DMS – Državni seminar 2018

Programski jezik Python i biblioteka Pygame

Ana Spasić
aspasic@matf.bg.ac.rs
Matematčki fakultet, Univerzitet u Beogradu

Vladimir Kuzmanović
vladimir_kuzmanovic@matf.bg.ac.rs
Matematički fakultet, Univerzitet u Beogradu

Milena Marić
milena.maric.f@gmail.com
Deveta gimnazija „Mihajlo Petrović – Alas“, Beograd

Februar, 2018.
Beograd

Uvod u Python

1. Primer ilustruje rad sa brojevima i operacije nad njima.

```
# operatori I operacije nad brojevima se ne razlikuju previse od C-ovskih
# definisemo 2 promenljive, x je celobrojna, a y realna
x = 3
y = 4.2

# stampamo tipove promenljivih
print( type(x) )
print( type(y) )

#stampa na ekran zbir dve promenljive
print(x + y)

# stampa na ekran vrednost y na x-ti stepen
print(y ** x)

# ceo deo kolicnika
print(y // x)

# realan kolicnik  realne i celobrojne promenljive
print(y / x)

# ostatak pri deljenju
print(y % x)

# eksplicitna promena tipa
# konverzija u int, odnosno u float
print( int(y) )
print( float(x) )

# podrska za kompleksne brojeve
z = complex(x, y)
# ili z = (3+4.2j)

print(z)
print( type(z) )

# sabiranje sa celobrojnom promenljivom
print(z + x)
# kompleksnim brojem
print(z + complex(1,x))
print (z + 20j)

# realan i imaginaran deo kompleksne promenljive z
print(z.real, z.imag)

# prikazujemo normu komplesnog broja
print(abs(z))  # sqrt(z.real**2 + z.imag**2)
```

Primer 1. Brojevi i osnovne operacije.

2. Primer ilustruje štampanje niski karaktera.

```
print( 'Hello world!'  )


print( "Zdravo svima!\
Nekada je lakse zapisati\
 u vise redova tekst, \
a da on bude ipak \
tretiran kao \
jedna linija \
teksta!\n")

print( "Mozemo \t koristiti \n specijalne karaktere\v!")

print( "Jel' moguce imati i ' u sredini" +" ogradjene sa \" ?")

# ako je niska ogradjena sa ' onda se " mogu koristiti unutar te niske,
# bez koriscenja \
print( 'levo,  a \t "ovo" je  desno' + " od reci 'levo'\n")


print( """Ukoliko je niska ogradjena sa
trostrukim navodnicima(ili apostrofima)
onda ce svaki
prelazak u novi red biti prikazan.\n""")

# raw string
print( r"ovo\tse ispisuje\"doslovno.\n")
```

Primer 2. Štampanje niski karaktera.

3. Primer ilustruje osnovna svojstva niski i operacije nad njima.

```
# definisemo promenljive tipa niska i stampamo ih na ekran
a = "Zdravo svima!\nDobro jutro!\n"

# stampamo tip promenljive a
print( type(a))
print( a )

b = "Pozdrav!"

print( b )

# niske se mogu nadovezivati
print( a + b + "\n\n")

# i same sa sobom zadat broj puta
print( b * 3 )
```

Primer 3a. Svojstva niski i operacije nad njima.

```
# pri cemu se originalna niska nece promeniti
print( b )


# karakteri stringa se ne mogu menjati
# a[0]='z'

# mogu se izdvojiti podniske koristeci indeksnu sintaksu
# indeksi karaktera pocinju od 0 za  prvi karakter i duzina-1 za poslednji
# karakter niske
# Python dozvoljava upotrebu pozitivnih i negativnih celih brojeva za
# indeksiranje
# Na primeru niske "Pozdrav!" :
#    P   o   z   d   r   a   v  !
#    0   1   2   3   4   5   6  7
#   -8  -7  -6  -5  -4  -3  -2 -1


print( b[2:6] )

# stampa podnisku od pocetka niske do karaktera na indeksu 2, ali bez
njega
print( b[:2] )

# stampa nisku od karaktera na indeksu 4 do kraja niske b
print( b[4:] )

# stampamo duzinu niske b
print( len(b) )

# stampamo sve karaktere niske b
print( b[:] )

# stampamo nisku od 3 karaktera do pretposlednjeg, neukljucujuci ga
print( b[2:-2] )

# stampamo nisku od 2 karaktera do poslednjeg, neukljucujuci ga
print( b[1:len(b)-1] )

# stampamo prvi karakter niske b
print( b[-0] )

# stampamo nisku koristeci samo negativne indekse
print( b[-5:-1] )

# moramo biti oprezni da se prvi indeks se odnosi na karakter koji je
# ranije u nisci od karaktera na koji se odnosi drugi indeks u intervalu
# inace cemo imati praznu nisku
print( b[-3: 1] )
print( b[4:2] )
```

Primer 4b. Svojstva niski i operacije nad njima.

4. Kolekcije u Python-u. Liste i ntorke.

```
# a je tipa ntorka navedenih elemenata (eng. tuple)
# moze se definisati I bez zagrada ()
a = (1,2,3,4,5)
print( a )
print(type(a))

# sa sekvencama mozemo sve sto smo mogli sa niskama
# I za njih vazi da ne mozemo menjati elemente, kao niskama karaktere

# kreiramo listu od sekvence( engl. tuple) u kojoj su nabrojani elementi
l = list((1,2,3,4,5))
# isto se postize i sledecom naredbom, jer je u zagradi ista sekvenca
# kao i a
l1 = list(a)

# prikazujemo listu l i za njom l1
print( l, l1 )

# kreiramo novu listu u kojoj nisu svi elementi celi brojevi
l2 = [7.2, 4, 'seminar', 'Python', 'programiranje']
print( l2 )

# menjamo element na indeksu 2
l2[2] = 'uvod'
print( l2 )

# stampamo rezultat nadovezivanja dve liste
print( l1 + l2 )


# Na liste mozemo primeniti indeksnu notaciju da izdvojimo pojedinacni
# element ili vise uzastopnih

# pravimo kopiju liste l1 i cuvamo je u promenljivu b
b = l1[:]
print( b )

# Zamenjujemo elemente liste b pocevsi od elementa sa indeksom 3 pa do
# kraja liste, praznom listom
# Drugim recima iz liste b uklanjamo sve pomenute elemente
b[3:] = []
print( b )

# l1 je nepromenjena jer smo radili sa kopijom
# Ukoliko u 32. redu sklonite [:] iz naredbe b = l1[:] primeticete
#razliku.
print( l1 )
```

Primer 4a. Kolekcije u Pythonu. Liste i ntorke.

```
# nadovezujemo b listu samom sobom i prikazujemo rezultat nadovezivanja.
# Lista b ostaje nepromenjena
print( b*2 )

# dodajemo na pocetak liste b listu od 2 elementa
b[:0]=['na','pocetak']
print( b )


# dodajemo u listu b izmedju 2. i 3. elementa listu od 2 elementa
b[2:3] = ["nesto","izmedju"]

# nadovezujemo listu b sa listom l2
# isto bi se moglo postici sledecom naredbom
# b = b + l2
b.extend(l2)
print( b )

# dodajemo na kraj liste nov element koji je lista l
b.append(l)
print( b )

# stampamo 1. element liste koja dodata na kraj liste b
print( b[-1][0])

# uklanjanje konkretnog elementa liste je moguce samo ako je element vec u
# listi inace cemo imati ValueError
# ispravno bi bilo
if 'pocetak' in b:
  b.remove('pocetak')
else:
  print("'pocetak' se ne nalazi u listi " + b)

# uklanjamo poslednji element liste i stampamo ga
print( b.pop() )
print( b )

#uklanjamo element na indeksu 0
print( b.pop(0) )
print( b )

# obrcemo listu b i stampamo je pre i posle te modifikacije
print( b )
b.reverse()
print( b )
```

Primer 4b. Kolekcije u Pythonu. Liste i ntorke.

```
# Sledece naredbe ponovo imaju smisla samo ako element vec postji u
# listi. Ukoliko element nije u listi, imacemo ValueError
# indeks elementa sa vrednoscu ‘uvod’
print( b.index('uvod') )
# broj pojavljivanja reci 'uvod' u listi b
print( b.count('uvod') )

# uklanjamo 4 elementa liste pocevsi od elementa na indeksu 3
b[3:7] = []
print( b )

# sortiramo listu b u opadajucem  poretku. Ona nakon brisanja sadrzi samo
# niske i poredjenje je leksikografsko
b.sort(reverse=True)
print( b )

# sortiramo listu niski po duzini opadajuce
# kada koristimo imena argumenata prilikom poziva funkcije nije bitan
#redosled
# key parametar je funkcija poredjenja koja ce se primeniti na tacno jedan
# element liste b
b.sort(key=len, reverse = True)
print( b )

# za slozenija poredjenja moze se koristiti lamda funkcija
b.sort(key=lambda x: len(x), reverse = True)
print( b )
```

Primer 4c. Kolekcije u Pythonu. Liste i ntorke.

5. Kolekcije u Pythonu. Liste i stringovi, iteracija.

```
# Iteriranje kroz stringove i liste
# Naredbe grananja i petlje
a="Moze i 0vako."
b=""

# na osnovu niske a formiramo nisku b tako sto svaki mali samoglasnik
# zamenjujemo velikim
for x in a:
  if x.islower() and x in ('a','e', 'i', 'o', 'u'):
    b = b + x.upper()
  elif not x.isalpha() :
    b = b +  '-'
  else :
    b += x

print( a )
print( b )
```

Primer 5a. Liste i stringovi, iteracija.

```
# zelimo da napisemo sve stepene broja 2 od 0. to 10.
# range(start, stop, step)
# funkcija kreira objekat koji generise sekvencu celih brojeva pocevsi od
# celog broja start do broja stop sa celobrojnim korakom step
# range(0, 11, 1) je potrebno za ovaj zadatak
# pri cemu ako je startni ceo broj 0 on se moze izostaviti jer je to
# podrazumevana vrednost. Takodje ukoliko je step 1 moze se izostaviti
# range(0, 11,1)  <=> range(11)
for i in range(0, 11, 1):
  print( 2, "^", i, "=", 2**i, end = "\n" )
print()

print()

# stampanje svakog elementa liste u novom redu
l = ['ovde', 'tamo', 'negde', 'svuda']
for x in l:
  print( x  )

print()

# Ukoliko u iteracijam planiramo da menjamo listu, posebno da uticemo na
# njenu duzinu,
# bolje je ne iterirati kroz originalnu listu vec kroz kopiju.
for x in l[:] :
  l.insert(0,x)
# Ukloniti [:] iza l u uslovu ostanka u petlji i naci cemo se u
# beskonacnoj petlji jer svakom iteracijom produzavamo listu za jedan
# element.

print()

# Stampamo elemente liste razdvojene zarezima, koristeci indeksnu sintaksu
for i in range(len(l)):
  if i != len(l)-1:
    print( l[i], end= ", ")
  else:
    print( l[i], end= "\n\n")


# Nacin za kreiranje liste
l = [ a for a in range(10) ]
print( l )
```

Primer 5b. Liste i stringovi, iteracija.

6. Kolekcije u Pythonu. Skupovi.

```
# Skup kao struktura podataka
# bez ponavljanja elemenata

# s ce biti skup razlicitih karaktera reci abrakadabra
s = set("magija")
print( s )

b= ['magija', "pokus", 'abrakadabra', 'magija', "hokus" ]
# stampamo skup ciji elementi su elementi liste b
print( set(b) )

# ukoliko skup s sadrzi 'f' uklanjamo ga, ukoliko ne sadrzi dodajemo
# Za uklanjanje je neophodno da se proveri da li je element u skupu. Ako
uklanjamo 'f' pre provere da li je element skupa
# imacemo KeyError, u slucaju da nije element skupa
if 'f' not in s:
    s.add('f')
else:
    s.remove('f')

c = set("mudro")

print( "\n",s,"\n",c )
# presek skupova
print( s&c )
print( s.intersection(c) )

# unija skupova
print( s|c )
print( s.union(c) )

# razlika skupova
print( s-c )
print( s.difference(c) )

# simetricna razlika
print( s^c )
print( s.symmetric_difference(c))
```

Primer 6. Skupovi.

