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

08_A
Napredna uporaba modula `turtle`

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

Pregled osnovnih funkcija modula `turtle`

Funkcija	Alternativni naziv	Opis djelovanja funkcije
`reset()`		briše sve crteže u grafičkom prozoru; postavlja kornjaču u početni položaj s početnim atributima
`home()`		postavlja kornjaču u početni položaj
`clear()`		briše crtež u grafičkom prozoru; kornjača ostaje na istom mjestu s jednakim atributima
`forward(d)`	`fd(d)`	pomiče kornjaču za `d` jedinica unaprijed
`backward(d)`	`back(d)`, `bk(d)`	pomiče kornjaču za `d` jedinica unatrag
`right(kut)`	`rt(kut)`	zakreće kornjaču za `kut` stupnjeva udesno, odnosno za `kut` u smjeru kazaljke na satu
`left(kut)`	`lt(kut)`	zakreće kornjaču za `kut` stupnjeva ulijevo, odnosno za `kut` u obrnutom smjeru od kazaljke na satu
`penup()`	`pu()`, `up()`	kornjača se podiže te pri sljedećem pomicanju neće ostavljati trag
`pendown()`	`pd()`, `down()`	kornjača se spušta te će pri sljedećem pomicanju ostavljati trag
`hideturtle()`	`ht()`	kornjača će postati nevidljiva
`showturtle()`	`st()`	kornjača postaje vidljiva
`dot(vel)`		nacrtat će se točka veličine `vel` piksela ako se funkcija napiše bez parametra, tj. `dot()` točka će biti veličine 4 piksela
`shape()`		kornjača će promijeniti svoj početni oblik u odabrani oblik, kornjača standardno ima oblik strelice
`stamp()`		ostavlja kopiju kornjače na trenutačnoj poziciji

Funkcija goto()

Napišimo u interaktivnom sučelju funkciju koja će nacrtati točku u grafičkom prozoru sa zadanim koordinatama (x, y).

Ta bi funkcija mogla izgledati ovako:

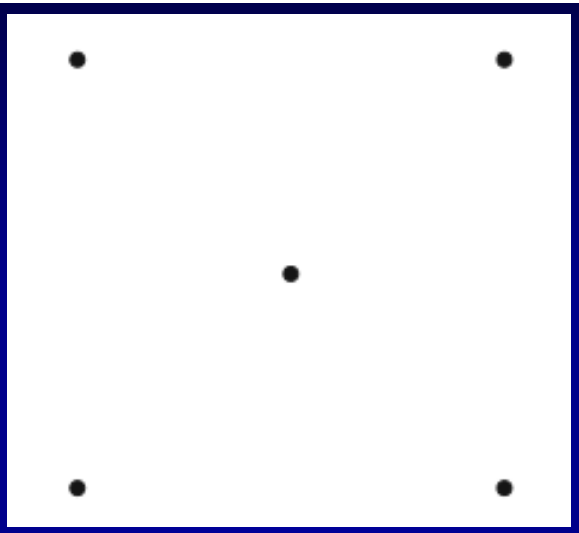
```
>>> from turtle import *
>>> def idi_u_točku(x, y):
        pu(); ht(); home()
        fd(x); lt(90); fd(y)
        dot(8)
```

Kako ne bismo vidjeli kretanje kornjače, primijenili smo funkciju `ht()`.

Pozovimo našu funkciju s nekoliko različitih vrijednosti:

```
>>> idi_u_točku(200, 200)
>>> idi_u_točku(-200, 200)
>>> idi_u_točku(-200, -200)
>>> idi_u_točku(200, -200)
>>> idi_u_točku(0, 0)
```

Pet točaka nacrtanih pozivanjem funkcije `idi_u_točku()`



U modulu `turtle` postoji niz funkcija pa tako i funkcija `goto()` kojom se možemo koristiti umjesto naše funkcije `idi_u_točku()`. Tu funkciju možemo pozivati na dva načina:

- dvama parametrima `goto(x,y)`, gdje su `x` i `y` koordinate točke u koju će se postaviti kornjača
- jednim parametrom `goto(t)`, gdje je `t` par brojeva, tj. `t = (x,y)`.

Ovdje treba naglasiti da se pomicanjem kornjače funkcijom `goto()` smjer kornjače ne mijenja – u novoj je točki jednak smjeru koji je kornjača imala u polaznoj točki.

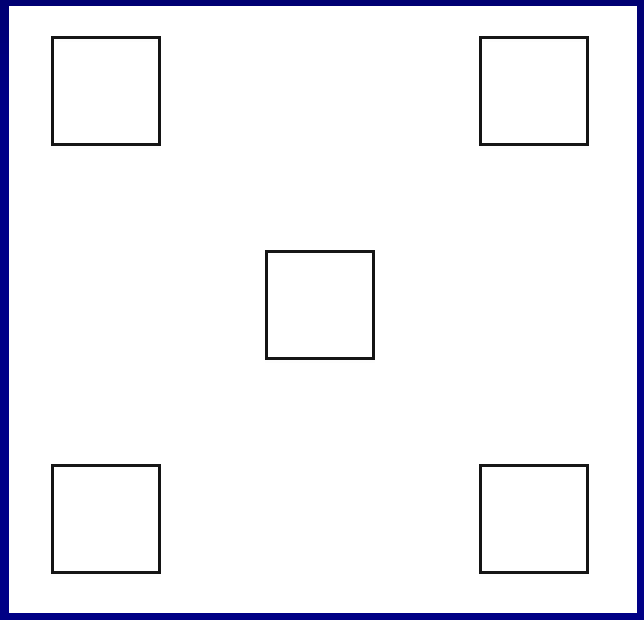
Primjer:

Napišimo program koji će unositi koordinate središta kvadrata (sjecišta dijagonala) te duljinu stranice kvadrata. Program treba nacrtati zadani kvadrat. Program napišimo u obliku modula.

Naš program će imati funkciju kvadrat čiji će ulazni parametri biti brojevi xs i ys koji određuju položaj središta kvadrata te duljina stranice kvadrata a. Na osnovi tih podataka definirat ćemo i listu (#4) vrhovi.

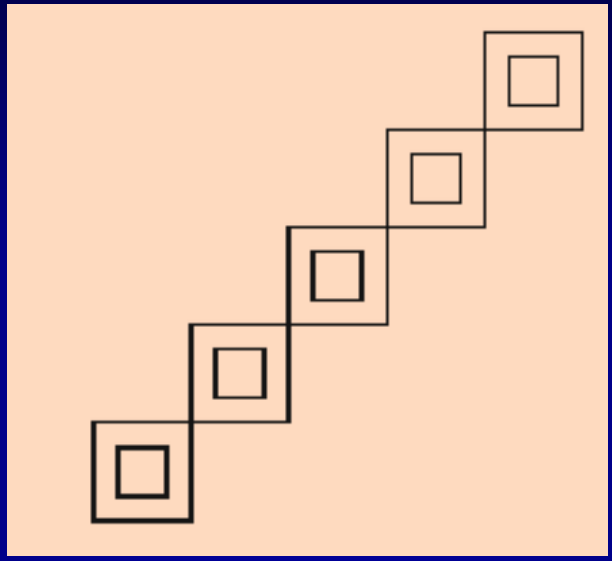
```
#Crtanje više kvadrata obilaskom vrhova - program_8_2.py

from turtle import *

def kvadrat(xs, ys, a):
    '''Funkcija crta kvadrat za koji je:
            a - duljina stranice
            xs, ys - koordinate sjecišta dijagonala
    '''
    pu(); ht()
    d = a // 2
    vrhovi = [(xs + d, ys + d),(xs - d, ys + d),\
              (xs - d, ys - d),(xs + d, ys - d)]                    #4
    goto(vrhovi[0])
    pd()
    for i in range(4):
        goto(vrhovi[(i+1) % 4])

if __name__ == '__main__':                                          #5
    kvadrat(0,0,50)
    kvadrat(100,100,50)
    kvadrat(-100,100,50)
    kvadrat(-100,-100,50)
    kvadrat(100,-100,50)
```

Pet kvadrata nacrtanih programom `program_8_2.py`



Primjer:

Napišimo program kojim ćemo nacrtati kvadrate na ovoj slici:



```
#Crtanje uzorka s kvadratima - program_8_3.py

from program_8_2 import *

reset()
x0 = -200
y0 = -200
for i in range(5):
    kvadrat(x0 + i * 100, y0 + i * 100, 100)                    #6
    kvadrat(x0 + i * 100, y0 + i * 100, 50)                     #7
```

Naredbom (#6) nacrtat ćemo kvadrat sa stranicom duljine 100, a naredbom (#7) kvadrat duljine stranice 50.

Primjer:

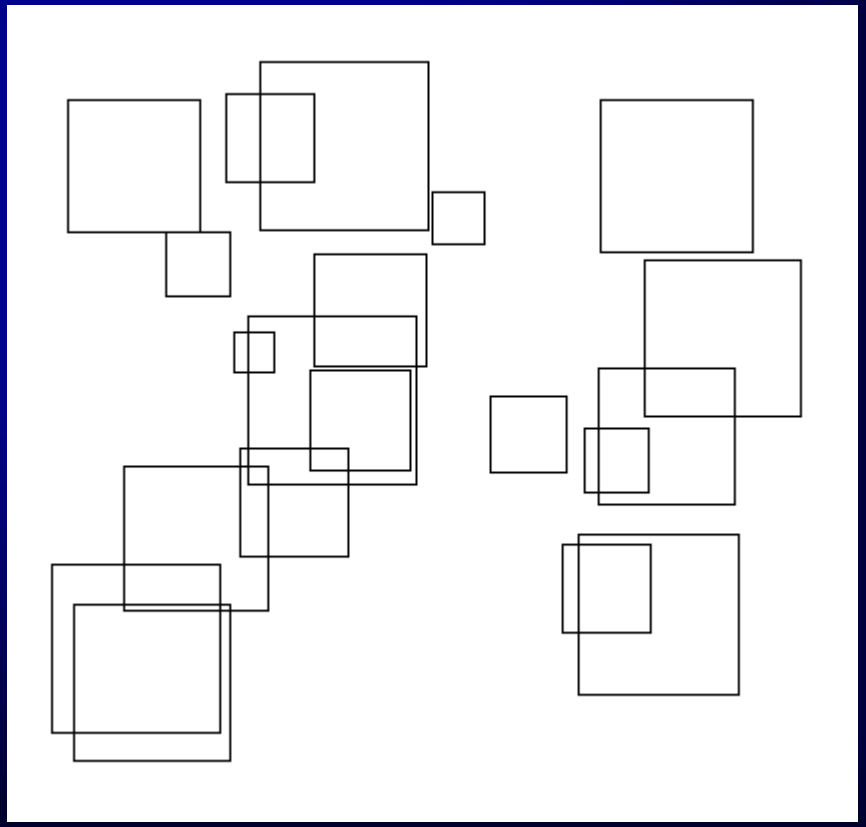
Nacrtajmo dvadeset nasumično razbacanih kvadrata s nasumično odabranim duljinama stranica.

