostatak = 8
n = 12, kut = 30, ostatak = 0
>>>
```

Iz ovog se ispisa vidi da će pravilni mnogokuti od trokuta do dvanaesterokuta svi osim sedmerokuta i jedanaesterokuta biti nacrtani bez pogreške. Pogreška pri crtanju sedmerokuta i jedanaesterokuta nastaje zbog toga što broj 360 nije djeljiv brojevima 7 i 11.

Uredan ispis metodom `format()`

Metoda `format()` omogućuje uredniji ispis brojeva:

```
>>> a = 1
>>> b = 10
>>> c = 100
>>> d = 1000
>>> print('{0:4d}'.format(a))
   1
>>> print('{0:4d}'.format(b))
  10
>>> print('{0:4d}'.format(c))
 100
>>> print('{0:4d}'.format(d))
1000
>>> print('{0:4d}\n{1:4d}\n{2:4d}\n{3:4d}'.format(a, b, c, d))
   1
  10
 100
1000
>>>
```

U ispisnom stringu unutar vitičastih zagrada iza rednog broja vrijednosti koja se ispisuje stavlja se dvotočka i zatim odabrani broj predviđenih mjesta te slovo `d` ako se ispisuje broj tipa `int`. Za ispis nekih drugih tipova podataka rabe se druga slova. Metoda `format()` će one brojeve koji imaju manje znamenaka od broja predviđenih mjesta za ispis ispisati poravnane zdesna. Moglo bi se reći da će metoda nadopuniti takve brojeve umetanjem praznina s lijeve strane.

Uočimo da se prethodni primjer za ispis veličine kutova pravilnih mnogokuta može napisati na uredniji način:

```
>>> for n in range(3, 13):
        kut, ostatak = divmod(360, n)
        print('n ={:2d}, kut ={:3d}, ostatak ={}'.format(n, kut, ostatak))

n =  3, kut =120, ostatak =0
n =  4, kut = 90, ostatak =0
n =  5, kut = 72, ostatak =0
n =  6, kut = 60, ostatak =0
n =  7, kut = 51, ostatak =3
n =  8, kut = 45, ostatak =0
n =  9, kut = 40, ostatak =0
n = 10, kut = 36, ostatak =0
n = 11, kut = 32, ostatak =8
n = 12, kut = 30, ostatak =0
>>>
```

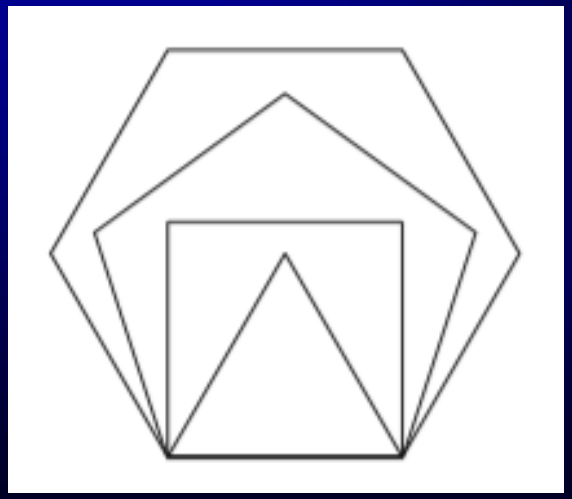
Uporaba funkcija nekog programa u interaktivnom sučelju

Nakon što neki program unutar kojeg smo definirali funkcije pokrenemo te ćemo funkcije moći pozivati u intreraktivnom sučelju.

Tako nakon izvođenja programa `program_3_8.py` možemo pisati:

```
>>> reset()
>>> for n in range(3, 7):
        pravilni_mnogokut(n, 150)

>>>ht()
```

i dobiti ćemo ovu sliku:

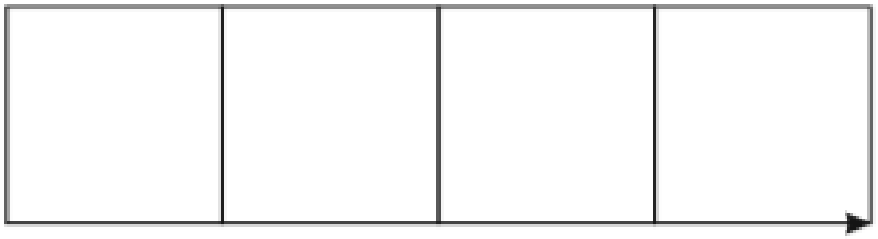


U interaktivnom sučelju možemo sada istraživati i razne druge mogućnosti uporabe funkcije koju smo definirali u našem programu. Primjerice, možemo jednostavno nacrtati niz kvadrata. Pogledajmo što će nacrtati sljedeći niz naredbi:

```
>>> reset()                                    #1
>>> pu(); bk(200); pd()                        #2
>>> for n in range(4):                         #3
        pravilni_mnogokut(4, 100)              #4
        fd(100)                                #5
```

Prije crtanja naredbom (#1) brišemo sve što je prije toga bilo nacrtano i postavljamo kornjaču u početni položaj i nakon toga naredbama (#2) pomičemo kornjaču ulijevo za 200 piksela. Počevši od te pozicije nacrtat će se četiri kvadrata, što je određeno naredbom (#3). U svakom će se prolazu petlje nakon crtanja kvadrata (#4) kornjača pomaknuti za duljinu stranice udesno (#5). S tog mjesta počinje crtanje kvadrata u sljedećem izvođenju petlje.

Izvođenjem ovoga niza naredbi dobili smo niz od četiri kvadrata, slika 3.12. Na slici se vidi i zadnji položaj kornjače.



Slika 3.12. *Niz od četiri kvadrata*

Dokumentacijski tekst (string) funkcije

U prethodnom poglavlju vidjeli smo da se prilikom upisa imena neke od standardnih funkcija npr. funkcija `input()` ili `print()` pojavljuje, u tekstualnom okviru, kratki opis njihova djelovanja. Taj će se opis pojaviti uvijek kada napišemo ime funkcije i otvorenu zagradu. Opis će nestati kada napišemo parametre funkcije i zatvorimo zagradu.

Na isti način možemo vidjeti i opise pojedinih funkcija iz modula `turtle`. U to se možemo uvjeriti nakon što u interaktivnom sučelju aktiviramo modul, ili izvedemo program u kojem se modul poziva. Pogledajmo:

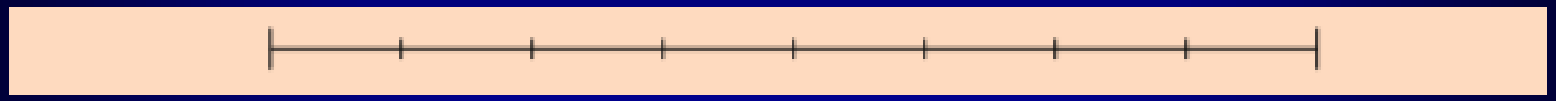
```
>>> from turtle import *
>>> fd(
       (distance)
       Move the turtle forward by the specified distance.
```

U opisu piše da će funkcija pomaknuti kornjaču naprijed za broj piksela koji ćemo napisati unutar zagrada.

Postavlja se pitanje možemo li za vlastite funkcije načiniti ovakve opise i kako? To je moguće postići ako napišemo prikladni opis za vlastitu funkciju i prikažemo ga u obliku tzv. **dokumentacijskog teksta**.

Primjer poželjnog izgleda programa

Pripremit ćemo program za crtanje dužina.



Jedinične dužine neka budu omeđene kraćim crticama, a krajeve dužine označit ćemo duljim crticama.

U modulu `turtle` nema funkcija za crtanje crtica i mi ćemo ju morati sami pripremiti.

Želimo zadavati broj piksela jedinične dužine i duljinu dužine (izraženu brojem jediničnih dužina).

Takav posao mogli bi obaviti sljedećim programom.

```
from turtle import *

def crtica(vel):                                                    #1
    '''Funkcija crta okomite crtice na mjestu na kojem
    se trenutno nalazi. Crtica će biti dugačka 2 * vel piksela.
    Preporuka: za kraće crtice odabrat ćemo vel = 3,
               za dulje crtice odabrat ćemo vel = 8.
    '''
    lt(90)                                                          #2
    fd(vel); bk(2 * vel); fd(vel)                                   #3
    rt(90)                                                          #4

def jedinična_dužina(m):                                            #5
    '''Funkcija crta dužinu duljine m piksela.
    Ta će dužina biti jedinična dužina.
    Na njezinu kraju nacrtat ćemo kraću crticu.
    '''
    fd(m)                                                           #6
    crtica(3)                                                       #7

def dužina(dulj, m):                                                #8
    '''Funkcija crta dužinu koja se sastoji od dulj jediničnih dužina.
          dulj - broj jediničnih dužina koje čine dužinu
          m - broj piksela jedinične dužine
    '''
    crtica(8)                                                       #9
    for i in range(dulj):                                           #10
        jedinična_dužina(m)                                         #11
    crtica(8)                                                       #12
```