7. Kolekcije u Pythonu. Rečnici ili mape.

```
# Dictionaries (dict)
dnevnik = {'Pera': 3,'Mira': 4, 'Dejan': 2}

print( dnevnik )
print (type(dnevnik))
```

Primer 7a. Rečnici.

```
# key() i values () su metode koje vracaju listu kljuceva, odnosno
# vrednosti sacuvanih u mapi
print( dnevnik.keys() )
print(dnevnik.values() )
print("\n")

# stampamo sortirane kljuceve
print( sorted(dnevnik.keys()) )

# Peri menjamo ocenu na 5
# Ukoliko imamo Peru u dnevniku promenicemo mu ocenu,
# a da ne postoji unos za Peru ovom naredbom bismo ga dodali
dnevnik['Pera'] = 5
# Stampamo mapu
print( dnevnik, end="\n" )


# Metod get vraca vrednost u mapi za navedeni kljud ukoliko postoji
# Vratice None ako nema unosa sa navedenim kljucem
print( dnevnik.get('Sonja') )

print ("\n\n")

# proverimo da li je Sonja u dnevniku
# Stampamo vrednost ako smo sigurni da postoji, inace imamo KeyError
if 'Sonja'  not in dnevnik.keys():
    dnevnik['Sonja']= 3
else:
    print( dnevnik['Sonja'] , "\n")


# items je metod koji nam vraca objekat koji ce nam sukcesivno generisati
# parove kljuc vrednost iz mape
print( dnevnik.items())

# proveravamo da li Pera ima 1
print( ("Pera",1) in dnevnik.items() )

print( "\n\n" )

# ispis sadrzaja mape koriscenjem kljuca kao indeksa
for k in dnevnik.keys():
    print( k, dnevnik[k] )

print ("\n\n")

# ispis parova kljuc i odgovarajuca vrednost iz mape koriscenjem objekta
# dobijenog sa metodom items()
# k ce uvek uzimati vrednosti prvog elementa para, tj, kljuca, a v ce
# uzimati uvek vrednost drugog u paru, tj, vrednosti
for k, v in dnevnik.items():
    print( k, '\t-> ', v )
```

Primer 7b. Rečnici.

8. Program ispituje da li je uneti broj prost.

```
# koristimo math biblioteku zbog funkcije za kvadratni koren
import math

# proverava da li je broj n prost
# vraca  1 ako je prost
# inace vraca x < n takvo da je n = x * (n/x)

# Funkcija prost ima jedan argument sa podrazumevanom vrednosti 2.
# To omogucuje da se funkcija pozove i bez argumenata, i u tom slucaju n
# ce imati vrednost 2.
def prost(n = 2):
  if n == 2:
    return 1

  if n % 2 == 0:
    return 2

  for x in range(3, sqrt(n), 2):
    if n % x == 0:
      return x
      break
    # ova else grana se odnosi na for petlju, i izvrsava se samo ako se iz
    # petlje izaslo usled iscrpljivanja liste u uslovu ostanka u petlji
  else :
    return 1

try:
  # input ispisuje poruku na ekran i preuzima liniju teksta sa ulaza.
  # int taj tekst konvertuje u int, ukoliko imamo int u zapisu. Inace baca
  # izuzetak.
  n = int(input('Unesite jedan ceo broj veci od 1: '))
  # ukoliko nije unet trazen broj bacamo izuzetak
  if n <= -1 :
    raise ValueError
except ValueError:
  exit('Potrebno je uneti prirodan broj veci od 1')

x = prost(n)
if( x == 1) :
  print( str(n) + " je prost" )
else:
  print( n, " = ", x, " * ", n/x )
```

Primer 8. Prost broj.

9. Program štampa n-ti fibonačijev broj.

```
def fibonacci(n = 6):
  niz = []
  a, b = 0, 1
  while a < n:
    niz.append(a)
    a, b = b, a + b

  return niz

print( fibonacci(2000))
print( fibonacci() )
print( fibonacci )
```

Primer 9. Fibonačijevi brojevi.

10. Program pravi kopiju fajla. Naziv fajla I naziv kopije se šalju preko argumenata komandne linije.

```
# Napisati Python skript koji ce preko argumenata komandne linije moci da
# primi naziv datoteke koju ce prepisati u drugu datoteku. Ako je naziv
# datoteke koja se prepisuje npr. ulaz.txt prepisujemo
# njen sadrzaj u datoteku sa nazivom kopija_ulaz.txt.

# Za pristup argumentima komandne linije iz skripta neophodno je da
# ukljucimo zaglavlje sys
import sys

# stampamo listu argumenata, koja nista drugo nego lista niski
# prvi argument je uvek naziv skripta
print( sys.argv, "\n")

# ako imamo vise od 1 argumenta komandne linije, mozda imamo naziv
# datoteke koji bismo citali
# Ukoliko nemamo prekidamo rad programa uz navedenu poruku
if len(sys.argv) == 1:
  exit('Nedovoljan broj argumenata komandne linije!\n')

# stampamo listu argumenata, koja nista drugo nego lista niski
# prvi argument je uvek naziv skripta
print( sys.argv, "\n")

# ako imamo vise od 1 argumenta komandne linije, mozda imamo naziv
# datoteke koji bismo citali
# Ukoliko nemamo prekidamo rad programa uz navedenu poruku
if len(sys.argv) == 1:
  exit('Nedovoljan broj argumenata komandne linije!\n')
```

Primer 10a. Kopiranje fajla.

```
# Imamo argument i pokusavamo da otvorimo datoteku za citanje
# Neophodno je da proverimo da li je uspesno proslo otvaranje datoteke sa
# prosledjenim nazivom. Datoteka sa tim nazivom moze da ne postoji ili da
# nemamo pravo da je citamo. U tom slucaju bi
# pokusaj citanja takve datoteke rezultovao izuzetkom IOError.
try:
  # otvaramo datoteku sa nazivom koji nam je prosledjen kao argument,
  # sa namerom da je citamo.
  f = open(sys.argv[1], "r")
except IOError:
  exit("Neuspesno otvaranje datoteke " + sys.argv[1] )


# Datoteku koju smo uspesno otvorili mozemo uspesno citati na vise nacina
# liniju po liniju
# f.readline() vraca narednu liniju datoteke. Uzastopnim pozivanjem metoda
# procitacemo sve linije.
# f.readlines() vraca listu svih linija iz datoteke

# karakter po karakter ili u  blokovima od n karaktera ili u celosti
# f.read() vraca nisku koja je celokupni sadrzaj datoteke
# f.read(n) vraca narednih n karaktera sadrzaja datoteke

# citamo ceo sadrzaj
sadrzaj = f.read()

# datoteka nije vise potrebna i zatvaramo je
f.close()

# otvaramo datoteku sa nazivom koji nam je prosledjen kao argument, sa
# namerom da pisemo u nju. Ukoliko datoteka postoji prepisacemo njen stari
# sadrzaj, ukoliko ne postoji kreiracemo je.
# posto smo prethodnu datoteku vezanu za f zatvorili mozemo promenljivu f
# ponovo da koristimo
try:
  f = open("kopija_"+sys.argv[1], "w")
except IOError:
  exit("Neuspesno otvaranje datoteke " + "kopija_" + sys.argv[1] )

# Upisujemo sadrzaj
f.write(sadrzaj)

# zatvaramo datoteku
f.close()
```

Primer 10b. Kopiranje fajla.

11. Program pravi kopiju fajla. Naziv fajla I naziv kopije se šalju preko argumenata komandne linije.

```
# Napisati Python skript koji ce preko argumenata komandne linije moci da
# primi naziv datoteke koju ce prepisati u drugu datoteku. Ako je naziv
# datoteke koja se prepisuje, npr. ulaz.txt prepisujemo njen sadrzaj
# liniju po liniju numerisane u datoteku linije_ulaz.txt

# Za pristup argumentima komandne linije iz skripta neophodno je da
ukljucimo zaglavlje sys
import sys

# stampamo listu argumenata, koja nista drugo nego lista niski
# prvi argument je uvek naziv skripta
print( sys.argv, "\n")

# ako imamo vise od 1 argumenta komandne linije, mozda imamo naziv
# datoteke koji bismo citali. Ukoliko nemamo prekidamo rad programa uz
# navedenu poruku
if len(sys.argv) == 1:
  exit('Nedovoljan broj argumenata komandne linije!\n')


# Imamo argument i pokusavamo da otvorimo datoteku za citanje
# Neophodno je da proverimo da li je uspesno proslo otvaranje datoteke sa
# prosledjenim nazivom. Datoteka sa tim nazivom moze da ne postoji ili da
# nemamo pravo da je citamo. U tom slucaju bi pokusaj citanja takve
# datoteke rezultovao izuzetkom IOError.
try:
  # otvaramo datoteku sa nazivom koji nam je prosledjen kao argument, sa
  # namerom da je citamo. Ako cemo mali broj operacija obaviti sa
  # datotekom i zatvoriti onda taj deo mozemo objediniti u with blok
  # f je dostupno samo u bloku i odnosi se na otvorenu datoteku
  # prilikom napustanja with bloka datoteka ce biti automatski zatvorena i
  # nije neophodno da je ekspicitno zatvorimo
  with  open(sys.argv[1], "r") as f:
    # citamo ceo sadrzaj po linijama
    sadrzaj = f.readlines()
except IOError:
  exit("Neuspesno otvaranje datoteke " + sys.argv[1] )

# otvaramo datoteku sa nazivom koji nam je prosledjen kao argument, sa
# namerom da pisemo u nju. Ukoliko datoteka postoji prepisacemo njen stari
# sadrzaj, ukoliko ne postoji kreiracemo je. posto smo prethodnu datoteku
# vezanu za f zatvorili mozemo promenljivu f ponovo da koristimo
try:
  f = open("linije_"+sys.argv[1], "w")
except IOError:
  exit("Neuspesno otvaranje datoteke " + "linije_" + sys.argv[1] )
```

Primer 11a. Kopiranje fajla 2.

```
# Upisujemo sadrzaj
for i  in range( 0, len(sadrzaj)) :
  # Ako zelimo da koristimo + za konkatenaciji niski onda ceo broj i
moramo pretvoriti u njegovu reprezentaciju u vidu niske
  f.write( str(i) + ": " + sadrzaj[i])

# zatvaramo datoteku
f.close()
```

Primer 11b. Kopiranje fajla 2.

12. Formatiranje izlaza.

```
pitanja=['ime','zanimanje','mesto u kom zivim']
odgovori=['Pera','programer', 'Beograd']

print("\n\nI nacin\n")
for a,b in zip(pitanja,odgovori):
    print( 'Tvoje' + a + ' je: Moje' +a +' je '+b+'.' )

print("\n\nII nacin\n")
for a,b in zip(pitanja,odgovori):
    print( 'Tvoje {0} je: Moje {0} je {1:10}.'.format(a,b) )

print("\n\nIII nacin\n")
# drugi nacin za formatiranje izlaza
for a,b in zip(pitanja,odgovori):
    print( 'Moje %s je: Moje %s je %10s.' %(a, a, b) )

print("\n\nIV nacin\n")
for a,b in zip(pitanja,odgovori):
    print( 'Moje %(pit)s je: Moje %(pit)s je %(odg)10s.' % {"pit":a,
            "odg":b} )

print("\n\nV nacin\n")
    # drugi nacin za formatiranje izlaza
for a,b in zip(pitanja,odgovori):
    print( 'Moje {pitanje} je: Moje {pitanje} je {odgovor:20s}.'
           .format(pitanje = a, odgovor = b) )
```

Primer 12. Formatiranje izlaza.