Naučili smo da nasumične brojeve možemo generirati funkcijom `randint` *iz modula* `random`*. Ovdje nam ta funkcija može poslužiti za crtanje nasumično razbacanih kvadrata s nasumično odabranim duljinama stranica. Pri crtanju tih kvadrata koristit ćemo se i funkcijom* `kvadrat()` *koja je dio modula koji smo napravili u primjeru 8.2.*

Naš program koji će nacrtati naše nasumično razbacane kvadrate koji imaju nasumične duljine stranica može izgledati ovako:

```
#Crtanje nasumično smještenih kvadrata s nasumičnim duljinama
#stranice - program_8_4.py
from program_8_2 import *
from random import randint

reset()
for i in range(20):
    kvadrat(randint(-150, 150), randint(-150, 150), randint(10, 100))
```

Nakon što smo importirali potrebne funkcije izvođenjem petlje `for`, crtamo dvadeset kvadrata na nasumičnim pozicijama i s nasumičnim duljinama stranica. Izvođenjem navedenih naredbi možemo dobiti crtež kao na slici 8.5. On će zbog nasumičnosti odabira centara i duljina stranica pri svakom izvođenju biti drukčiji.



Funkcija `position()`

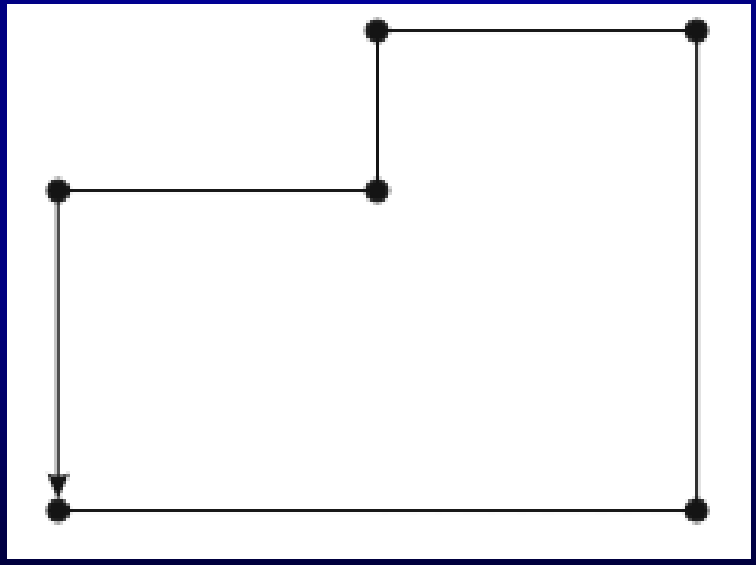
U modulu `turtle` postoji i funkcija `position()` (engl. *position* – položaj, pozicija) koja vraća par brojeva, odnosno koordinate točke u kojoj se kornjača trenutačno nalazi.

Ta će nam funkcija pomoći pri crtanju nekih likova. Pogledajmo u interaktivnom sučelju kako ona djeluje:

```
>>> reset()
>>> dot(8)
>>> position()
(0.00,0.00)
>>> fd(200); dot(8); lt(90)
>>> position()
(200.00,0.00)
>>> fd(150); dot(8); lt(90)
>>> position()
(200.00,150.00)
>>> fd(100); dot(8); lt(90)
>>> position()
(100.00,150.00)
>>> fd(100); dot(8); rt(90)
>>> position()
(100.00,50.00)
>>> fd(100); dot(8); lt(90)
>>> position()
(-0.00,50.00)
>>> fd(50)
>>> position()
(-0.00,0.00)
```

Nakon djelovanja funkcije `reset()` kornjača se nalazi u početnom položaju koji je određen parom brojeva (0, 0). Kao što vidimo funkcija `position()` vratit će taj položaj, ali u obliku decimalnog broja s dvjema decimalnim znamenkama. Decimalnim brojevima bavit ćemo se kasnije. Za sada ćemo promatrati samo cijele dijelove dobivenih brojeva. Uočimo da se u ispisu pojavila i vrijednost `-0.00` što ćemo protumačiti kao 0.00, odnosno 0.

Nakon svakog pomaka kornjače možemo se zapitati gdje se kornjača nalazi. Vidimo da se kornjača nalazila redom u točkama (0, 0), (200, 0), (200, 150), (100, 150), (100, 100), (0, 100) i (0, 0). Crtež koji je kornjača načinila na tom putu prikazan je slikom 8.6.



Tijekom izvođenja naredbi mogli bismo pohranjivati uzastopne položaje točaka u jednu listu npr. `tocke` i tako na neki način sačuvati crtež. Pogledajmo kako bismo to učinili.

```
>>> tocke = []                                                    #9
>>> reset(); tocke.append(position())                             #10
>>> fd(200); tocke.append(position()); lt(90)
>>> fd(150); tocke.append(position()); lt(90)
>>> fd(100); tocke.append(position()); lt(90)
>>> fd(50); tocke.append(position()); rt(90)
>>> fd(100); tocke.append(position()); lt(90)
>>> fd(100); tocke.append(position())
>>>
```

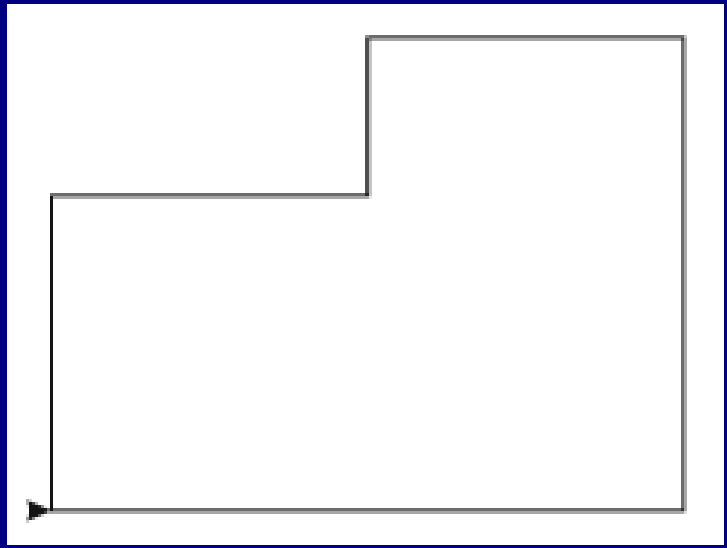
Nakon što smo naredbom (#9) definirali praznu listu `tocke` te sljedećim nizom naredba (#10) kornjaču ćemo dovesti u početni položaj i dodati u listu prvi par brojeva `(0,0)`. Slijedi niz naredbi kojima ćemo u listu `tocke` redom dodavati pozicije kornjače nakon pojedinog niza naredbi, uključujući i posljednji niz naredbi kojim se kornjača vraća u početni položaj.

Ispisom te liste možemo se uvjeriti da su pohranjene ispravne vrijednosti:

```
>>> tocke
[(0.00,0.00), (200.00,0.00), (200.00,150.00), (100.00,150.00), (100.00,100.00),
(-0.00,100.00), (-0.00,0.00)]
```

Sliku puta kornjače koja bi obišla točke iz navedene liste možemo dobiti tako da izvedemo sljedeće naredbe:

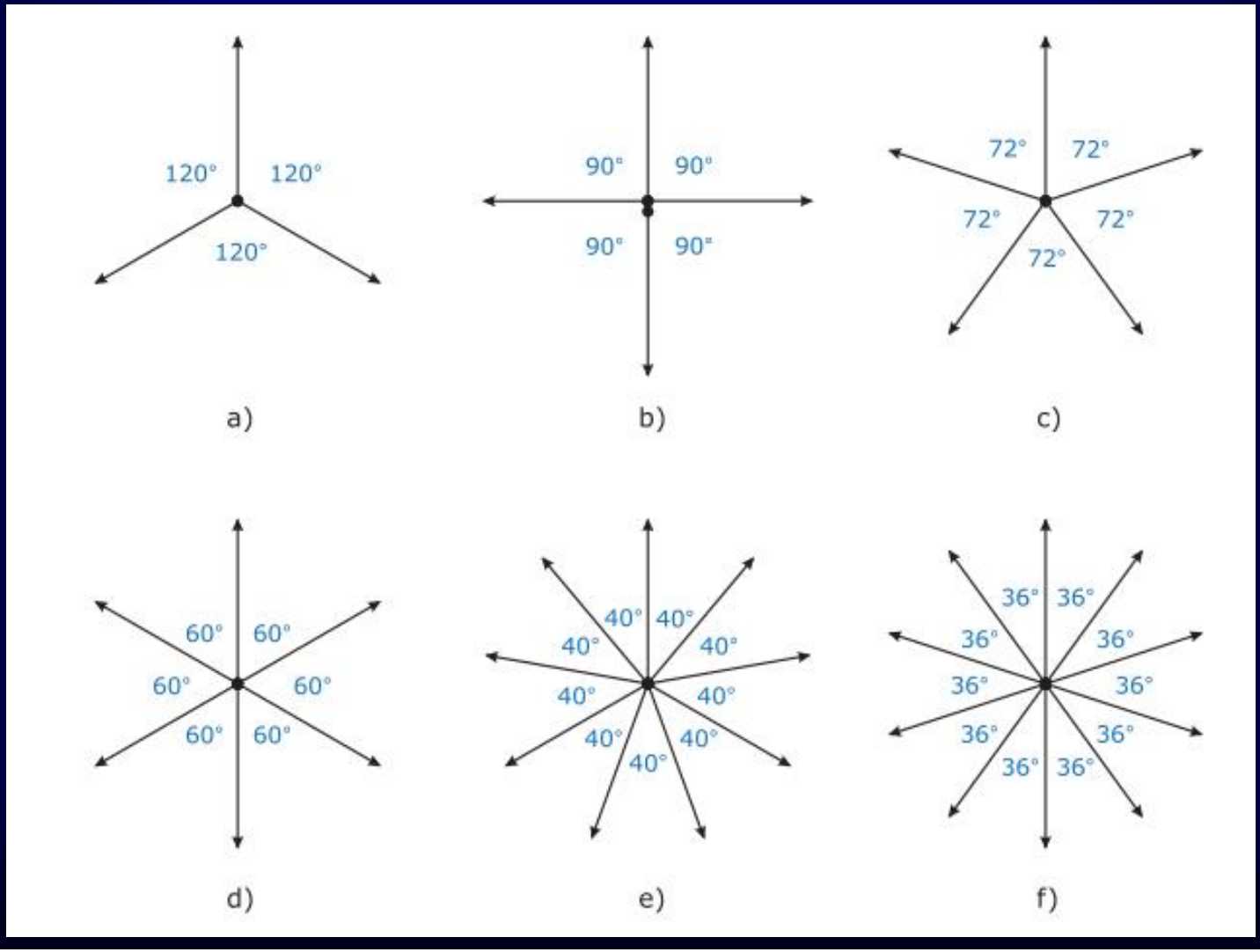
```
>>> reset()
>>> for t in točke:
        goto(t)
```