```
def main():
    '''Funkcija main() najprije postavlja kornjaču u početni
    položaj 200 piksela lijevo od njezina središnjeg položaja
    i nakon toga traži upisivanje duljine dužine i broj
    piksela jedinične dužine i zatim u grafičkom prozoru
    crta zadanu dužinu pozivom funkcije dužina().
    '''
    home(); ht()                                                    #13
    pu(); bk(200); pd()                                             #14
    dulj = int(input('Duljina = '))
    m = int(input('Broj piksela jedinične dužine m = '))
    dužina(dulj, m)

main()
```

Program osim funkcije `main()` sadrži i sljedeće funkcije: `crtica()`, `jedinična_duži-na()` i `dužina()`. Dokumentacijski tekst svake funkcije opisuje djelovanje i način njezine uporabe.

Funkcija `crtica()` (#1) će na kraju svake dužine nacrtati crticu okomitu na dužinu. Ona to radi tako da najprije naredbom (#2) zakreće kornjaču za 90° ulijevo i zatim je naredbama (#3) pomiče naprijed, nazad i opet naprijed i tako je dovede u početni položaj. Nakon toga se naredbom (#4) kornjača zakreće u svoj početni smjer. Funkcija `crtica()` nam na neki način zamjenjuje funkciju `dot()` koju smo rabili u crtanju crte s točkama. Veličinu crtice određuje parametar `vel` kako je to opisano u dokumentacijskom tekstu funkcije, pri čemu je duljina crtice jednaka `2 * vel`.

Funkcija `dužina(a, m)` (#8) nacrtat će dužinu koja će se sastojati od `a` jediničnih dužina od kojih se svaka sastoji od `m` piksela. Na početku dužine naredbom (#9) pozvat će se funkcija `crtica()` koja će nacrtati okomitu dulju crticu i nakon toga će se u petlji `for` jedna za drugom iscrtati `a` jediničnih dužina i na kraju naredbom (#12) dulja okomita crtica. Uočimo da će funkcija `jedinična_dužina()`, kod zadnjeg izvođenja nacrtati prvo kraću okomitu crticu, no to nas ne treba smetati.

U funkciji `main()` prvo se u naredbi (#13) kornjača postavlja u središnji položaj i nakon toga naredbama (#14) se pomaknula za 200 piksela ulijevo. Tako smo kornjaču postavili u početnu točku iz koje će funkcija `dužina()` početi crtati zadanu dužinu.

Pogledamo li pažljivo dokumentacijske tekstove pojedinih funkcija, uočimo da nam gornji opis cijelog programa nije ni bio potreban – sve što je važno za razumijevanje djelovanja funkcija piše u dokumentacijskim tekstovima.

Pokretanjem tog programa pojavit će se u prozoru interaktivnog sučelja zahtjevi za upis duljine i broja piksela jedinične dužine koje moramo upisati. Ako smo utipkali sljedeće vrijednosti:

```
>>>
Duljina = 12
Broj piksela jedinične dužine m = 20
>>>
```

dobit ćemo u grafičkom prozoru crtež prikazan slikom 3.15.

Slika 3.15. *Dužina duljine 12*

Nakon izvođenja programa njegove funkcije možemo pozivati i iz interaktivnog sučelja.

Za pojedinu funkciju možemo vidjeti i njezin dokumentacijski tekst:

```
>>> crtica(
        (vel)
        Funkcija crte vertikalne crtice na mjestu na kojem
        se trenutno nalazi. Crtica će bit dugačka 2 * vel piksela.
        Preporuka: za kraće crtice odabrati vel = 3,
        za dulje crtice odabrati vel = 8.
```

Pogledajmo djelovanje sljedećeg niza naredbi:

```
>>> reset()                          #15
>>> pu(); bk(200); pd()              #16
>>> a, b, c = 5, 3, 7                #17
>>> m = 20                           #18
>>> dužina(a, m)                     #19
>>> dužina(b, m)                     #20
>>> dužina(c, m)                     #21
>>> ht()
```

Naredba (#15) izbrisat će crtež koji je nacrtao program, a naredbama (#16) kornjača je postavljena u početni položaj, 200 piksela ulijevo od njezina središnjeg položaja. U naredbi (#17) odredili smo duljine triju dužina koje želimo nacrtati i u naredbi (#18) varijabli `m` pripisali smo broj piksela jedinične dužine.

Funkcija `dužina()` počinje crtati dužinu počevši od položaja u kojem se nalazi pa dužina nacrtana naredbom (#19) počinje crtati od početnog položaja kornjače. Crtanje dužine naredbom (#20) počinje na kraju prve dužine tako da se one nadovezuju. Konačno, naredba (#21) nadovezat će i treću dužinu.

Duljina dužine dobivena nadovezivanjem dužina jednaka je zbroju njihovih duljina