NumPy biblioteka

1. Numpy.array tip podataka i osnovna svojstva.

```
import sys
# ukljucujemo biblioteku numpy
import numpy as np

# pravimo numPy niz na osnovu liste
a = np.array([1,2,3])
# obican python niz (array.array) je uvek jednodimenziona kolekcija
# dok numpy niz (numpy.array) podrzava proizvoljan broj dimenzija

# da bismo prikazali kreirani niz, koristimo funkciju print
print(a)

# ako hocemo da saznamo kog tipa je neka promenljiva koristimo funkciju
# type
print(type(a))
# ako hocemo da saznamo dimenziju niza, treba nam atribut 'shape'
# promenljive tipa numpy.array
print(a.shape)

# kao i klasicni nizovi, i numpy nizovi podrzavaju indeksni prisup
# indeksiranje pocinje od 0
print(a[0], a[1], a[2])
# nump.array je mutabilna kolekcija, tj. mozemo da menjamo njene elemente
a[0] = 5
# stampamo izmenjeni niz
print(a)

# visedimenzioni nizovi se kreiraju tako sto kreiramo listu cije je svaki
# element druga lista na primer, hocemo da kreriramo dvodimenzioni niz
b = np.array([[1, 2, 3], [4,5,6]]);

# da bismo se uverili da je zaista u pitanju dvodimenzioni niz, mozemo da
# saznamo njegovu dimenziju
print(b.shape)

# indeksiranje visedimenzionih nizova je takodje moguce, pri cemu
# neophodan broj indeksa odgovara broju dimenzija. npr. dvodimenzioni niz
# zahteva dva indeksa
print(b[0,0], b[0,1], b[1,0])

# pored rucnog kreiranja nizova, numpy podrzava i skup funkcija koje
# automatski kreiraju
# nizove zeljenih dimenzija sa unapred predefinisanim svojstvima

# slicno Matlabu, nula niz dimenzije 2x2 dobijamo sledecom naredbom
a = np.zeros((2,2))
# stampamo kreirani niz
print(a)
```

Primer 13a. Numpy.array tip i osnovna svojstva.

```
# niz jedinica dimenzije 3x2 dobijamo sledecom naredbom
b = np.ones((3,2))
# stampamo kreirani niz
print(b)


# konstantni niz dimenzije 2x3 dobijamo sledecom naredbom
c = np.full((3,2), 7)
# stampamo kreirani niz
print(c)

# jednicni niz dimenzije 3x3 dobijamo sledecom naredbom
d = np.eye(3)
# stampamo kreirani niz
print(d)

# niz slucajnih brojeva dimenzija 2x2 dobijamo sledecom naredbom
e = np.random.random((2,2))
# stampamo kreirani niz
print(e)

# jos jedan nacin za kreiranje nizova je i funkcija arange
# kojom se kreiraju sekvence
# opsti oblik poziva je arange(donja_granica, gornja_granica, korak)
# ukoliko se izostavi donja granica podrazumeva se da je 0
# ukoliko se izostavi korak podrazumeva se da je 1
f = np.arange(10)
print(f)

g = np.arange(5, 10)
print(g)

h = np.arange(5,10,2)
print(h)

# slicno Matlabu, moguce je kreirati proizvoljne ekvidistantne podele
# na unapred zadatom intervalu. za to se koristi funkcija linspace
print(np.linspace(1., 10., 20))
```

Primer 13b. Numpy.array tip i osnovna svojstva.

2. Iteracija, indeksiranje.

```
import sys
# ukljucujemo biblioteku numpy
import numpy as np

# kreiramo dvodimenzioni niz dimenzije 3x4
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

# kao i u slucaju stringova i obicnih nizova i sa numpy nizovima
# mozemo da koristimo 'slicing', odnosno izdvajanje podnizova
```

Primer 14a. Iteracija, indeksiranje.

```
# ukoliko zelimo da izdvojimo podniz koga cine prva i druga kolona
# i sve vrste manje od dva treba da napisemo sledeci izraz
b = a[:2, 1:3]

# prilikom indeksiranja na ovaj nacin, treba biti posebno
# pazljiv kada su granice u pitanju. opsti oblik indeksiranja
# je [min:max], pri cemu se min granica tretira kao >=, a max
# granica kao <. Jos jedna vazna napomena je da mora da vazi
# min <= max, jer u suprotnom izraz nece moci da se evaluira.
# npr. zapis [:2] znaci od pocetka do indeksa strogo manjeg od 2.
#       -> izdvajaju se samo 0 i 1
# npr. zapis [1:3] znaci od indeksa 1 do indeksa strogo manjeg od 3.
#       ->izdvajaju se 1 i 2

# kada ovaj zapis sklopimo u jedno indeksiranje oblika [:2, 1:3],
# ono se tumaci kao izvlacenje podniza sacinjenog od elemenata iz
# 0. i 1. vrste koji se nalaze u 1. i 2. koloni.

# prikazujemo dobijeni podniz
print(b)

# stampamo element u nizu a koji odgovara prvom elementu niza b
print(a[0,1])
# modifikujemo prvi element niza b
b[0,0] = 77
# stampamo ponovo prvi element niza a koji odgovara nizu b
print(a[0,1])

# BITNO: izdvajanje podniza na ovaj nacin ne kreira novi niz,
# vec samo kreira novi pogled na vec postojeci niz.
# Ovakvo ponasanje za sobom povlaci da bilo koja izmena izdvojenog
# podniza direktno menja polazni niz.
# Ako ste koritili Matlab ranije i navikli ste na njegov nacin rada,
# treba da budete posebno oprezni kod izvrsavanja ovakvih naredbi,
# jer se ponasaju skroz drugacije

# pored klasicnog indeksiranja numpy, podrzava i indeksiranje nizova
# drugim nizovima

# kreiramo novi niz
a = np.array([[1,2], [3, 4], [5, 6]])

# pristupamo elementima niza a, pomocu niza indeksa
print(a[[0, 1, 2], [0, 1, 0]])

# zapis [[0, 1, 2], [0, 1, 0]] se tumaci kao skup uredjenih parova
# indeksa [0,0], [1,1], [2,0]
# pa je ovo u stvari izdvajanje elemenata a[0,0], a[1,1], a[2,0]
# i njihovom smestanju u novi niz [a[0,0], a[1,1], a[2,0]]

# zapis iz prethodne naredbe je ekvivalentan zapisu
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

Primer 14b. Numpy.array tip i osnovna svojstva.

```
# na ovaj nacin mozemo vise puta da koristimo isti element iz niza
print(a[[0, 0], [1, 1]])
# sto je ekvivalentno sa
print(np.array([a[0, 1], a[0, 1]]))

# kreiramo novi niz
a = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])

# kreiramo niz indeksa
b = np.array([0,2,0,1])

# ako hocemo da prikazemo elemente iz svake vrste pomocu niza indeksa b,
# to mozemo da postignemo na sledeci nacin
print(a[np.arange(4), b])

# ako hocemo da izmenimo vrednost tih elemenata, to postizemo na slican
# nacin
a[np.arange(4), b] += 10

# stampamo izmenjeni niz
print(a)

# pored klasicnog numerickog indeksiranja, numpy nizovi podrzavaju i
# bulovsko indeksiranje

# kreiramo novi niz
a = np.array([[1,2], [3, 4], [5, 6]])

# kreiramo niz bulovskih indeksa koji nam govore na kojim mestima se u
# nizu a nalaze brojevi veci od 2
bool_idx = (a > 2)

# stampamo dobijeni niz indeksa
print(bool_idx)

# izdvajamo podniz elemenata koji zadovoljavaju uslov a > 2
print(a[bool_idx])

# sve ovo moguce je zapisati i u jednoj naredbi
print(a[a > 2])
```

Primer 14c. Numpy.array tip i osnovna svojstva.

3. Numpy tipovi podataka.

```
import sys
# ukljucujemo biblioteku numpy
import numpy as np

# automatsko odredjivanje tipova
x = np.array([1, 2])
print(x.dtype)

x = np.array([1.0, 2.0])
print(x.dtype)

# eksplicitno definisanje tipova
x = np.array([1, 2], dtype=np.int64)
print(x.dtype)
```

Primer 15. Numpy tipovi podataka.

4. Operacije sa numpy nizovima.

```
import sys
# ukljucujemo biblioteku numpy
import numpy as np

# pored definisanja tipova, numpy biblioteka podrzava
# gotovo sve uobicajene aritmeticke operacije nad numpy.array tipom

# definisemo dva niza koja cemo koristiti kao operande
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# sabiranje nizova je pokoordinatno i rezultat ce biti iste
# dimenzije kao i operandi. Operandi moraju biti iste dimenzije,
# inace se opracija ne moze primeniti i dobice se ValueError
print(x + y)
print(np.add(x, y))

# oduzimanje nizova je pokoordinatno i rezultat ce biti iste
# dimenzije kao i operandi. Operandi moraju biti iste dimenzije,
# inace se opracija ne moze primeniti i dobice se ValueError
print(x - y)
print(np.subtract(x, y))

# deljenje nizova je pokoordinatno i rezultat ce biti iste
# dimenzije kao i operandi. Operandi moraju biti iste dimenzije,
# inace se opracija ne moze primeniti i dobice se ValueError
print(x * y)
print(np.multiply(x, y))
```

Primer 16a. Operacije sa numpy nizovima.

```
# ako ste ranije koristili Matlab, ovo nije ocekivano ponasanje,
# pa povedite racuna kada mnozite numpy nizove

# deljenje nizova je pokoordinatno i rezultat ce biti iste
# dimenzije kao i operandi. Operandi moraju biti iste dimenzije,
# inace se opracija ne moze primeniti i dobice se ValueError
print(x / y)
print(np.divide(x, y))

# primena korena je takodje pokoordinatna
print(np.sqrt(x))
# u ovom slucaju vazno je da koristimo pravu verziju korena
# s obzirom da radimo sa tipom podataka numpy.array potrebno je da
# koristimo koren, tj sqrt, iz numpy paketa. klasicni koren iz math
# paketa se ne moze primeniti na numpy tipove podataka.

# pored klasicnih pokoordinatnih operacija, numpy podrzava i
# operacije nad matricama

#definisemo dve 2x2 matrice
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

# definisemo dva vektora
v = np.array([9,10])
w = np.array([11, 12])

# unutrasnji, tj. skalarni  proizvod dva vektora. rezultat je broj
print(v.dot(w))
print(np.dot(v, w))

# proizvod matrice i vektora. rezultat je vektor
print(x.dot(v))
print(np.dot(x, v))

# proizvod dve matrice. Rezultat je matrica
print(x.dot(y))
print(np.dot(x, y))

# ako ste ranije koristili Matlab, ovo nije ocekivano ponasanje,
# pa povedite racuna kada mnozite matrice. proizvod ne dobijate
# operatorom *, vec posebnom funkcijom dot

# matrice i vektore je moguce transponovati
print(x.T)
print(v.T)
# transponovanje vektora ne menja oblik polaznog vektora,
# tj. ne radi nista

# pored standardnih operacija sa nizovima, numpy podrzava i
# mnoge agregatne funkcije

# ako nam treba suma cele matrice, treba da koristimo funkciju sum
print(np.sum(x))
```

Primer 16b. Operacije sa numpy nizovima.

```
# sumu po kolonama ili vrstama dobijamo tako sto preciziramo po
# kojoj osi zelimo da se izvrsi sabiranje

# suma po kolonama
print(np.sum(x, axis=0))
# suma po vrstama
print(np.sum(x, axis=1))

# pored funkcije sum, korisne su jos operacije
# average - prosek
# mean - srednja vrednost
# std - standardna devijacija
# var - varijansa
# amax - maksimum
# amin - minimum
```

Primer 16c. Operacije sa numpy nizovima.

5. Dinamičko kreiranje numpy nizova.

```
import sys
# ukljucujemo biblioteku numpy
import numpy as np

# pored rucnog kreiranja vektora, mozemo i tokom izvrsavanja
# da im menjamo oblik i dimenzije

# kreiramo matricu
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
#kreiramo vektor
v = np.array([1, 0, 1])
# kreiramo praznu matricu koja ima iste dimenzije kao matrica x
y = np.empty_like(x)
# u petlji popunjvamo matricu y, tako sto na svaki red matrice
# x dodajemo vektor v
for i in range(4):
    y[i, :] = x[i, :] + v;

# prikazujemo dobijeni rezultat
print(y)
# drugi nacin kako mozemo da postignemo ovo isto je da 4 puta
# naslazemo vektor v samog na sebe. to postizemo sledecom naredbom
vv = np.tile(v, (4, 1));

# parametri (4,1) oznacavaju cetvorostruko nadovezivanje vrsta i
# jednostruko nadovezivanje kolona

#prikazujemo dobijenu matricu
print(vv)
# dodajemo matricu vv na polaznu matricu
y = x + vv
#stampamo rezultat
print(y)
```

Primer 17. Dinamičko kreiranje numpy nizova.

Lambda izrazi

1. Primer ilustruje upotrebu lambda izraza I anonimnih funkcija.