Uočimo da je završni smjer kornjače drugqačiji nego na prethodnoj slici. To je zbog toga jer funkcija `goto()` ne mijenja smjer kornjače.
Isto tako, nema ni točaka na svakoj poziciji jer u petlji nismo rabili funkciju `dot()`.

Funkcija `position()` omogućuje nam osmišljavanje drukčijeg načina crtanja pravilnih mnogokuta. Zamislimo da polupravac izlazi iz jedne točke te ga zakrećemo tako da opisuje puni krug. Podijelimo li puni krug na *n* jednakih dijelova, tada možemo na putu polupravca obilježiti *n* točaka koje bi mogle biti vrhovi pravilnog mnogokuta.

Slika 8.8. *Vrhovi pravilnih mnogokuta: a) trokut, b) četverokut, c) peterokut, d) šesterokut, e) deveterokut, f) deseterokut*



Napišimo program koji će crtati pravilne mnogokute tako da obilazi vrhove polupravaca prikazanih slikom 8.8.

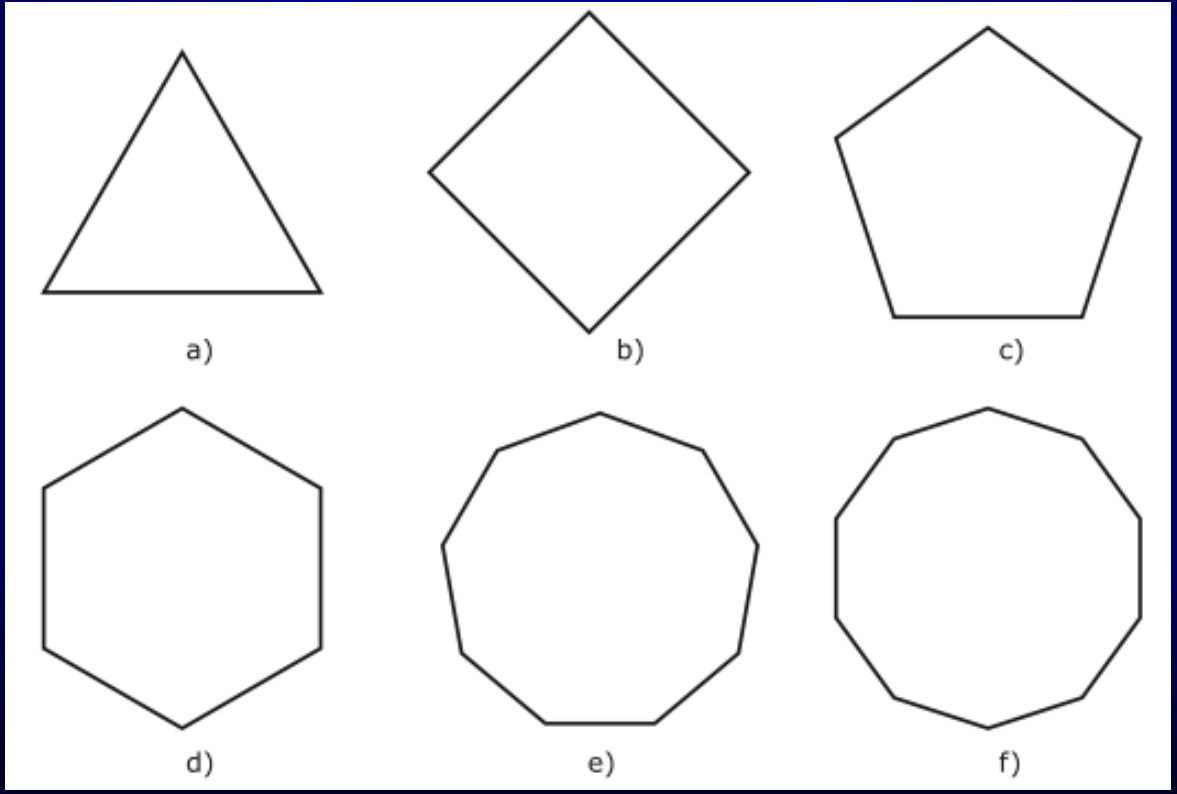
Prije no što pogledamo programsko rješenje, uočimo da su svi kutovi na crtežima na slici 8.8. prirodni brojevi. To nas može uputiti na zaključak da će naš program dobro crtati mnogokute za one vrijednosti `n` za koje je izračunan kut prirodni broj.

```
from turtle import *

def pravilni_mnogokut(n, r, xs, ys):
    '''Funkcija crta pravilni mnogokut za koji je:
            n - broj vrhova
            r - udaljenost vrhova od središta
            xs, ys - položaj središta
    '''
    pu(); ht()
    goto(xs, ys)
    setheading(90)                                          #11
    kut = 360 // n                                          #12
    vrhovi = []
    for i in range(n):                                      #13
        fd(r)
        vrhovi.append(position())
        bk(r)
        lt(kut)
    goto(vrhovi[0])
    pd()
    for i in range(n):                                      #14
        goto(vrhovi[(i+1) % n])

if __name__ == '__main__':
    pravilni_mnogokut(3, 100, -250, 150)
    pravilni_mnogokut(4, 100, 0, 150)
    pravilni_mnogokut(5, 100, 250, 150)
    pravilni_mnogokut(6, 100, -250, -150)
    pravilni_mnogokut(9, 100, 0, -150)
    pravilni_mnogokut(10, 100, 250, -150)
```

Nakon što smo u funkciji `pravilni_mnogokut()` podigli olovku, načinili je nevidljivom i postavili ju u središte mnogokuta naredbom (#11), odredili smo da će početni vrh biti postavljen u smjeru od 90°, tj. prema gore. Naredbom (#12) smo odredili veličinu kutova između polupravaca.

Slijedi naredba kojom smo uveli praznu listu `vrhovi`, koja će se popunjavati prolascima kroz petlju `for` (#13). S obzirom na to da je kornjača ostala podignuta, to se kretanje neće vidjeti u grafičkom prozoru. Nakon popunjavanja liste vrhovi, kornjača se postavlja na točku `vrhovi[0]` i tek se nakon spuštanja kornjače u petlji (#14) crtaju sve stranice mnogokuta.

Program šest puta poziva funkciju `pravilni_mnogokut()` i crta trokut, četverokut, peterokut, šesterokut, deveterokut i deseterokut kao što prikazuje slika 8.9.



Funkcije `speed()` *i* `tracer()`

Promatranje kretanja kornjače u grafičkom prozoru olakšava praćenje odvijanja dijela programa koji upravlja kretanjem kornjače. To nam olakšava pisanje i ispitivanje programa. Dio modula `turtle` koji nadzire brzinu kretanja kornjače naziva se `tracer` (engl. *trace* – slijediti trag, *tracer* – tragač). On ustvari jako usporava odvijanje programa kako bismo ljudskim okom mogli promatrati promjene u grafičkom prozoru.

Ako ne želimo gledati vizualizaciju crtanja našeg crteža, možemo ubrzati izvođenje našeg programa. To ćemo postići tako da isključimo dio modula turtle koji vizualizira kretanje kornjače.

Isključivanje se obavlja naredbom `tracer(False)`, a ponovno uključivanje naredbom `tracer(True)`. Prisjetimo se da vrijednost `0` tipa `int` odgovara vrijednosti `False` i da bilo koja druga vrijednost tipa `int` odgovara vrijednosti `True`, pa isključivanje odnosno uključivanje praćenja možemo obaviti i naredbom `tracer(0)` i `tracer(1)`.

Ako je praćenje uključeno onda se brzina kretanja može odrediti funkcijom `speed(brzina)`. Parametar `brzina` može poprimati vrijednosti iz intervala [1, 10], pri čemu vrijednost 1 određuje najmanju brzinu, a vrijednost 10 najveću brzinu kretanja kornjače.

Ako funkcije `tracer()` i `speed()` pozivamo s praznim zagradama, onda možemo ustanoviti u kakvom je stanju praćenje kornjače, tj. koje su trenutačne vrijednosti parametara. Kod pokretanja modula turtle praćenje je uključeno i brzina je postavljena na vrijednost 3 (što je umjerena brzina kretanja kornjače). Pogledajmo:

```
>>> from turtle import *
>>> tracer()
1
>>> speed()
3
```

Promjenu brzine kretanja kornjače možemo ilustrirati sljedećim primjerom:

```
>>> from turtle import *
>>> def deseterokut():
        for i in range(10):
                fd(100)
                lt(36)


>>> for k in range(1, 11):
        reset()
        speed(k)
        deseterokut()
```

Petlja `for` će crtati deseterokut deset puta i to počevši od najmanje brzine uz `k = 1` pa do najveće brzine uz `k = 10`.

S naredbom `tracer(False)` ili `tracer(0)` treba postupati pažljivo, jer se može dogoditi da crtež ne bude potpuno iscrtan ako se praćenje zaboravi ponovno uključiti. Zbog toga ga treba uvijek na kraju ponovno uključiti s `tracer(True)` ili s `tracer(1)`.