```
import sys
# da bismo koristili reduce moramo da ucitamo odgovarajucu biblioteku
from functools import reduce

# anonimne funkcije i lambda izrazi
# lamdba izrazi su mehanizam kojim se kreiraju mali parcici koda koji
# su potrebni samo u tom delu programa i nikada vise. Imajuci u vidu
# njihovu ogranicenu upotrebu besmisleno je davati im imena, te se stoga
# nazivaju jos i anonimne funkcije

# opsti oblik definisanja lambda izraza je sledeci
# lambda lista_argumenata : izraz

# pravimo jednostavnu anonimnu funkciju koja racuna zbir dva broja
# lambda izrazu cemo dodeliti ime koje cemo koristiti kasnije, iako to
# nije u duhu sa upotrebom anonimnih funkcija
sum = lambda x, y: x + y

# ukoliko hocemo da upotrebimo ovaj lambda izraz da bismo nasli zbir dva
# broja dovoljno je da napisemo sledece
zbir = sum(3,4)
# prikazujemo  dobijeni rezultat
print(zbir)

# prethodni primer je vestacki i lambda izrazi se ne koriste tako u
# realnim programima lambda izrazi se najcesce koriste u paru sa
# funkcijama map, filter i reduce

# funkcija map se koristi ako zelimo da primenimo istu operaciju na svaki
# element kolekcija na primer, zelimo da kvadriramo svaki element liste

# kreiramo listu
L = list(range(1, 10))
# stampamo kreiranu listu
print(L)

# kvadriramo svaki element liste
kvadrirani = list(map(lambda x: x**2, L))
# stampamo novu listu kvadriranih elemenata
print(kvadrirani)

# uvecavamo svaki element liste za 10
uvecani = list(map(lambda x: x + 10, kvadrirani))
# prikazujemo elemente
print(uvecani)

# definisemo listu temperatura
temperature = (36.5, 37.1, 37.5, 38.3, 39.7)
# prikazujemo temperature
print(temperature)
```

Primer 18a. Lambda izrazi I anonimne funkcije.

```
# pomocu lambda izraza prebacujemo temperature u farenhajte
farenhajti = list(map(lambda x: (9.0/5.0)*x + 32, temperature))
# prikazujemo rezultat
print(farenhajti)
# pomocu lambda izraza vracamo temperature u celzijuse
celzijusi = list(map(lambda x: (5.0/9.0)*(x - 32), farenhajti))
# prikazujemo rezultat
print(celzijusi)

# funkciju map i lambda izraze je moguce primeniti na vise od jedne liste
# jedini uslov je da liste budu jednake duzine

# definisemo tri liste
a = [1, 2, 3, 4]
b = [17, 12, 11, 10]
c = [-1, -4, 5, 9]
# izracunavamo zbir prve dve liste
zbir_a_b = list(map(lambda x,y:x+y, a,b))
# stampamo dobijeni rezultat
print(zbir_a_b)

# izracunavamo zbir sve tri liste
zbir_tri = list(map(lambda x,y,z:x+y+z, a,b,c))
# stampamo dobijeni rezultat
print(zbir_tri)

# izracunavamo malo slozeniji izraz
lista1 = list(map(lambda x,y,z : 2.5*x + 2*y - z, a,b,c))
# stampamo dobijeni rezultat
print(lista1)

# funkcija filter se koristi kada zelimo da izdvojimo elemente kolekcije
# koji zadovoljavaju neki uslov
# uslov se zadaje kao lambda izraz

# kreiramo listu prirodnih brojeva
L = list(range(1,20))
# stampamo kreiranu listu
print(L)

# iz lista iszdvajamo samo neparne elemente
neparni = list(filter(lambda x: x%2, L))
# stampamo dobijeni rezultat
print(neparni)

# iz list izdvajemo samo parne elemente
parni = list(filter(lambda x: x%2 == 0, L))
# stampamo dobijeni rezultat
print(parni)

# stampanje prostih brojeva
brojevi = list(range(2,50))
# filtriramo listu (Eratostenovo sito)
for i in range(2,8):
    brojevi = list(filter(lambda x: x == i or x % i, brojevi))
```

Primer 18b. Lambda izrazi I anonimne funkcije.

```
# algoritam kaze sledece:
# ostavi element u listi ako je jednak i ili ako podeljen sa i daje
# ostatak razlicit od 0 u suprotnom, ukloni ga iz liste

# stampamo dobijeni rezultat
print(list(brojevi))

# funkcija reduce se koristi kada zelimo da svedemo listu na neku skalarnu
# vrednost reduce radi tako sto sto kontinualno primenjuje funkciju na
# glavu i rep liste.

# ako zamislimo da imamo listu s oblika [s1, s2 ,s3, s4] i da zelimo da je
# redukujemo po nekoj funkciji f  sematski, primena funkcije f na
# redukovanje liste s bi tekla sledecim tokom

# prvo se izracunava vrednost funkcije za prva dva elementa
# f1 = f(s1, s2)
# a zatim se ta dobijena vrednost ukljucuje u izracunavanje sa sledecim
# elementom liste
# f2 = f(f1, s3) = f(f(s1, s2), s3)
# zatim se novi rezultat f2 ukljucuje u dalje izracunavanje
# f3 = f(f2, s4) = f(f(f(s1,s2),s3), s4)
# cim iskoristimo poslednji element, tada funkcija reduce prestaje sa
# radom

# imajuci ovo u vidu, mozemo da probamo da izracunam zbir elemenata u
# listi koriscenjem funkcije reduce

# kreiramo listu
S = list(range(1,100))
# da bismo napravili lambda izraz na pravi nacin, moramo da pazljivo
# definisemo ono sta zelimo da postignemo imajuci u vidu kako radi reduce

# reduce u svakom koraku primenjuje funkciju na stari zbir uzimajuci u
# obzir sledeci
# element u listi, pa bi lamdba izraz koji to opisuje treba da izgleda na
# sledeci nacin
# lambda x,y: x+y
# x je stari zbir, tj. zbir elemenata do i-tog
# y je taj i-ti element do kog smo stigli
# i rezultat lamdba izraza je zbir x+y, odnosno na tekuci zbir smo dodali
# novi sledeci element liste

# stampamo dobijeni zbir
print(reduce(lambda x,y:x+y, S))

# na slican nacin mozemo da probamo da nadjemo najveci element u listi

# definisemo lambda izraz koji opisuje veci od dva elementa
f = lambda a,b: a if (a>b) else b
# stampamo dobijeni rezultat
print(reduce(f, [47,11,42,102,13]))
```

Primer 18c. Lambda izrazi I anonimne funkcije.

```
# na osnovu primera sa zbirom brojeva lako mozemo da izracunamo proizvod,
# tj. faktorijel
print(reduce(lambda x,y:x*y, S))

# verovatnoca da izvucemo sedmicu na lotou
p = (1.0*reduce(lambda x,y:x*y, range(1,8)))
            /reduce(lambda x,y:x*y, range(43,50))
print(p)
print(1/p)
```

Primer 18d. Lambda izrazi I anonimne funkcije.

Matplotlib biblioteka

1. Primer crta grafik funkcija sinus I kosinus.

```
import numpy as np
# da bismo crtali grafike u pajtonu, potrebna nam je
# biblioteka matplotlib
from matplotlib import pyplot as plt

# matplotlib je izuetno slicna toolbox-u za crtanje u Matlabu
# gotovo sva podesavanja i opcije su ekvivalentni

# u ovom primeru hocemo da ncrtamo grafik funkcija sin(x) i cos(x)

# da bismo ncrtali grafik, prvo treba da preciziramo interval na kome
# crtamo i da definisemo podelu intervala

# podelu kreiramo ranije pomenutom funkcijom linspace iz paketa numpy
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
# zatim izracunavamo vrednosti funkcije u tackama podele
C, S = np.cos(X), np.sin(X)

# s obzirom da radimo sa numpy objektima, moramo da koristimo funkcije
# iz numpy biblioteke

# da bismo nacrtali grafik koristimo funkciju plot
# ciji je prvi argument nezavisno promenljiva, a drugi argument zavisno
# promenljiva

# crtamo funkciju kosinus
plt.plot(X, C)
# crtamo funkciju sinus
plt.plot(X, S)

# crtnaje funkcije i prikazivanje crteza korisniku nisu ista stvar
# ostaje samo jos da prikazemo crtez korisniku
plt.show()

# prikazani grafik je sa podrazumevanim podesavanjima
```

Primer 19. Grafik funkcija sinus i kosinus.

2. Formatiranje grafika.

```
import numpy as np
# da bismo crtali grafike u pajtonu, potrebna nam je
# biblioteka matplotlib
from matplotlib import pyplot as plt

# matplotlib je izuetno slicna toolbox-u za crtanje u Matlabu
# gotovo sva podesavanja i opcije su ekvivalentni

# u ovom primeru hocemo da ncrtamo grafik funkcija sin(x) i cos(x)
```

Primer 20a. Formatiranje grafika.

```
# da bismo ncrtali grafik, prvo treba da preciziramo interval na kome
# crtamo i da definisemo podelu intervala

# definisemo velicinu figure u incima i broj tacaka po incu
# num - broj figure
# facecolor - boja pozadine
# edgecolor - boja rama oko crteza
# frameon - prikzati ram, boolean
plt.figure(figsize=(8,6), dpi=80)

# podelu kreiramo ranije pomenutom funkcijom linspace iz paketa numpy
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
# zatim izracunavamo vrednosti funkcije u tackama podele
C, S = np.cos(X), np.sin(X)

# s obzirom da radimo sa numpy objektima, moramo da koristimo funkcije
# iz numpy biblioteke

# da bismo nacrtali grafik koristimo funkciju plot
# ciji je prvi argument nezavisno promenljiva, a drugi argument zavisno
# promenljiva

# crtamo funkciju kosinus plavom bojom sa linijom debljine 2.5
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-",
label="kosinus")
# crtamo funkciju sinus krsticima crvene boje
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-", label="sinus")

# postavljamo granice za prikazivanje grafika po x osi
plt.xlim(-4.0, 4.0)

# postavljamo granice za prikazivanje grafika po x osi
plt.ylim(-1.0, 1.0)

# da bismo postavili natpise na osama, potrebno je da uhvatimo trenutno
# aktivnu osu
ax = plt.gca()
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))


# obelezavamo tacaka na x osi
plt.xticks(np.linspace(-4,4,9, endpoint=True))

# obelezavamo tacaka na y osi
plt.yticks(np.linspace(-1,1,5,endpoint=True))

# drugi nacin za obelezavanje osa. umesto brojeva, mozemo da napisemo
# imena u latex formatu
# plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
#           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
# plt.yticks([-1, 0, +1], [r'$-1$', r'$0$', r'$+1$'])
```

Primer 20b. Formatiranje grafika.

```
# da bismo ubacili legendu, moramo uz svaki grafik da dodamo opciju label
# i da ga na taj nacin umenujemo

# prikazujemo legendu uz eksplicitno navodjenje lokacije
plt.legend(loc='upper left')

# matplotlib nam dozvoljava i da obelezavamo tacke od interesa
# pretpostavimo da nam je vazna tacka 2pi/3 i da hocemo da obelezimo
# ponasanje funkcija u toj tacki
t = 2*np.pi/3

# crtamo tacku (t, cos(t)) na grafiku
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')
# crtamo liniju koja spaja tacku i vrednost funkcije u tacki
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5, linestyle="-
-")
# potpisujemo ucrtanu tacku
plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2"))

# crtamo tacku (t, sin(t)) na grafiku
plt.scatter([t, ],[np.sin(t), ], 50, color='red')
# crtamo liniju koja spaja tacku i vrednost funkcije u tacki
plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5, linestyle="--
")

# potpisujemo ucrtanu tacku
plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2"))

# crtnaje funkcije i prikazivanje crteza korisniku nisu ista stvar
# ostaje samo jos da prikazemo crtez korisniku
plt.show()


# ako zelimo, mozemo i da sacuvamo nacrtani grafik
plt.savefig("grafik.png", dpi=72)
```

Primer 20c. Formatiranje grafika.

3. Višestruki grafici na istom crtežu, subplot.

```
import numpy as np
# da bismo crtali grafike u pajtonu, potrebna nam je
# biblioteka matplotlib
from matplotlib import pyplot as plt

# sve sto smo do sada crtali je prikazivalo jedan crtez na jednoj slici
# ponekad, zelimo da prikazemo vise crteza na jednoj figuri radi uporedne
# analize

# da bismo to postigli, treba da koristimo funkciju subplot
# kojom figuru delimo pna matricu figura

# na primer, zelimo da prikazemo sve osnovne trigonometrijske funkcije na
# jednoj figuri
# crtez jedan treba da prikaze funkciju sinus,
# crtez dva treba da prikaze funkciju kosinus,
# crtez tri treba da prikaze eksponencijalnu funkciju
# crtez cetiri treba da prikaze logaritamsku funkciju

# kao i u prethodnom primeru, prvo pravimo podelu intervala

# podelu kreiramo ranije pomenutom funkcijom linspace iz paketa numpy
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
# zatim izracunavamo vrednosti funkcije u tackama podele
C, S= np.cos(X), np.sin(X)

# racunamo novu podelu za druge dve funkcije
XX = np.linspace(0.01, 10, 1000, endpoint=True)
CT = np.log(XX)

XXX = np.linspace(-10, 5, 1000, endpoint=True)
T = np.exp(XXX)

# podesavamo velicinu figure
plt.figure(figsize=(8,6))

# trebaju nam cetiri crteza na istog figuri i organizovacemo ih kao
# matricu 2x2. polja u matrici se redom obelezavaju brojevima pocevsi od
# gornjeg levog polja u smeru s desna na levo.
# U slucaju matrice 2x2, numeracija polja bi bila sledeca
# 1 2
# 3 4

# u prvom polju zelimo da nacrtamo grafik funkcije sinus
# da bismo proglasili prvo polje kao aktivno, moramo da koristimo komandu
subplot
plt.subplot(2,2,1)
# sva formatiranja koja navedemo u nastavku odnosice se samo na prvo polje
na crtezu
```

Primer 21a. Subplot

```
# crtamo grafik
plt.plot(X, S, color="blue", linewidth=2.5, linestyle="-", label="sinus")
# ofarbamo pozitivni deo na grafiku
plt.fill_between(X, 0, S, S > 0, color='blue', alpha=.25)
# ofarbamo negativni deo na grafiku
plt.fill_between(X, S, 0, S < 0, color='red',  alpha=.25)

# postavljamo granice za prikazivanje grafika po x osi
plt.xlim(-4.0, 4.0)
# postavljamo granice za prikazivanje grafika po x osi
plt.ylim(-1.0, 1.0)
# obelezavamo tacaka na x osi
plt.xticks(np.linspace(-4,4,9, endpoint=True))
# obelezavamo tacaka na y osi
plt.yticks(np.linspace(-1,1,5,endpoint=True))
# prikazujemo legendu uz eksplicitno navodjenje lokacije
plt.legend(loc='upper left')
# prikazujemo mrezu
plt.grid()
# prikazujemo i ose
plt.axvline(0)
plt.axhline(0)

# na drugom polju crtamo funkciju kosinus

# proglsavamo drugo polje kao aktivno
plt.subplot(2,2,2)
# sva formatiranja koja navedemo u nastavku odnosice se samo na prvo polje
# na crtezu

# crtamo grafik
plt.plot(X, C, color="red", linewidth=2.5, linestyle="-", label="kosinus")
# postavljamo granice za prikazivanje grafika po x osi
plt.xlim(-4.0, 4.0)
# postavljamo granice za prikazivanje grafika po x osi
plt.ylim(-1.0, 1.0)
# obelezavamo tacaka na x osi
plt.xticks(np.linspace(-4,4,9, endpoint=True))
# obelezavamo tacaka na y osi
plt.yticks(np.linspace(-1,1,5,endpoint=True))
# prikazujemo legendu uz eksplicitno navodjenje lokacije
plt.legend(loc='upper left')
# prikazujemo mrezu
plt.grid()
# prikazujemo i ose
plt.axvline(0)
plt.axhline(0)

# na trecem polju crtamo funkciju eksponencijalnu funkciju

# proglsavamo drugo polje kao aktivno
plt.subplot(2,2,3)
# sva formatiranja koja navedemo u nastavku odnosice se samo na prvo polje
# na crtezu
```

Primer 21b. Subplot

```
# crtamo grafik
plt.plot(XXX, T, color="green", linewidth=2.5, linestyle="-", label="exp")

# postavljamo granice za prikazivanje grafika po x osi
plt.xlim(-10, 5)
# postavljamo granice za prikazivanje grafika po x osi
plt.ylim(-0.5, 50)
# obelezavamo tacaka na x osi
plt.xticks(np.linspace(-10, 5, 5, endpoint=True))
# obelezavamo tacaka na y osi
plt.yticks(np.linspace(--05, 50, 10,endpoint=True))
# prikazujemo legendu uz eksplicitno navodjenje lokacije
plt.legend(loc='upper left')
# prikazujemo mrezu
plt.grid()
# prikazujemo i ose
plt.axvline(0)
plt.axhline(0)

# na trecem polju crtamo logaritamsku funkciju

# proglsavamo drugo polje kao aktivno
plt.subplot(2,2,4)
# sva formatiranja koja navedemo u nastavku odnosice se samo na prvo polje
# na crtezu

# crtamo grafik
plt.plot(XX, CT, color="magenta", linewidth=2.5, linestyle="-",
label="logaritam")
# postavljamo granice za prikazivanje grafika po x osi
plt.xlim(0, 10.0)
# postavljamo granice za prikazivanje grafika po x osi
plt.ylim(-5.0, 5.0)
# obelezavamo tacaka na x osi
plt.xticks(np.linspace(0,10,11, endpoint=True))
# obelezavamo tacaka na y osi
plt.yticks(np.linspace(-5,5,11,endpoint=True))
# prikazujemo legendu uz eksplicitno navodjenje lokacije
plt.legend(loc='upper left')
# prikazujemo mrezu
plt.grid()
# prikazujemo i ose
plt.axvline(0)
plt.axhline(0)

# na kraju podesavamo zeljeni razmak izmedju polja na crtezu
plt.tight_layout()
# i prikazujemo crtez
plt.show()
```

Primer 21c. Subplot

4. Razni tipovi grafika.

```
import numpy as np
# da bismo crtali grafike u pajtonu, potrebna nam je
# biblioteka matplotlib
from matplotlib import pyplot as plt
# za crtanje 3d grafike moramo treba nam funkcija Axes3d iz
# paketa mpl_toolkits.mplot3d
from mpl_toolkits.mplot3d import Axes3D

# pored klasicnih grafika, biblioteka matplotlib podrzava i bar grafike

# pravimo dva niza slucajnih brojeva
n = 12;
X = np.arange(n);
Y1 = (1-X/float(n))*np.random.uniform(0.5,1.0,n)
Y2 = (1-X/float(n))*np.random.uniform(0.5,1.0,n)

# startujemo novu figuru
plt.figure(num = 1, figsize = (6,4))

# postavljamo novu osu
plt.axes([0.025, 0.025, 0.95, 0.95])
# crtamo oba niza sa razlicitim znacima
plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')

# postavljamo natpise iznad svakog pravougaonika
for x, y in zip(X, Y1):
    plt.text(x + 0.4, y + 0.05, '%.2f' % y, ha='center', va= 'bottom')

# postavljamo natpise ispod svakog pravougaonika
for x, y in zip(X, Y2):
    plt.text(x + 0.4, -y - 0.05, '%.2f' % y, ha='center', va= 'top')

# postavljamo granice i obelezja na osama
plt.xlim(-.5, n)
plt.xticks(())
plt.ylim(-1.25, 1.25)
plt.yticks(())

# prikazujemo crtez
plt.show()

# za prikazivanje uredjenih parova tacaka u ravni koristi se scatter plot
# scatter plot moze da se koristi i za prikazivanje krivih drugog reda

# kreiramo uredjene parove tacaka
n = 1024
X = np.random.normal(0, 1, n)
Y = np.random.normal(0, 1, n)
# boju tacke cemo da odredimo njenim uglom
T = np.arctan2(Y, X)
```

Primer 22a. Razni tipovi grafika.

```
# startujemo novu figuru
plt.figure(num = 2, figsize = (6,4))

# definisemo novu osu na crtezu
plt.axes([0.025, 0.025, 0.95, 0.95])

# prikazujemo uredjene parove tacaka
plt.scatter(X, Y, s=75, c=T, alpha=.5)

# postavljamo granice i obelezja na osama
plt.xlim(-1.5, 1.5)
plt.xticks(())
plt.ylim(-1.5, 1.5)
plt.yticks(())

# prikazujemo crtez
plt.show()

# matplotlib ume da crta i 3d grafike
# da bismo crtali 3d graike potreban nam je objekat Axes3d iz paketa
mpl_toolkits.mplot3d

# kreiramo novu figuru i pamtimo njenu referencu
fig = plt.figure(num = 3)
# definise ose na figuri
ax = Axes3D(fig)
# kreiramo podelu po x osi
X = np.arange(-4, 4, 0.25)
# kreiramo podelu po y osi
Y = np.arange(-4, 4, 0.25)
# na osnovu kreirane podele kreiramo mrezu koju cemo koristiti za
# crtanje
X, Y = np.meshgrid(X, Y)
# racunamo pomocnu funkciju
R = np.sqrt(X ** 2 + Y ** 2)
# racunamo vrednosti funkcije sinus u tackama podele
Z = np.sin(R)

# crtamo povrs
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)
# crtamo konture u XY ravni
ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=plt.cm.hot)
# postavljamo limit za prikazivanje grafika po z osi
ax.set_zlim(-2, 2)

# prikazujemo crtez
plt.show()
```

Primer 22b. Razni tipovi grafika.

SciPy biblioteka

1. Ugrađene konstante.

```
import sys
# ukljucujemo biblioteku numpy
import scipy.constants as sp

# ugradjene konstante
print("sciPy - pi = %.16f"%sp.constants.pi)
print("sciPy - zlatni presek = %.16f"%sp.constants.golden)
print("sciPy - brzina svetlosti = %.16f"%sp.constants.c)
print("sciPy - avogadrov broj = %g"%sp.constants.Avogadro)
print("sciPy - masa elektrona = %g"%sp.constants.m_e)
print("sciPy - masa protona = %g"%sp.constants.m_p)
print("sciPy - masa neutrona = %g"%sp.constants.m_n)

print("sciPy - mili = %.16f"%sp.constants.milli)
print("sciPy - mikro = %.16f"%sp.constants.micro)
print("sciPy - nano = %.16f"%sp.constants.nano)
print("sciPy - kilo = %.16f"%sp.constants.kilo)

print("sciPy - stepen u radijanima = %.16f"%sp.constants.degree)
print("sciPy - minut u sekundama = %.16f"%sp.constants.minute)
print("sciPy - dan usekundama = %.16f"%sp.constants.day)
print("sciPy - svetlosna godina u metrima =
%.16f"%sp.constants.light_year)
```

Primer 23. Ugrađene konstante.

2. Integraljenje.

```
import sys
import numpy as np
import scipy.integrate

# hocemo da nadjemo vrednost integrala funkcije e^(-x^2) na intervalu 0,1

# prvo definisemo lambda izraz koji opisuje nasu funkciju
f = lambda x: np.exp(-x**2)
# racunamo vrednost integrala pomocu funkcije quad iz scipy paketa
# argumenti funkcije su podintegralna funkcija i granice integraljenja
i = scipy.integrate.quad(f, 0, 1)
# prikazujemo dobijeni rezultat
print(i)
# prvi broj u rezultatu je vrednost integrala, a drugi broj je ocean
# apsolutne greske integrala

# na slican nacin mogu se racunati i dvostruki (dblqaud), trostruki
# (tplquad) i n-tostruki (nquad) integral

# u narednom primer izracunavamo dvostruki integral funkcije 16xy pri cemu
# su granice integraljenja po x 0 i sqrt(1 - 4y^2), a granice
# integraljenja po y su 0 i 1/2
```

Primer 24a. Integraljenje.