U programu `program_8_5.py` u funkciji `pravilni_nmnogokut()` na početku dodajmo naredbu `tracer(False)`, a na njezinu kraju naredbu `tracer(True)`. Nakon toga funkcija će izgledati ovako:

```
#Nadogradnja funkcije iz primjera 8.5.

def pravilni_mnogokut(n, r, xs,ys):
    '''Funkcija crta pravilni mnogokut za koji je:
            n - broj vrhova
            r - udaljenost vrhova od središta
            xs, ys - položaj središta
    '''
    tracer(False)
    pu(); ht()
    goto(xs, ys)
    setheading(90)
    kut = 360 // n
    vrhovi = []
    for i in range(n):
        fd(r)
        vrhovi.append(position())
        bk(r)
        lt(kut)
    goto(vrhovi[0])
    pd()
    for i in range(n):
        goto(vrhovi[(i + 1) % n])
    tracer(True)
```

Ovu ćemo mogućnost isključivanja animacije kretanja kornjače često rabiti u našim programima.

Funkcija za crtanje kružnice `circle()`

Napišimo program koji će crtati sve mnogokute za one brojeve kutova `n` za koje će biti `360 % n` jednako nuli.

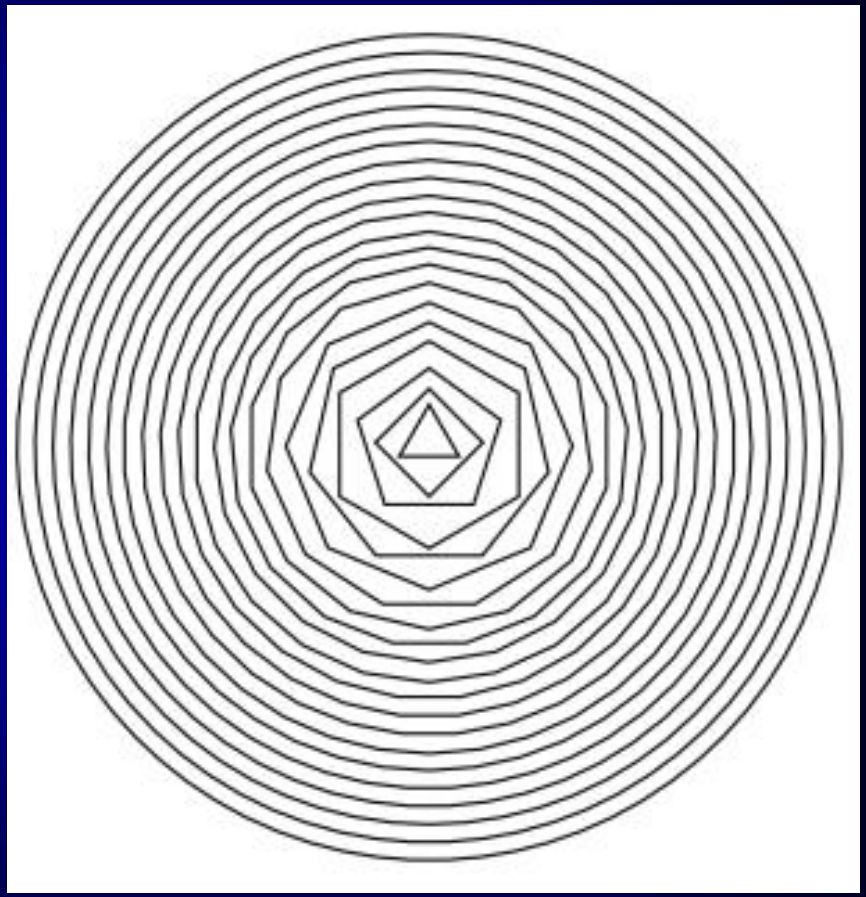
Pri pripremi rješenja ovog problema možemo uočiti da nema potrebe ponovno pisati funkciju `pravilni_mnogokuti()` koju smo definirali u programu `program_8_5.py`, posebice što je `program_8_5.py` pripremljen kao modul. Nakon što importiramo `program_8_5.py` funkcijom `pravilni_mnogokuti()` ćemo se bez ikakvog problema moći koristiti. Ostaje nam napraviti dodatnu funkciju `obitelj_mnogokuta()` koja će crtati niz mnogokuta.

Želimo crtati samo pravilne n-terokute za koje je `360 % n == 0`.

```
>>> n = [i for i in range(3, 361) if 360 % i == 0]
>>> n
[3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 30, 36, 40, 45, 60, 72, 90, 120, 180, 360]
```

```
#Crtanje obitelji mnogokuta - program_8_6.py

from program_8_5 import *

def obitelj_mnogokuta():
    n = [i for i in range(3, 361) if 360 % i == 0]
    r = [20 + 15 * j for j in range(len(n))]                    #1
    for k in range(len(n)):
        pravilni_mnogokut(n[k], r[k], 0, 0)

def main():
    obitelj_mnogokuta()

main()
```

Na već poznati način pripremili smo listu broja vrhova mnogokuta koji će biti iscrtani te smo naredbom (#1) pripremili listu udaljenosti vrhova od središta. Te se udaljenosti povećavaju s porastom indeksa. Slijedi petlja `for` u kojoj crtamo sve mnogokute tako da redom iz liste `n` uzimamo broj vrhova, a iz liste `r` udaljenosti vrhova od istog središta `(0, 0)` za sve mnogokute. Pokretanjem tog programa dobit ćemo crtež prikazan slikom 8.10.

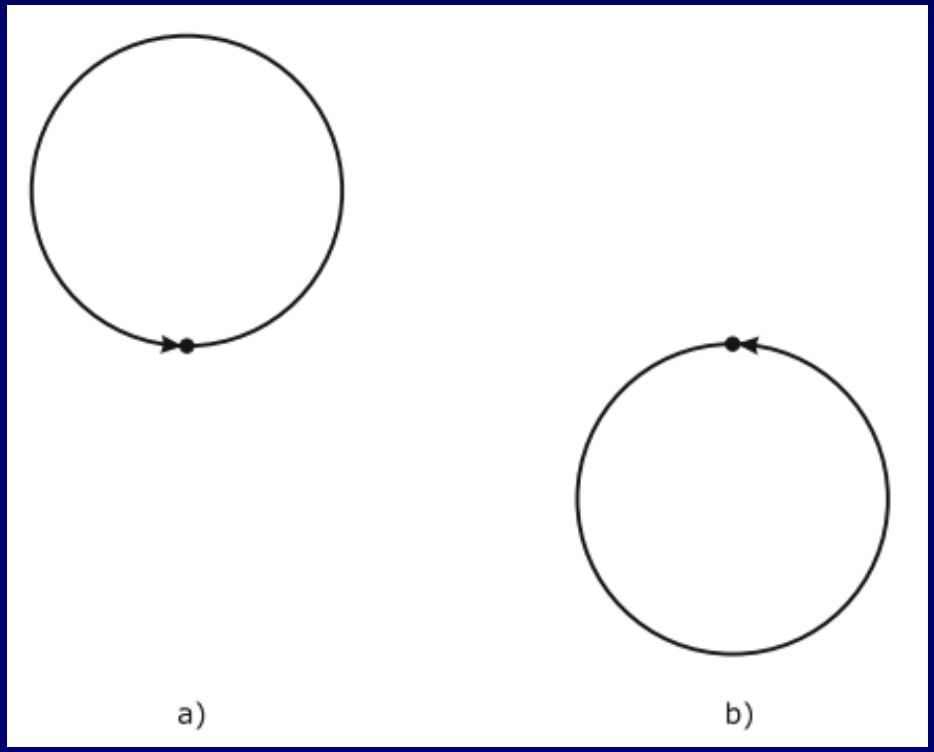


Pogledamo li sliku 8.10., vidjet ćemo da crteži desetak vanjskih mnogokuta izgledaju kao kružnice tj. možemo zaključiti da s porastom broja kutova crteži mnogokuta sve više sliče kružnicama.

Prema tome mogli bismo našu funkciju za crtanje mnogokuta rabiti i za crtanje kružnice.

No modul `turtle` ima svoju funkciju `circle()` koja crta kružnice. Napišemo li u interaktivnom sučelju sljedeće naredbe:

```
>>> reset()
>>> dot(8)
>>> circle(100)
```

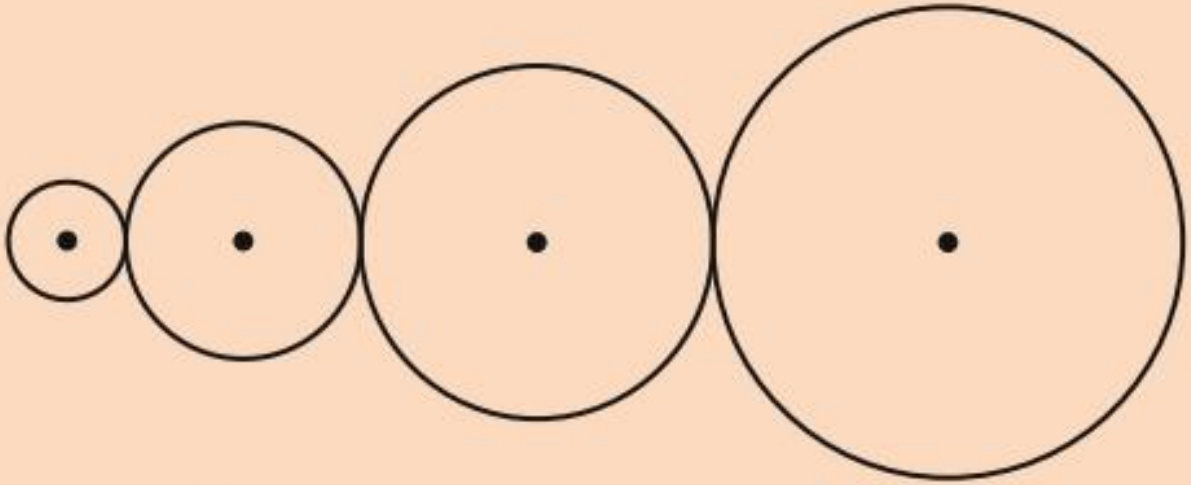


Ako prije crtanja kružnice kornjaču zakrenemo u suprotnom smjeru i nakon toga obavimo crtanje kružnice kao što je navedeno sljedećim nizom naredbi:

```
>>> reset()
>>> dot(8)
>>> lt(180)
>>> circle(100)
```

Primjer:

Napišimo program koji će crtati pet kružnica kao na slici 8.12. pri čemu prva kružnica ima polumjer 20, a svaka sljedeća kružnica ima polumjer za 20 veći od prethodne kružnice.