```
# da bismo to uradili treba nam funkcija dblquad iz paketa scipy.integrate
# argumenti funkcije dblquad su podintegralna funkcija, granice
# integraljenja za x i granice integraljenja za y
# granice integraljenja za y se uvek moraju zadati kao funkcije, cak i
# kada se radi o konstantama

# prvo definisemo lambda izraz koji opisuje podintegralnu funkciju
f = lambda x,y: 16*x*y
# a nakon toga definisemo lambde koje opisuju granice integraljenja po x

# donja granica
# iako se radi o konstanti, moramo da je definisemo kao funkciju pomocu
# lambda izraza
g = lambda x : 0
# gornja granica
h = lambda y: np.sqrt(1 - 4*y**2)
# izracunavamo integral
ii = scipy.integrate.dblquad(f, 0, 0.5, g, h)
# stampamo dobijeni rezultat
print(ii)
```

Primer 24b. Integraljenje.

3. Interpolacija

```
import sys
import numpy as np
import scipy.interpolate as sc
import matplotlib.pyplot as plt

# scipy biblioteka podrzava i skup funkcija za interpolaciju

# pravimo podelu intervala
x = np.linspace(0, 4, 12)
# racunamo vrednosti funkcije u tackama podele
y = np.cos(x**2/3+4)
# stampamo dobijene parove
print(x,y)

# prikazujemo na grafiku dobijene tacke
plt.figure(num = 1)
plt.plot(x, y, 'o')
plt.show()

# na osnovu skupa uredjenih parova (x,y) zelimo da kreiramo
# interpolacioni polinom
# da bismo to postigli treba nam funkcija interp1d koja kao
# argumente ima vrednosti x i y i metod koji zeimo da primenimo
# prilikom izracunavanja interpolacionog polinoma

# racunamo linearni interpolacioni polinom
f1 = sc.interp1d(x, y, kind = 'linear')

# racunamo kubni interpolacioni polinom
f2 = sc.interp1d(x, y, kind = 'cubic')
```

Primer 25a. Interpolacija.

```
# na grafiku prikazujemo dobijene interpolacione polinome

# prvo pravimo novu proizvoljnu podelu intervala
xnew = np.linspace(0, 4, 30)

# krairamo novu figuru
plt.figure(num = 2)
# na istom grafiku prokazujemo sva tri crteza
# to postizemo tako sto redom navodimo parove nezavisnih i zavisno
# promenljivih
# u pozivu funkcije plot
plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')
# prikazujemo legendu
plt.legend(['data', 'linear', 'cubic'], loc='best')
# na kraju, prikazujemo crtez
plt.show()

# pored interpolacionih polinoma, scipy biblioteka ima podrsku i za
# splajnove

# pravimo novu podelu intervala
x = np.linspace(-3, 3, 50)
# unosimo beli sum u vrednosti funkcije
y = np.exp(-x**2) + 0.1*np.random.rand(50)

# prikazujemo dobijenu parove tacaka
plt.figure(num=3)
plt.plot(x,y,'ro', ms = 5)
plt.show()

# na osnovu kreiranog skupa tacaka iracunavamo splajn aprkocimaciju
spl = sc.UnivariateSpline(x,y)
# pravimo novu podelu intervala
xs = np.linspace(-3,3, 1000)
# crtamo novi grafiku
plt.figure(num=4)
plt.plot(xs, spl(xs), 'g', lw = 3)
plt.show()

# zapamtimo staru aproksimaciju
old = spl(xs)

# rucno menjanmo faktor usrednjavanja
spl.set_smoothing_factor(0.5)
plt.figure(num = 5)
plt.plot(xs, spl(xs), 'b', lw=3)
plt.show()

# na kraju priakzujemo sve na istom grafiku
plt.figure(num = 6)
plt.plot(x,y,'ro', xs, old, 'g', xs, spl(xs), 'b')
plt.show()
```

Primer 25b. Interpolacija.

4. Rešavanje sistema jednačina, algebarski problemi.

```
import sys
import numpy as np
import scipy.linalg as sc

# pored integraljenja i interpolacije, scipy podrzava i skup
# funkcija za resavanje algebarskih problema

# pomocu funkcije solve mozemo da resavamo sisteme jednacina oblika
# Ax=b

# prvo definisemo matricu sistema
A = np.array([[3,2,0],[1,-1,0],[0,5,1]])
# definisemo slobodni vektor
b = np.array([2,4,-1])

# resavamo sistem Ax = b pomocu funkcije solve
x = sc.solve(A,b)

# prikazujemo dobijeno resenje
print(x)

# pomocu funkcije det, mozemo da pronadjemo determinantu matrice

# definisemo matricu
A = np.array([[1,2],[3,4]])
# izracunavamo determinantu
x = sc.det(A)

#stampamo dobijeni rezultat
print(x)

# da bismo nasli inverznu matricu treba da koristimo funkciju inv
A_1 = sc.inv(A)
# stampamo inverznu matricu
print(A_1)
# proveravamo dobijeni rezultat
print(np.dot(A, A_1))


# pored determinante i inverzne matrice, mozemo da izracunamo i sopstvene
# vektore i sopstvene vrednosti matrice
# da bismo to postigli, treba nam funkcija eig
l, v = sc.eig(A)

# stmapamo sopstvene vrednosti
print(l)

# stampamo sopstvene vektore
print(v)
```

Primer 26. Rešavanje sistema jednačina, algebarski problemi.

5. Permutacije, kombinacije, binomni koeficijent, gama funkcija.

```
import sys
import numpy as np
# neophodno za koriscenje comb i perm
import scipy.special as sc

# za izracunavanje binomnog koeficijenta koristi se funkcija binomnog
print(sc.binom(10,3))

# racunamo broj kombinacije sa ponavljanjem
res = sc.comb(10, 3, exact = False, repetition = True)
print(res)

# racunamo broj kombinacije bez ponavljanjem
res = sc.comb(10, 3, exact = False, repetition = False)
print(res)

# za racunanje k-permutacija koristi se funkcija perm
# tj. odjednom izvlacim 3 od 10 kuglica
res = sc.perm(10, 3, exact = True)
print(res)

# gamma funkcija
print(sc.gamma([0.5, 1, 1.5]))
```

Primer 27. Permutacije, kombinacije, binomni koeficijent, gama funkcija.

6. Konveksni omotač.

```
import sys
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt

# cest problem u matematici i racunarstvu je pronalazenje konveksnog
# omotaca. biblioteka scipy nam omogucava da lako pronadjemo konveksni
# omotac skupa tacaka

# kreiramo niz trideset slucajnih tacaka u ravni
points = np.random.rand(30,2)
# izracunavamo konvekstni omotac skupa tacaka
hull = ConvexHull(points)
# graficki priakzujemo dobijene rezultate
# kreiramo novu figuru
plt.figure(num = 1)
# crtamo skup tacaka
plt.plot(points[:,0], points[:,1], 'o')
# crtamo omotac liniju po liniju
for simplex in hull.simplices:
    plt.plot(points[simplex,0], points[simplex,1], 'k-')
# prikazujemo dobijeni grafik
plt.show()
```

Primer 28. Permutacije, kombinacije, binomni koeficijent, gama funkcija.

SimPy biblioteka

1. Simboličko izračunavanje. Osnove.

```
import sys
from sympy import *
init_printing(use_unicode=True)

# sve sto smo do sada radili bilo je numericko izracunavanje
# ponekad to nije ono sto nam treba, jer numericko izracunavanje sa sobom
# donosi nezgodnu stvar, tj. implicitno se svi brojevi numericki
# aproksimiraju sa nekom tacnoscu
# na primer, sqrt(2) je iracionalan broj i kao takav ima beskonacno mnogo
# decimal. s obzirom da je racunar ipak konacna masina, iracionalni
# brojevi su uvek aproksimirani sirinom tipa podataka koji se koristi
# tj. i pre pocetka iracunavanja mi smo uneli  gresku u nas sistem

# simbolicko izracunavanje nam u tom slucaju moze pomoci. Svi brojevi i
# konstante se tumace kao
# apsolutno tacne vrednost i ne aproksimiraju se. simbolicko izracunavanje
# nije zasnovano na
# numerickim pravilima i kao takvo ne poznaje koncept numericke greske

# definisemo racionalan broj, simbolicki
a = Rational(1,2)
# definisemo a kao racinalan broj klasicno
aa = 1.0/2
# stampamo dobijeni rezultat
print(a)
# stampamo dobijeni rezultat
print(aa)

# na ovom primeru se ne vidi razlika izmedju simbolickog i numerickog
# izracunvanja, ali primer sa nekim racionalni ili iracionalnim brojem to
# bolje ilustruje
b = Rational(1,3)
print(b)
# definisemo b kao realni broj
bb = 1.0/3.0
print(bb)
# ovde je b tacna vrednost, tj. 1/3
# a bb je aproksimirana vrednost sa konacno mnogo cifara

# pi je konstanta sa tacnom vrednoscu
print(pi)
# koju moemo aproksimirati sa prozivoljno mnogo cifara
print(pi.evalf(1000))

# mozemo i da racunamo apsolutno tacno
print(pi**2)
print(pi + exp(1))
# ali mozemo i da aproksimiramo izraze
print((pi+exp(1)).evalf())
```

Primer 29a. Simboličko izračunavanje. Osnove.

```
# pored konacnih vrednosti sympy podrzava i koncept beskonacnosti. Klasa
# beskonacnosti se zove oo
print(oo > 9999)
# racunanje sa beskonacnostima je takodje podrzano
print(oo + 1)

# tacna vrednost korena
print(sqrt(2))
# aprkosimirana vrednost korena
x = sqrt(2)
print(x.evalf(100))

# racunamo zbir racionalnih brojeva
print(Rational(1,2) + Rational(1,3))
# racunamo bir racinalnog i iracionalnog broja
print(pi + Rational(1,2))

# poread ugradjenih simbola, mozemo da definisemo i svoje simbole i
# kasnije ih koristimo u izraima

# definisemo dva nova simbola
x = Symbol('x')
y = Symbol('y')
z = Symbol('z')

# simbolicki izracunvamo izraze
print(x + y + x - y)
# izracunavamo kvadrat binoma
print((x+6)**2)

# ovakav prikaz bas i nije reprezentativan. najcesce ovo prikzuje u
# razvijenom obliku da bismo razvili algebrasku jednacini koristi se
# funkcija expandtabs
print(expand((x+y)**2))
print(expand((x+y)**3))

# prilikom razvijanje, mozemo da naglasimo da treba da se koristi
# kompleksni domen
print(expand(x+y, complex=True))
# mozemo da razvijamo i trigonometrijske formule
print(expand(cos(x+y), trig=True))

# pored razvijanja, mozemo i da uproscavamo jednacine
print(simplify((x + x*y)/x))
print(simplify(sin(x)/cos(x)))

# pored uproscavanja biblioteka sympy podrzava i faktorisanje polinoma
# u tu svrhu koristi se funkcija factor
print(factor(x**3 - x**2 + x - 1))
print(factor(x**2*z + 4*x*y*z + 4*y**2*z))

# ukoliko je neophodno pretrazivanje faktora ili pronalazenje konkretnog
# faktora, bolje je koristiti funkciju factor_list koja vraca faktore
# spakovane u listu
print(factor_list(x**2*z + 4*x*y*z + 4*y**2*z))
```

Primer 29b. Simboličko izračunavanje. Osnove.

```
# funkcija factor u opstem slucaju ne radi samo nad polinomima, vec se
# moze primenjivati i na druge klase funkcija
print(factor(cos(x)**2 + 2*cos(x)*sin(x) + sin(x)**2))

# pored funkcije factor, cesto se koristi i funkcija collect, koja grupise
# cinioce po stepenima

# definisemo izraz
expr = x*y + x - 3 + 2*x**2 - z*x**2 + x**3
# i na primer, zelimo da grupisemo izraz oko stepena promenljive x
# to postizemo funkcijom collect
collected_expr = collect(expr, x)
print(collected_expr)

# iz ovako grupisanog izraza, lako mozemo da saznamo koji se koeficijent
# nalazi uz koji stepen promenljive da bismo to postigli koristimo metod
# coeff

# na primer, da bismo dobili koeficijent uz x^2, treba da napisemo sledecu
# naredbu
print(collected_expr.coeff(x, 2))

# funkcija cancel se koristi za ponistavanje clanova u racionalnim
# funkcijama
print(cancel((x**2 + 2*x + 1)/(x**2 + x)))

# definisemo izraz
expr = 1/x + (3*x/2 - 2)/(x - 4)
# svodimo izraz na neskrativ obliku
print(cancel(expr))

# definisemo novi izraz
expr = (x*y**2 - 2*x*y*z + x*z**2 + y**2 - 2*y*z + z**2)/(x**2 - 1)
# svodimo izraz na neskrativ obliku
print(cancel(expr))
```

Primer 29c. Simboličko izračunavanje. Osnove.

2. Limesi, izvodi, integrali.

```
import sys
from sympy import *
init_printing(use_unicode=True)

# uz pomoc sympy biblioteke mozemo da racunamo i analiticke probleme

# prvo definisemo simbole koje cemo koristiti u primerima
x = Symbol('x')
y = Symbol('y')
z = Symbol('z')

# sympy podrzava rad sa limesima pomocu funkcije limit
# potrebno je samo da definisemo funkciju i kazemo cemu tezi promenljiva
```

Primer 30a. Limesi, izvodi, integrali.

```
print(limit(sin(x)/x, x, 0))
print(limit(x, x, oo))
print(limit(1/x, x, oo))
print(limit(x**x, x, 0))

# pored izracunavanja limesa, sympy ume da racuna i izvode funkcija pomocu
# funkcije diff
print(diff(sin(x), x))
print(diff(2*sin(x/3), x))
print(diff(tan(x), x))

# mozemo da probamo i po definicija da izracunamo izvod
print(limit((tan(x+y) - tan(x))/y, y, 0))

# pored jednostrukih izvoda, moguce je raditi i visetruke izvode tako sto
# se navede treci argument

# prvi izvod
print(diff(2*sin(x/3), x, 1))
# drugi izvod
print(diff(2*sin(x/3), x, 2))
# treci izvod
print(diff(2*sin(x/3), x, 3))

# sympy ume da racuna parcijalne izvode funkcija vise promenljivih
# pri cemu se izvodi rade onim redom kojim su navedeni u listi argumenata
expr = exp(x*y*z)
# stampamo parcijalni izovd po x, y^2 i z^4
print(diff(expr, x, y, y, z, z, z, z))
# sto je ekvivalentno sa
print(diff(expr, x, y, 2, z, 4))

# pored izvoda, integrali su takodje direktno podrzani u sympy bilbioteci.
# za razliku od ranije pomenutih metoda integracije u scipy, integracija u
# sympy je simbolicka sto znaci da je rezultat integraljenja analiticki
# oblik resenja
print(integrate(cos(x), x))
print(integrate(log(x), x))
print(integrate(1/(1+x**2), x))
# ali to ne znaci da ne mozemo da resavamo i odredjene integrale
print(integrate(exp(-x), (x, 0, oo)))
# iako broj, resenje je simbolicko, tj. apsolutno tacno

# pored jednostrukih integrala, moguce je izracunavati i visestruke
# integrale
print(integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo)))

# u slucaju da integrate ne moze da nadje simbolicko resenje integrala,
# povratna vrednost ce biti Integral. u tom slucaju ili treba rucno uvesti
# neku smenu ili resiti integral numericki
```

Primer 30b. Limesi, izvodi, integrali.

```
# razvijanje funkcija u tejlorov red je takodje dostupno u sympy paketu

# definisemo funkciju
expr = exp(sin(x))

# razivjamo funkciju oko 0 u red 4. stepena
print(expr.series(x, 0, 4))
```

Primer 30c. Limesi, izvodi, integrali.

3. Rešavanje jednačina.

```
import sys
from sympy import *
init_printing(use_unicode=True)

# prvo definisemo simbole
x = Symbol('x')
y = Symbol('y')

# za resavanje jednacina u paketu sympy koristi se funkcija solve.
# prvi argument funkcije je jedancina koju treba resiti, a drugi je
# promenljiva po kojoj treba resiti jednacinu

# na primer, hocemo da nadjemo resenja x**4 - 1 == 0
print(solve(x**4 - 1, x))

# u slucaju da resavamo sistem jednacina, potrebno je da argumente
# prosledimo funkciji kao liste.
print(solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y]))

# pored algebraskih jednacina, solve ume da resi i neke nelinearne
# jednacine poput
print(solve(exp(x) + 1, x))

# u opstem slucaju, solve se koristi iskljucivo za resavanje algebraskih
# jednacina

# pored algebraskih jednacina , sympy ume da resava i bulovske jednacine.
# U tu svrhu se koristi funkcija satisfiable
print(satisfiable(x & y))
print(satisfiable(x & ~x))
```

Primer 31. Rešavanje jednačina.

4. Matrice.

```
import sys
from sympy import *
init_printing(use_unicode=True)

# u sympy biblioteci matrice se definisu funkcijom Matrix
# i mogu da sadrze brojeve i simbole

# kreiramo matricu
M = Matrix([[1, -1], [3, 4], [0, 2]])
# prikazujemo kreiranu matricu
print(M)

# kreiramo novu matricu
M = Matrix([[1, 2, 3], [3, 2, 1]])
# kreiramo slobodan vektor
N = Matrix([0, 1, 1])
# mnozimo matricu i vektor i stampamo rezultat
print(M*N)
# dimenzije matrice dobijamo pomocu polja shape
print(M.shape)

# pristup vrstama i kolonama se postize pomocu polja row i col

# stampamo prvu vrstu
print(M.row(0))
# stampamop poslednju kolonu
print(M.col(-1))
# indeksiranje je isto kao kod stringova

# uklanjanje kolona se vrsi pomocu funkcije col_del, a uklanjanje vrsta
# pomocu. funkcije row_del. Uklanjanje modifikuje postojecu matricu

# uklanjamo prvu kolonu
M.col_del(0)
# stampamo smanjenu matricu
print(M)

# uklanjamo drugu vrstu
M.row_del(1)
# stampamo smanjenu matricu
print(M)

# dodavanje kolona se radi pomocu funkcije col_insert, a vrsta pomocu
# funkcije row_insert
# dodavanje vrsta i/ili kolona kreira novu matricu, a stara ostaje
# nepromenjena

# dodajemo vrstu na kraj matrice
# kreiramo novu matricu i pamtimo je pod starim imenom
M = M.row_insert(1, Matrix([[0, 4]]))
# stampamo novu matricu
print(M)
```

Primer 32a. Matrice.

```
# dodajemo kolonu na pocetak matrice
# kreiramo novu matricu i pamtimo je pod starim imenom
M = M.col_insert(0, Matrix([1, -2]))
# stampamo dobijenu matricu

# sympy podrzava i sve klasicne operacije sa matricama

# kreiramo dve nove matrice
M = Matrix([[1, 3], [-2, 3]])
N = Matrix([[0, 3], [0, 7]])

print(M+N)
print(M-N)
print(M*N)
print(3*M)
print(M**2)
# inverzna matrica
print(M**-1)
# matrica N ima determinantu jednaku 0, pa nema inverznu matricu
# nakon ove naredbe trebalo bi da dobijemo ValueError
try:
    print(N**-1)
except (ValueError):
    print('Matrica nije invertibilna')

# transponovana matrica
print(M.T)
print(N.T)

# determinanta matrice
print(M.det())
print(N.det())

# sopstvene vrednosti matrice se racunaju pomocu funkcije eigenvals
# a sopstveni vektori se racunaju pomocu funkcije eigenvects

# kreiramo matricu M
M = Matrix([[3, -2,  4, -2], [5,  3, -3, -2], [5, -2,  2, -2], [5, -2, -3,
3]])
# izracunavamo sopstvene vrednosti
print(M.eigenvals())

# izracunavamo sopstvene vektore
print(M.eigenvects())

# da bismo dijagonalizovali matricu koristimo funkcije diagonalize
P, D = M.diagonalize();
print(P*D*P**-1)
print(P*D*P**-1 == M)
```

Primer 32b. Matrice.

Tkinter biblioteka

1. Prazan prozor.

```
# da bismo mogli da pravimo gui u Pythonu
# potrebno je da ukljucimo biblioteku Tkinter
from Tkinter import *

# prazan prozor pravimo pomocu funkcije
# Tk() iz biblioteke Tkinter
window = Tk()

# naslov prozora postavljamo pomocu
# metoda title kojem kao argument
# prosledimo zeljeni naslov
window.title('Prazan prozor')

# pored prozora mozemo da dozvolimo korisniku
# da menja velicinu kreiranog prozora
# to postizemo pomocu funkcije resizable ciji
# se argumenti redom omogucavaju menjanje velicine
# prozora po sirini i po visini.

# u ovom slucaju, zabranjeno je menjanje velicine
# prozora po sirini, ali je dozvoljeno menjanje
# po visini
window.resizable(0,1)

# da bismo korisniku prikazali kreirani
# prozor moramo da pozovemo metod mainloop
window.mainloop()
```

Primer 33. Prazan prozor.

2. Labele i pozicioniranje na formi.

```
# da bismo mogli da pravimo gui u Pythonu
# potrebno je da ukljucimo biblioteku Tkinter
from Tkinter import *

# prazan prozor pravimo pomocu funkcije
# Tk() iz biblioteke Tkinter
window = Tk()
# postavljamo naslov na prozor
window.title('Labels')
# dozvoljavamo promenu velicine prozora
# po obe ose
window.resizable(1,1)
```

Primer 34a. Labele i pozicioniranje na formi.

```
# definisemo klasu koja ce nam omoguciti
# da rasporedimo labele na jednostavan nacin
class Application(Frame):

    # kreiramo konstruktor
    # prvi argument je referenca na samog sebe
    # drugi argument je referenca na prozor nad kojim
    # zelimo da napravimo frejm
    def __init__(self, master):
        # kreiramo Frame
        Frame.__init__(self, master)
        # crtamo objekte na formi
        self.createLabels()

    def createLabels(self):
        # kreiramo prvu labelu i postavljamo tekst
        self.labelOne = Label(self, text="Prva labela")
        # postavljamo labelu na formu
        # labela se nalazi u prvoj vrsti, prvoj koloni
        # i prostire se na 2 kolone
        self.labelOne.grid(row = 0, column = 0, columnspan = 3)

        # kreiramo drugu labelu i postavljamo tekst i pozadinu
        self.labelTwo = Label(self, text="Labela 2", background = 'red')
        # postavljamo labelu na formu
        self.labelTwo.grid(row = 1, column = 0, columnspan = 1)

        # kreiramo trecu labelu i postavljamo tekst i boju teksta
        self.labelThree = Label(self, text="Labela 3",foreground = 'blue',
                                background = 'white')
        # postavljamo labelu na formu
        self.labelThree.grid(row = 1, column = 1, columnspan = 2)

        # kreiramo cetvrtu labelu i postavljamo tekst i boju teksta
        # i visinu labele
        self.labelFour = Label(self, text="Labela 4",foreground = 'red',
                               background = 'black',height = 5)
        # postavljamo labelu na formu
        self.labelFour.grid(row = 2, column = 0, columnspan = 2)

        # kreiramo petu labelu i postavljamo tekst i boju teksta
        # i visinu labele
        self.labelFive = Label(self, text="Labela 5",foreground = 'white',
                               background = 'black', height = 5,
                               font = ('Helvetica', 18),
                               borderwidth = 4,justify = RIGHT )
        # postavljamo labelu na formu
        self.labelFive.grid(row = 2, column = 2, columnspan = 1)
```

Primer 34b. Labele i pozicioniranje na formi.