Slika 8.12. *Četiri kružnice koje treba nacrtati*

Pri rješavanju ovog primjera rabit ćemo funkciju `circle()`*. Uočimo da nam se pozicija središta kružnica mijenja te da se polumjeri kružnica povećavaju. Napravit ćemo funkciju* `kruznica()` *koja će crtati pojedinu kružnicu. Parametri te funkcije bit će polumjer i koordinate središta kružnice.*

```
#Crtanje kružnice - program_8_7.py

from turtle import *

def kružnica(r, xs, ys):
    '''Funkcija crta kružnicu gdje su:
            r - polumjer kružnice
            xs, ys - središte kružnice
    '''
    pu(); goto(xs + r, ys); pd()                                    #2
    setheading(90); ht()
    circle(r)

if __name__ == '__main__':
    pu(); goto(-180, 0); dot()
    kružnica(20, -180, 0)
    pu(); goto(-120, 0); dot()
    kružnica(40, -120, 0)
    pu(); goto(-20, 0); dot()
    kružnica(60, -20, 0)
    pu(); goto(120, 0); dot()
    kružnica(80, 120, 0)
```

Prvo pogledajmo našu funkciju `kruznica()` u kojoj smo kornjaču postavili desno od središta `xs`, `ys` na udaljenosti jednakoj polumjeru kružnice `r`. Nakon toga smo postavili smjer kornjače (**#2**) tako da ona *vidi* središte sa svoje lijeve strane upravo onako kako to zahtijeva funkcija `circle()`. U glavnom programu funkciju ćemo pozvati četiri puta te će se nacrtati četiri kružnice s nacrtanom točkom u njihovim središtima.

Crtanje kružnih lukova funkcijom `circle()`

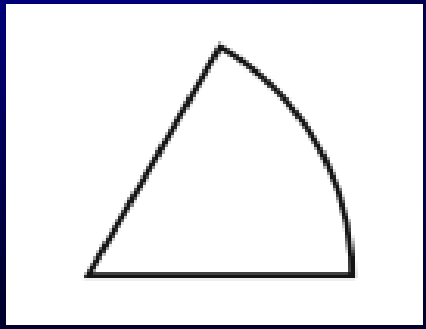
Funkcija `circle()` ima dva *skrivena*, ali ključna parametra. Ti parametri su:

- `extend` koji određuje kut kružnog luka koji će biti nacrtan (ako on nije naveden, funkcija crta kružni luk od 360°, tj. cijelu kružnicu)
- `steps j` koji određuje broj koraka u crtanju kružnice ili kružnog luka (ako se crta kružnica, onda je to broj vrhova mnogokuta koji će biti nacrtan, jer funkcija crta kružnicu kao mnogokut za koji sama određuje najpogodniji broj vrhova ako parametar `steps` nije zadan).

Napišimo u interaktivnom sučelju sljedeći niz naredbi:

```
>>> reset()
>>> r = 100
>>> kut = 60
>>> fd(r)                    #1
>>> lt(90)                   #2
>>> circle(r, kut)           #3
>>> lt(90)
>>> fd(r)                    #4
>>> ht()
```

Promatrajmo kako će taj niz naredbi nacrtati kružni isječak sa središnjim kutom od 60 stupnjeva

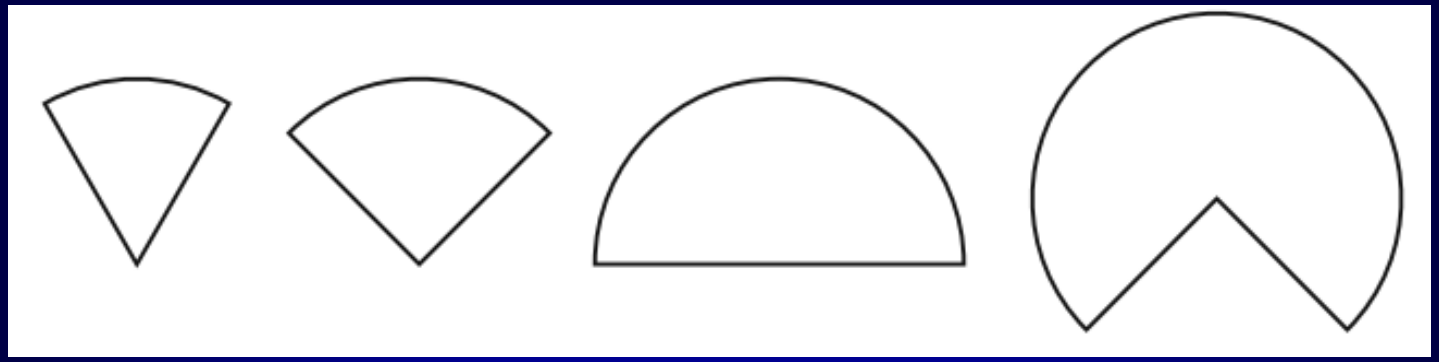


Primjer:

Napišimo program u obliku modula s funkcijom koja će crtati kružni isječak, a koja će imati kao parametre polumjer kružnog luka, koordinate ishodišta, početni kut za koji je isječak zakrenut i kut koji opisuje luk.

```
#Crtanje kružnog isječka - program_8_8.py

from turtle import *

def kružni_isječak(r, xs, ys, kutp, kutl):
    '''Funkcija crta kružni isječak gdje su:
            r - polumjer kružnog isječka
            xs, ys - koordinate ishodišta
            kutp - početni kut isječka
            kutl - središnji kut isječka
    '''
    pu(); goto(xs, ys); ht()
    seth(kutp)
    pd()
    fd(r); lt(90)
    circle(r, kutl)
    lt(90); fd(r)

if __name__ == '__main__':
    kružni_isječak(100, -120, 50, 60, 60)
    kružni_isječak(100, 120, 50, 45, 90)
    kružni_isječak(100, -120, -100, 0, 180)
    kružni_isječak(100, 120, -100, 315, 270)
```

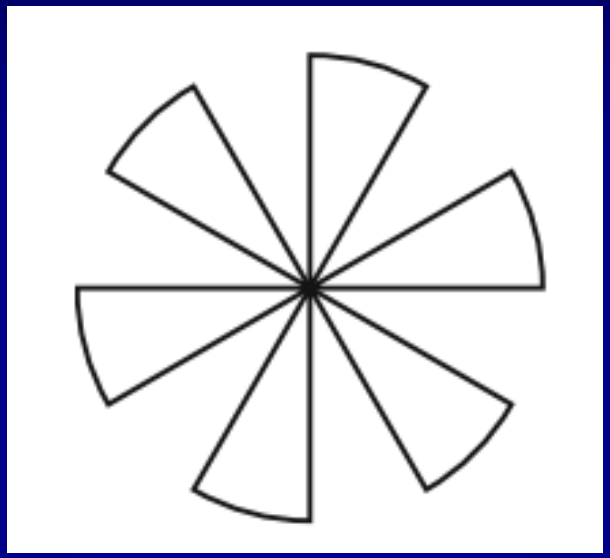
Kružni isječci nacrtani programom `program_8_6.py`



Kao što već znamo nakon izvođenja programa u interaktivnom sučelju možemo pozivati sve njegove funkcije pa i funkcije modula `turtle` koje je taj program importirao. Tako ćemo u interaktivnom sučelju moći pozvati i funkciju `kružni_isječak()`. Pogledajmo što će nam nacrtati sljedeći niz naredbi:

```
>>> reset()
>>> kut_početni = 0
>>> for i in range(6):
        kružni_isječak(150, 0, 0, kut_početni, 30)
        kut_početni += 60
```

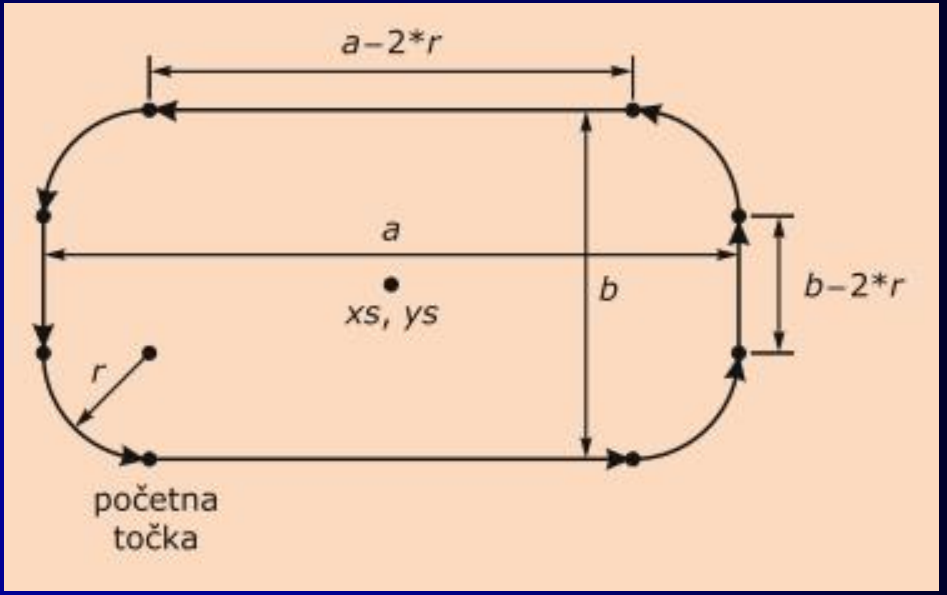
Nakon što naredbom `reset()` obrišemo prethodno nacrtane crteže, niz naredbi će nacrtati šest kružnih isječaka sa središnjim kutom od 30°, s tim da su im početni kutovi višekratnici kuta od 60°.



Primjer:

Napišimo funkciju za crtanje pravokutnika sa stranicama a *i* b *i sa zaobljenjima u vrhovima. Polumjer zaobljenja je* r. *Uz osnovne podatke o* pravokutniku *definiran je i položaj sjecišta zamišljenih dijagonala tj.* središte pravokutnika. *Skica pravokutnika prikazana je na slici 8.16.*

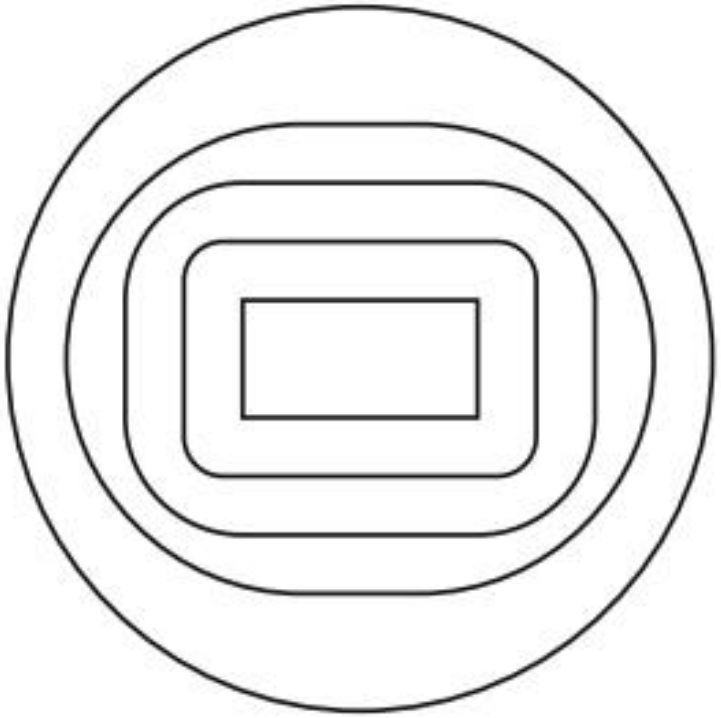
Slika 8.16. *Skica pravokutnika sa stranicama* a *i* b *i polumjerom zaobljenja* r *na vrhovima*



Pravokutnik ćemo najjednostavnije nacrtati tako da kornjaču postavimo u početnu točku, slika 8.16., i zatim ćemo obići opseg zaobljenog pravokutnika u smjeru strelica. To će nam raditi funkcija `zaobljeni_pravokutnik()`. *U toj ćemo se funkciji za crtanje kružnih lukova koristiti funkcijom* `circle()`. *Parametre naše funkcije opisat ćemo u dokumentacijskom stringu. Pravokutnik ćemo najjednostavnije nacrtati tako da kornjaču postavimo u početnu točku, nacrtamo vodoravnu stranicu, zatim kružni luk u smjeru strelica te potom okomitu stranicu i tako redom.*

```
#Crtanje zaobljenih pravokutnika - dio programa program_8_9.py

from turtle import *

def zaobljeni_pravokutnik(a, b, r, xs, ys):
    '''Funkcija crta pravokutnik sa zaobljenim uglovima.
       Parametri funkcije su:
          a - duljina vodoravne stranice pravokutnika
          b - duljina okomite stranice pravokutnika
          r - polumjer kružnog luka zaobljenog vrha
              (kada je r > min(a, b) funkcija će odabrati
               a = min(a, b) // 2)
          xs,ys - položaj središta pravokutnika
    '''
    if r >= min(a, b) // 2:
        r = min(a, b) // 2                                          #1
    pu(); goto(xs - a // 2 + r, ys - b // 2); pd()
    seth(0)
    for i in range(2):                                              #2
        fd(a - 2 * r)
        circle(r, 90)
        fd(b - 2 * r)
        circle(r, 90)
        ht()
if __name__ == '__main__':
    zaobljeni_pravokutnik(100, 50, 0, 0, 0)
    zaobljeni_pravokutnik(150, 100, 20, 0, 0)
    zaobljeni_pravokutnik(200, 150, 50, 0, 0)
    zaobljeni_pravokutnik(250, 200, 100, 0, 0)
    zaobljeni_pravokutnik(300, 300, 150, 0, 0)
```

Nakon izvođenja ovog programa dobit ćemo crtež kao na slici 8.17. Uočimo da svi pravokutnici imaju središte u `(0,0)`. Za prvi pravokutnik polumjer zaobljenja jednak je nuli te će funkcija nacrtati pravokutnik s oštrim vrhovima. Zanimljiv je i zadnji pravokutnik koji je zapravo kvadrat jer su mu zadane stranice jednakih duljina . Osim toga, polumjer zaobljenja u vrhovima jednak je polovini duljine stranice. Dakle, funkcija će zapravo nacrtati kružnicu.



Slika 8.17. *Pravokutnici sa zaobljenjima*

Funkcije za bojenje u modulu `turtle`

U modulu `turtle` postoje funkcije kojima se mogu izrađivati crteži u boji. Bojiti se mogu:

- pozadina grafičkog prozora
- trag olovke
- ispuna likova omeđenih zatvorenom crtom.

Mi nećemo izučavati sve moguće načine zadavanja boja već ćemo samo rabiti neke unaprijed pripremljene boje koje se odabiru njihovim engleskim imenima napisanim u obliku stringa.

Boja	String
crna	`'black'`
siva	`'gray'`
bijela	`'white'`
crvena	`'red'`
plava	`'blue'`
zelena	`'green'`
žuta	`'yellow'`
smeđa	`'brown'`
narančasta	`'orange'`
ružičasta	`'pink'`

Bojenje pozadine funkcijom `bgcolor()`

Boja pozadine postavlja se funkcijom `bgcolor(boja)`, gdje je boja string koji odabire jednu od boja. Ako se napiše prazna zagrada `bgcolor(boja)`, onda će nam funkcija vratiti trenutačnu boju pozadine.

Nakon otvaranja grafičkog prozora pozadina je bijele boje, što možemo vidjeti na sljedeći način:

```
>>> from turtle import *
>>> bgcolor()
'white'
```

Iz interaktivnog prozora možemo mijenjati boju pozadine ili pitati koje je boje pozadina:

```
>>> bgcolor('red')
>>> bgcolor()
'red'
```

Debljina i boja olovke, funkcije `pensize()` i `pencolor()`

Neposredno nakon importiranja modula `turtle` ili nakon poziva funkcije `reset()` debljina olovke iznosi jedan piksel te je boja pisanja crna. Boju olovke možemo promijeniti funkcijom `pencolor(boja)` gdje je boja string koji će odabrati određenu boju, a debljinu olovke odredit ćemo funkcijom `pensize(debljina)` gdje je debljina izražena u pikselima.

Želimo li provjeriti trenutačnu debljinu olovke, kao i boju pisanja olovke, trebamo navedene funkcije napisati bez parametara.

```
>>> reset()
>>> pensize()
1
>>> pencolor()
'black'
```

Pogledajmo što će se ispisati ako promijenimo debljinu olovke te boju pisanja:

```
>>> pensize(30)
>>> pensize()
30
>>> pencolor('red')
>>> pencolor()
'red'
>>> fd(100)
```

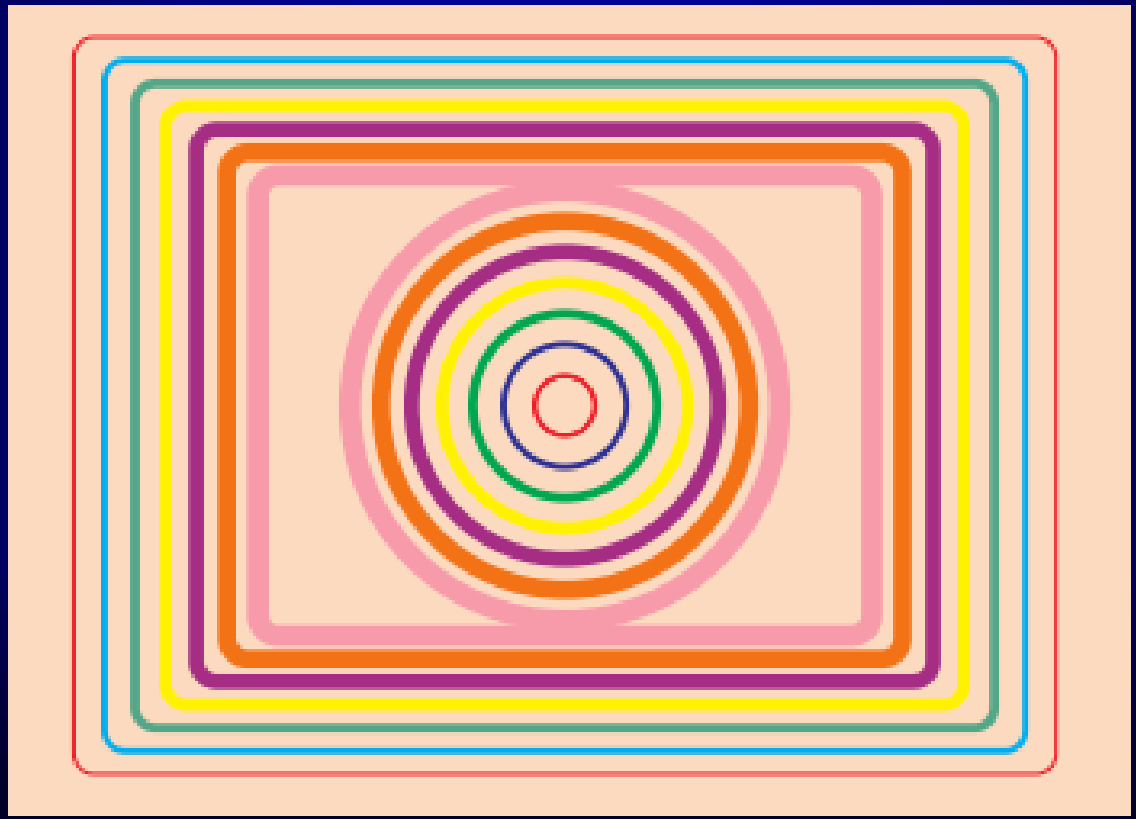
Učinak tih naredbi neće se vidjeti dok kornjača ne obavi neki pomak te smo kornjači zadali naredbu da se pomakne za 100 piksela. U grafičkom prozoru nacrtat će se crvena crta debljine 30 piksela.

Primjer:

Nacrtajmo sedam kružnica koje imaju središte u istoj točki, a svaka kružnica ima polumjer za 20 veći od polumjera prethodne kružnice. Prva kružnica ima polumjer 20. Kružnice trebaju biti obojene redom bojama kao na slici te debljina linije svake sljedeće kružnice teba biti za 2 piksela veća od debljine linije prethodne kružnice.

Nadalje nacrtajmo sedam zaobljenih pravokutnika od kojih prvi ima dimenzije 400 × 300, a svakom sljedećem je duljina vodoravne stranice za 40 veća od prethodne, a duljina okomite stranice za 30 veća od prethodne.

Boje linija pravokutnika te debljine trebaju biti u obrnutom redoslijedu od boja i debljina crta kod kružnica.



Za crtanje kružnica kao i za crtanje pravokutnika sa zaobljenim vrhovima možemo iskoristiti funkcije iz već postojećih modula:

- `kružnica()` *koja za razliku od ugrađene funkcije* `circle()`*, crta kružnicu sa zadanim koordinatama središta, a nalazi se u modulu* `program_8_7.py`
- `zaobljeni_pravokutnik()` *funkcija koja se nalazi u modulu* `program_8_9.py`*.*

Prisjetimo se da se radi jednostavnosti povezivanja svi ovi moduli moraju nalaziti u istoj mapi.

Uočimo da su boje kružnica kao i boje "pravokutnika" identične i da ih ima sedam, ali da su poredane u obrnutom redu. Zaključimo da možemo napraviti listu korištenih boja u glavnoj funkciji programa te da nam ta lista boja može biti ulazna vrijednost funkcije koja će crtati prvo koncentrične kružnice, a potom i naše "pravokutnike". Dakle, naša glavna funkcija programa će imati samo dvije naredbe: definiranje liste boje i poziv funkcije koju ćemo nazvati `kružnice_i_pravokutnici()`*. Osim što se mijenjaju boje crtanja, mijenjaju se i debljine olovke. Početnu debljinu olovke ćemo proizvoljno odrediti, kao i vrijednost za koju se podebljava trag olovke. Pogledajmo mogući izgled programa:*

```
#Crtanje kružnica i pravokutnika
#raznim bojama i debljinama olovke - program_8_11.py

from turtle import *
from program_8_7 import kružnica
from program_8_9 import zaobljeni_pravokutnik


def kružnice_i_pravokutnici(boje):
    r = 20
    debljina = 2
    n = len(boje)
    for i in range(n):                                          #1
        pencolor(boje[i])
        pensize((i + 1) * debljina)
        kružnica(r + i * 20, 0, 0)

    a = 400
    b = 300
    boje = boje[ : : -1]                                        #2
    for j in range(n):                                          #3
        pencolor(boje[j])                                       #4
        pensize((n - j - 1) * debljina)
        zaobljeni_pravokutnik(a + j * 40, b + j * 30, 20, 0, 0)

def main():
    boje = ['red', 'blue', 'green', 'yellow', 'brown', 'orange', 'pink']
    kružnice_i_pravokutnici(boje)

main()
```

Nakon što smo proizvoljno odredili početni polumjer kružnica, kao i početnu debljinu olovke, unutar petlje (#1) nacrtat ćemo za svaku od boja po jednu kružnicu. Sve kružnice imaju središte u `(0,0)`, a polumjer im se u svakom prolazu kroz petlju povećava za `20`. U svakom prolazu povećava se i debljina olovke i mijenja boje redom kako su smještene u listu boja.

Prije crtanja "pravokutnika" obrnut ćemo redoslijed boja u listi boje (#2) te ćemo unutar petlje (#3) crtati "pravokutnike". Početne duljine stranica `a` i `b` ponovno su proizvoljno zadane. Duljine stranica će se postupno povećavati, ali će se boje pojavljivati obrnutim redoslijedom (#3) te će se debljina olovke smanjivati. Uočimo da smo mogli izbjeći naredbu za stvaranje obrnute liste boja, ali bi nam onda naredba (#4) bila `pencolor(boje[n - j -1])`.

Primijetimo da smo se u ovom programu koristili glavnom funkcijom `main()` te samim time ovaj program nije pripremljen kao modul.

Uvijek kada mislimo da bi nam funkcije nekog programa mogle poslužiti i u nekim drugim programima razumno ih je pripremiti kao module.

Mi ćemo i dalje rabiti oba načina pripreme programa.

Bojenje likova, funkcije `fillcolor()`, `begin_fill()`, `end_fill()`, `color()`

Funkcija `fillcolor()` određuje boju kojom će se ispunjavati likovi nastali zatvorenom crtom. Samim odabirom boje ništa se neće dogoditi. Prije početka crtanja lika naredbom `begin_fill()` mora se označiti početak bojenja, a završetak crtanja lika treba pozvati funkciju `end_fill()`.

Pogledajmo sljedeći niz naredbi:

```
>>> from turtle import *
>>> fillcolor()
'black'
>>> begin_fill()                                    #1
>>> circle(100)
>>> end_fill()                                      #2
```

Nakon što smo importirali modul `turtle`, možemo provjeriti koja je boja početno postavljena kao boja ispune. To ćemo napraviti kao i u prijašnjim situacijama pozivom određene funkcije u ovom slučaju `fillcolor()` bez parametara. Vidimo da je postavljena boja crna.

Naredbom (#1) najavili smo početak crtanja lika koji trebamo ispuniti. Taj lik bit će omeđen kružnicom s polumjerom od 100 piksela. Funkcija `circle(100)` nacrtat će nam kružnicu koja neće biti ispunjena. Tek nakon što se izvede naredba (#2) kružnica će biti ispunjena crnom bojom i postat će crni krug.

Boju ispune možemo promijeniti pa ćemo nakon sljedećeg niza naredbi dobiti crveni krug:

```
>>> reset()
>>> fillcolor('red')
>>> begin_fill()
>>> circle(100)
>>> end_fill()
```

Pažljivim promatranjem vidimo da je kružnica koja omeđuje krug ostala crne boje. Naime, mi smo promijenili samo boju ispune. Ako želimo da i olovka promijeni boju, to treba načiniti funkcijom za promjenu boje olovke. Promjenu boje olovke i ispune ćemo dobiti naredbom `color(boja_olovke, boja_ispune)`.

Funkcija `color()` može se pozvati na tri načina:

- bez parametra – `color()`, pri čemu funkcija vraća imena dviju boja: boju olovke i boju ispune
- jednim parametrom – `color(boja)`, pri čemu funkcija određuje da boja olovke i boja ispune budu jednake
- dvama parametrima – `color(boja_a, boja_b)`, pri čemu olovka poprima boju `boja_a`, a boja ispune će biti `boja_b`.

Pogledajmo to u interaktivnom sučelju:

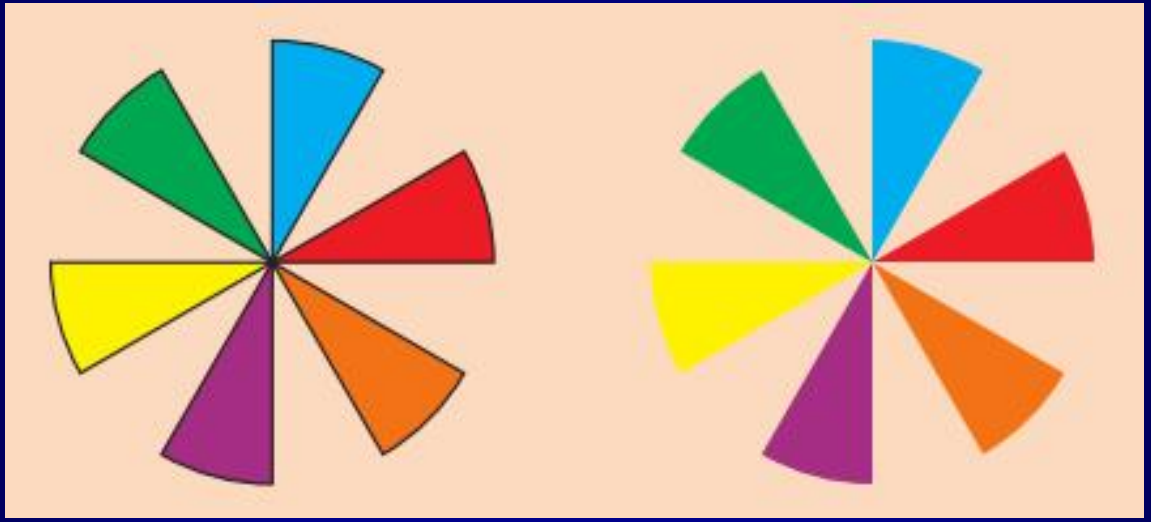
```
>>> reset()
>>> color()
('black', 'black')
>>>  pencolor()                                    #3
'black'
>>> fillcolor()                                   #4
'black'
>>> color('red', 'blue')
>>> color()
('red', 'blue')
```

Nakon što smo naredbom `reset()` između ostalog postavili i boju pisanja olovke i boju ispune u crnu boju, možemo to provjeriti funkcijom `color()` bez navođenja parametara. Vrijednosti parametara možemo provjeriti i posebno za svaki od tih parametara naredbama (#3) i (#4). Nakon što postavimo boju olovke i boju ispune (#3), možemo to ponovno provjeriti funkcijom `color()`.

Zaključimo da funkcije `pencolor()`, `fillcolor()` i `color()` možemo naizmjence rabiti za postavljanje i ispitivanje boja olovke i ispune.

Primjer:

Nacrtajmo dvije rozete kao na slici



Uočimo da su rozete sastavljene od šest kružnih isječaka s time da su isječci na lijevoj rozeti omeđeni crnim rubom dok na desnoj rozeti rubovi nisu posebno istaknuti. Kao što smo i u prošlom primjeru vidjeli da možemo za crtanje iskoristiti već prije napisane funkcije, tako i ovdje možemo preuzeti funkciju `kružni_isječak()` *iz modula* `program_8_8.py`*. Funkciju ćemo uobičajeno uvesti na samom početku programa. Pogledajmo mogući izgled programa:*

```
#Bojenje kružnih isječaka - program_8_12.py

from turtle import *
from program_8_8 import kružni_isječak

def obojana_roseta(x, y, olovka):
        boje = ['red', 'blue', 'green', 'yellow','brown', 'orange']
    kut_početni = 0
    pensize(3)
    n = len(boje)
    for i in range(n):
        fillcolor(boje[i])                                    #5
        if olovka == True:
            pencolor(boje[i])
            begin_fill()
            kružni_isječak(x, y, 0, kut_početni, 30)
            end_fill()
            kut_početni += 60

def main():
    obojena_roseta(150, -170, False)
    obojena_roseta(150, 170, True)

main()
```

Za debljinu olovke odabrali smo tri piksela tako da se trag olovke bolje vidi. Naredbom (#5) odabrali smo boju tako da svaki od isječaka bude druge boje. Lijevu rozetu nacrtat ćemo sa središtem u (-170,0) i ta rozeta će imati crne rubove, boju olovke nismo mijenjali. U drugoj petlji for u svakom prolazu kroz petlju mijenjat ćemo i boju olovke pa kružni isječci rozete sa središtem u (170,0) neće imati crne rubove oko isječaka.

Pisanje u kornjačinu prozoru

U modulu `turtle` postoji i funkcija kojom se u kornjačinu prozoru mogu ispisati znakovi – funkcija `write()`. Ta se funkcija znatno razlikuje od funkcije `print()` koja ispisuje nizove znakova u interaktivnom prozoru.

Funkcija `write()` prilagođena je svojoj osnovnoj svrsi – opisivanju crteža. Počinjemo li funkciju pisati u interaktivnom sučelju, pojavit će se njezin opis

```
>>> write(
        (arg, move=False, align='left', font=('Arial', 8, 'normal'))
Write text at the current turtle position.
```

Parametar	Značenje
`arg`	string koji će se ispisati kao i u funkciji `print()`, ova će funkcija ispisivati i brojeve tipa `int` u predviđenom obliku
`move`	pomicanje kornjače – ako je `move=False` kornjača se pri ispisivanju teksta neće pomicati, ako je `move=True` kornjača će se nakon ispisa pomaknuti na kraj ispisanog teksta
`align`	određuje smještanje teksta u odnosu na položaj kornjače: ▪ `'left'` – tekst se ispisuje tako da se njegov početak nalazi na mjestu položaja kornjače ▪ `'right'` – tekst se ispisuje tako da se kraj ispisanog teksta nalazi na mjestu položaja kornjače, ▪ `'center'` – tekst se ispisuje tako da će se sredina ispisanog teksta nalaziti na mjestu položaja kornjače
`font`	oblik i veličina znakova kojim će se tekst ispisivati

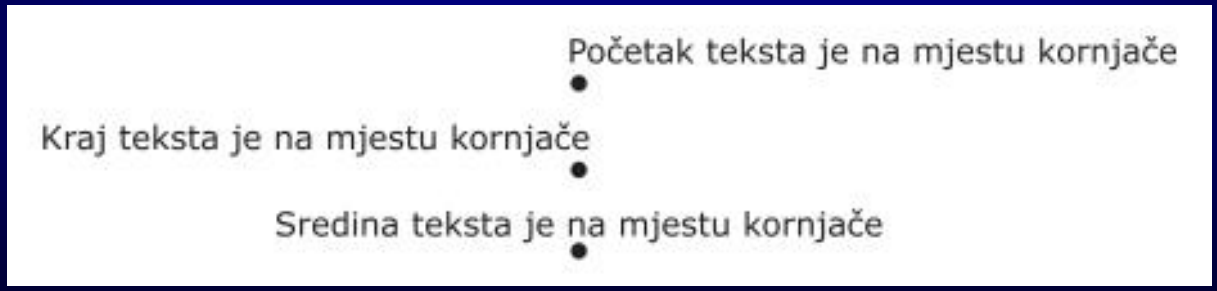
Parametar `align`

```
>>> reset(); pu()
>>> rt(90); dot()
>>> write('Početak teksta je na mjestu položaja kornjače')                #1
>>> fd(40); dot()
>>> write('Kraj teksta je na mjestu položaja kornjače', align='right')    #2
>>> fd(40); dot()
>>> write('Sredina teksta je na mjestu položaja kornjače', align='center')
>>> ht()
```

Nakon postavljanja početnih vrijednosti grafičkog prozora i podizanja kornjače, kornjaču ćemo usmjeriti prema dolje i trenutačni položaj označiti točkom.

Naredbom (#1) ispisat će se tekst uz zadani parametar `align='left'`. Nakon što kornjaču pomaknemo prema dolje za 40 piksela i nacrtamo točku, ispisujemo tekst uz `align='right'` (#2). I na kraju, nakon ponovnog pomicanja prema dolje uz `align='center'` ispisujemo zadani tekst.

Izgled ispisa koji ćemo dobiti izvođenjem navedenog niza naredbi

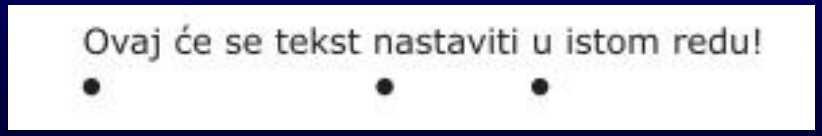


Parametar `move`

Parametar move određuje hoće li se kornjača pri ispisu pomicati. Pogledajmo što će ispisati sljedeći niz naredbi:

```
>>> reset(); pu()
>>> bk(100); dot()
>>> write('Ovaj će se tekst ', move=True)                    #3
>>> dot()
>>> write('nastaviti ', move=True)                           #4
>>> dot()
>>> write('u istom redu!')                                   #5
>>> ht()
```

Nakon niza naredbi kojima smo kornjaču premjestili za 100 piksela ulijevo i na toj poziciji nacrtali točku, naredba (#3) ispisat će zadani string i kornjača će se postaviti na mjesto iza zadnjeg ispisanog znaka. U string smo na kraj dodali jednu bjelinu tako da će naredba (#4) nastaviti pisati novu riječ nakon te bjeline. Isto tako, naredba (#5) će nastaviti ispis u istom retku. No, u tom zadnjem pozivu funkcije ne spominje se ni jedan parametar tako da se kornjača više neće pomicati. Dobiveni ispis u grafičkom prozoru izgleda kao na slici 8.26. Na njoj vidimo i položaje kornjače tijekom pisanja, a obilježili smo ih točkama.



Parametar `font`

Parametrom `font` određuju se tri vrijednosti: oblik slova, veličina slova i način pisanja slova.

Naziv `font` susreće se u svim programima za pisanje tekstova. U tim programima postoji veliki izbor oblika slova koje biramo njihovim imenima kao što su: `'Arial'`, `'Calibri'`, `'Comic Sans MC'`, `'Times New Roman'`. Oblik `'Arial'` se najčešće rabi pri opisu crteža i zbog toga je on i odabran kao font koji je unaprijed postavljen. Mi ćemo najčešće također rabiti taj oblik slova.

Veličina slova izražava se brojem piksela. Unaprijed je postavljena veličina slova od 8 piksela. Češće će nam trebati veća slova pa ćemo tu vrijednost morati mijenjati.

Slova se mogu ispisivati na četiri načina:

- `'normal'` – normalni način pisanja
- `'bold'` – pisanje podebljanim slovima
- `'italic'` – pisanje ukošenim slovima
- `'bold italic'` – pisanje podebljanim ukošenim slovima.

Način ispisivanja možemo ilustrirati sljedećim programom:

```
#Ispitivanje funkcije write - program_ispis_teksta.py

from turtle import *

def ispitivanje_funkcije_write():
    pu(); goto(-200,200); seth(270)
    write("Ispis uz font=('Arial',8,'normal')")                          #1
    fd(30)
    write("Ispis uz font=('Arial',10,'normal')",
          font=('Arial',10,'normal'))                                    #2
    fd(30)
    write("Ispis uz font=('Arial',12,'normal')", font=('Arial',12,'normal'))
    fd(30)
    write("Ispis uz font=('Arial',12,'bold')", font=('Arial',12,'bold'))
    fd(30)
    write("Ispis uz font=('Arial',12,'italic')",font=('Arial',12,'italic'))
    fd(30)
    write("Ispis uz font=('Arial',12,'bold italic')",\
          font=('Arial',12,'bold italic'))
    fd(30)
    write("Ispis uz font=('Comic Sans MS',12,'normal')",\
          font=('Comic Sans MS',12,'normal'))
    fd(30)
    write("Ispis uz font=('Comic Sans MS',12,'bold')",\
          font=('Comic Sans MS',12,'bold'))
    fd(30)
    write("Ispis uz font=('Comic Sans MS',12,'italic')",\
          font=('Comic Sans MS',12,'italic'))
    fd(30)
    write("Ispis uz font=('Comic Sans MS',12,'bold italic')",\
          font=('Comic Sans MS',12,'bold italic'))
    ht()

ispitivanje_funkcije_write()                                             #3
```

Prvo smo kornjaču doveli u početni položaj `(-200, 200)` i usmjerili je prema dolje kako bismo naredbama `fd()` nakon svakog ispisa mogli kornjaču postaviti na poziciju za ispis novog reda. Naredba (**#1**) ispisat će tekst s unaprijed postavljenim vrijednostima parametara. Uočimo da smo stringove koji su argumenti funkcije `write()` ogradili dvostrukim navodnicima kako bismo mogli unutar tog stringa rabiti jednostruke navodnike. U naredbi (**#2**), kao i sljedećim naredbama, uz tekst koji se ispisuje postavili smo i vrijednosti parametra font.

S obzirom na to da želimo samo ispitati te funkcije, nismo definirali funkciju `main()` već smo ju samo pokrenuli pozivom (**#3**). Ispis programa prikazan je na slici 8.27. Na početku zadnjeg reda ne vidi se zadnji položaj kornjače. Sakrili smo ga naredbom `ht()`.

Ispis uz font=('Arial',8,'normal')

Ispis uz font=('Arial',10,'normal')

Ispis uz font=('Arial',12,'normal')

Ispis uz font=('Arial',12,'bold')

Ispis uz font=('Arial',12,'italic')

Ispis uz font=('Arial',12,'bold italic')

Ispis uz font=('Comic Sans MS',12,'regular')

Ispis uz font=('Comic Sans MS',12,'bold')

Ispis uz font=('Comic Sans MS',12,'italic')

Ispis uz font=('Comic Sans MS',12,'bold italic')