```
# crtamo komponente na prozoru
# i postavljamo menadzer rasporedjivanja
# (layout manager) na formi
# izabrani menadzer je grid, tj. mreza
# izabrani menadzer je grid, tj. mreza
app = Application(window).grid()
# da bismo korisniku prikazali kreirani
# prozor moramo da pozovemo metod mainloop
window.mainloop()
```

Primer 34c. Labele i pozicioniranje na formi.

3. Dugmići i polja za unos.

```
# da bismo mogli da pravimo gui u Pythonu
# potrebno je da ukljucimo biblioteku Tkinter
from Tkinter import *

# prazan prozor pravimo pomocu funkcije
# Tk() iz biblioteke Tkinter
window = Tk()
# postavljamo naslov na prozor
window.title('Buttons and entries')
# dozvoljavamo promenu velicine prozora
# po obe ose
window.resizable(1,1)

# definisemo klasu koja ce nam omoguciti
# da rasporedimo labele na jednostavan nacin
class Application(Frame):
    # kreiramo konstruktor
    # prvi argument je referenca na samog sebe
    # drugi argument je referenca na prozor nad kojim
    # zelimo da napravimo frejm
    def __init__(self, master):
        # kreiramo Frame
        Frame.__init__(self, master)
        # crtamo objekte na formi
        self.createWidgets()

    #definisemo funkciju koja brise sadrzaj sa textBoxa
    def clearText(self):
        # brisemo tekst sa text boxa
        self.textBox.delete(0, END)
        # upisujemo poruku korisniku da je akcija izvrsena
        self.textBox.insert(0, 'Cleared')
        # stampamo log na konzoli
        print('Text box cleared')

    # definisemo funkciju koja cita sadrzaj sa textBoxa
    def readText(self):
        # citamo sadrzaj i pamtimo ga u promenljivoj s
        s = self.textBox.get()
        # stamapamo log na konzoli
        print('Text box: %s'%(s))
```

Primer 35a. Dugmići i polja za unos.

```
    def createWidgets(self):
        # kreiramo text box pomocu koga cemo da unosimo tekst
        # komponenta za unos u tkinter-u se zove Entry
        self.textBox = Entry(self,
                             font=("Helvetica", 16),
                             borderwidth=0,
                             relief=RAISED,
                             justify=CENTER
                            )
        # upisujemo pocetni tekst na komponentu
        self.textBox.insert(0, "Izmenite tekst")
        # postavljamo komponentu na formu
        self.textBox.grid(row = 0, column = 0, columnspan = 2)

        # kreiramo dugme za brisanje sadrzaja sa textboxa
        # jedan nacin da se podesi akcija prilikom
        # klika na dugme je polje command
        # da bi se akcija podesila dovoljno je da
        # dodelimo lambda izraz koji opisuje zeljenu akciju
        self.clearButton = Button(self,
                                  font=("Helvetica",11),
                                  text="Clear",
                                  borderwidth=0,
                                  command = lambda: self.clearText()
                                 )
        # postavljamo dugme na formu
        self.clearButton.grid(row = 1, column = 0, columnspan = 1)

        # kreiramo dugme za citanje sadrzaja sa textboxa
        self.readButton = Button(self,
                                  font=("Helvetica",11),
                                  text="Read",
                                  borderwidth=0,
                                  command = lambda: self.readText()
                                 )
        # postavljamo dugme na formu
        self.readButton.grid(row = 1, column = 1, columnspan = 1)

# crtamo komponente na prozoru
# i postavljamo menadzer rasporedjivanja
# (layout manager) na formi
# izabrani menadzer je grid, tj. mreza
app = Application(window).grid()
# da bismo korisniku prikazali kreirani
# prozor moramo da pozovemo metod mainloop
window.mainloop()
```

Primer 35b. Dugmići i polja za unos.

4. Obrada događaja I vezivanje handler-a.

```
# da bismo mogli da pravimo gui u Pythonu
# potrebno je da ukljucimo biblioteku Tkinter
from Tkinter import *

# prazan prozor pravimo pomocu funkcije
# Tk() iz biblioteke Tkinter
window = Tk()
# postavljamo naslov na prozor
window.title('Buttons and entries')
# dozvoljavamo promenu velicine prozora
# po obe ose
window.resizable(1,1)

# definisemo klasu koja ce nam omoguciti
# da rasporedimo labele na jednostavan nacin
class Application(Frame):

    # kreiramo konstruktor
    # prvi argument je referenca na samog sebe
    # drugi argument je referenca na prozor nad kojim
    # zelimo da napravimo frejm
    def __init__(self, master):
        # kreiramo Frame
        Frame.__init__(self, master)
        # crtamo objekte na formi
        self.createWidgets()

    # definisemo event handler koji stampa tekst sa
    # dugmeta na koje je korisnik kliknuo
    def buttonName(self, event):
        # dohvatamo referencu na dugme na koje
        # je korisnik kliknuo
        b = event.widget
        # pomocu funkcije cget citamo text sa dugmeta
        # ako nam treba bilo koji drugi parametar, treba samo
        # navesti njegovo ime
        t = b.cget('text')

        # stampamo procitani tekst u konzoli
        print('Kliknuto je dugme: %s'%(t))

        # pomocu funkcije config menjamo boju dugmeta
        # na isti nacin menja se i bilo koji drugi parametar
        # navodjenjem imena i zeljene vrednosti
        b.config(background='red')

    # definisemo funkciju koja popunjava
    # formu komponentama
    def createWidgets(self):
        # lista dugmica
        self.buttons = []
```

Primer 36a. Obrada događaja I vezivanje handler.

```
        # u petlji kreiramo mrezu dugmica
        for i in range(4):
            for j in range(4):
                # kreiramo dugme i pamtimo ga
                # u listi dugmica
                self.buttons.append(Button(self, text=str(i*4+j)))
                # postavljamo dugme na formu
                self.buttons[i*4+j].grid(row = i, column = j)
                # vezujemo dugme za event handler
                # kada korisnik klikne levim tasterom misa
                # prikazace se u konzoli tekst sa dugmeta
                # levi klik misa je Button-1, srednji klik Button-2
                # a desni klik Button-3
                self.buttons[i*4+j].bind("<Button-1>", self.buttonName)

                # prednost vezivanja event handlera u odnosu na
                # postavljanje akcija pomocu command polja se ogleda u
                # tome sto u svakom trenutku znamo koje dugme je izazvalo
                # dogadjaj i na taj nacin mozemo da imamo samo jedan event
                # handler za citavu klasu srodnih dogadjaja.
                # u slucaju da koristimo command polje, morali bismo da
                # imamo po jednu funkciju za svako pojedinacno dugme, sto
                # nezanemarljivo usloznjava odrzavanje koda

# crtamo komponente na prozoru
# i postavljamo menadzer rasporedjivanja
# (layout manager) na formi
# izabrani menadzer je grid, tj. mreza
app = Application(window).grid()
# da bismo korisniku prikazali kreirani
# prozor moramo da pozovemo metod mainloop
window.mainloop()
```

Primer 36b. Obrada događaja I vezivanje handler.

5. Jednostavni kalkulator

```
# da bismo mogli da pravimo gui u Pythonu
# potrebno je da ukljucimo biblioteku Tkinter
from Tkinter import *

# prazan prozor pravimo pomocu funkcije
# Tk() iz biblioteke Tkinter
calculator = Tk()
# postavljamo naslov na prozor
calculator.title('Calculator')
# dozvoljavamo promenu velicine prozora
# po obe ose
calculator.resizable(1, 1);
```

Primer 37a. Jednostavni kalkulator

```
# definisemo klasu koja ce nam omoguciti
# da rasporedimo labele na jednostavan nacin
class Application(Frame):

    # kreiramo konstruktor
    # prvi argument je referenca na samog sebe
    # drugi argument je referenca na prozor nad kojim
    # zelimo da napravimo frejm
    def __init__(self, master):
        # kreiramo Frame
        Frame.__init__(self, master)
        # crtamo objekte na formi
        self.createWidgets()

    # funkcija menja tekst prikazan na ekranu kalkulatora
    def replaceText(self, text):
        # brisemo stari sadrzaj
        self.display.delete(0,END)
        # upisujemo novi od pocetka ekrana
        self.display.insert(0, text)

    # funkcija nadovezuje tekst na postojeci tekst
    # na ekranu kalkulatora
    def appendToDisplay(self, text):
        # citamo postjeci tekst sa ekrana
        self.entryText = self.display.get()
        # odredjujemo duzinu teksta
        self.textLength = len(self.entryText)

        # ako je na ekranu samo 0
        if self.entryText == "0":
            # upisujemo novi tekst
            self.replaceText(text)
        else:
            # u suprotnom, dodajemo novi tekst na kraj
            self.display.insert(self.textLength, text)

    # funkcija brise sadrzaj ekrana kalkulatora
    def clearText(self):
        self.replaceText("0")

    # funkcija brise jednu cifru sa ekrana
    def clearDigit(self):
        # citamo tekst sa ekrana
        self.entryText = self.display.get()
        # odredjujemo duzinu teksta
        self.textLength = len(self.entryText)

        # ako nije u pitanju nula i ako je tekst duzi od 1
        if self.entryText != "0" and self.textLength > 1:
            # izbacujemo poslednji upisani broj
            self.replaceText(self.entryText[:-1])
        else:
            # u suprotnom upisujemo 0 na ekran
            self.replaceText("0")
```

Primer 37b. Jednostavni kalkulator

```
    # funkcija izracunava izraz zapisan na ekranu
    def calculateExpression(self):
        # citamo tekst sa ekrana
        self.expression = self.display.get()
        # procenat menjamo deljenjem sa 100
        self.expression = self.expression.replace("%", "/ 100")

        try:
            # pokusavamo da izracunamo izraz
            self.result = eval(self.expression);
            # ako uspemo, prikazujemo rezultat
            self.replaceText(self.result)
        except:
            # ako ne, hvatamo izuzetak i prikazujemo poruku o gresci
            messagebox.showinfo("Error", "Invalid input",
                                icon="warning", parent=calculator)

    # definisemo funkciju koja popunjava
    # formu komponentama
    def createWidgets(self):

        # displej
        self.display = Entry(self, font=("Helvetica", 16), borderwidth=0,
                             relief=RAISED,justify=RIGHT)
        self.display.insert(0,"0")
        self.display.grid(row=0, column=0,columnspan=5)

        # prvi red
        self.sevenButton = Button(self, font=("Helvetica",11), text="7",
                                  borderwidth=0,
                                  command = lambda: self.appendToDisplay("7"))
        self.sevenButton.grid(row = 1, column = 0)

        self.eightButton = Button(self, font=("Helvetica",11), text="8",
                                  borderwidth=0,
                                  command = lambda: self.appendToDisplay("8"))
        self.eightButton.grid(row = 1, column = 1)

        self.nineButton = Button(self, font=("Helvetica",11), text="9",
                                  borderwidth=0,
                                  command = lambda: self.appendToDisplay("9"))
        self.nineButton.grid(row = 1, column = 2)

        self.timesButton = Button(self, font=("Helvetica",11), text="*",
                                  borderwidth=0,
                                  command = lambda: self.appendToDisplay("*"))
        self.timesButton.grid(row = 1, column = 3)

        self.clearButton = Button(self, font=("Helvetica",11), text="C",
                                      borderwidth=0,
                                      command = lambda: self.clearDigit())
        self.clearButton.grid(row = 1, column = 4)
```

Primer 37c. Jednostavni kalkulator

```
        # drugi red
        self.fourButton = Button(self, font=("Helvetica",11), text="4",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("4"))
        self.fourButton.grid(row = 2, column = 0)

        self.fiveButton = Button(self, font=("Helvetica",11), text="5",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("5"))
        self.fiveButton.grid(row = 2, column = 1)

        self.sixButton = Button(self, font=("Helvetica",11), text="6",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("6"))
        self.sixButton.grid(row = 2, column = 2)

        self.divideButton = Button(self, font=("Helvetica",11), text="/",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("/"))
        self.divideButton.grid(row = 2, column = 3)

        self.clearAllButton = Button(self, font=("Helvetica",11),
                                 text="CE",  borderwidth=0,
                                 command = lambda: self.clearText())
        self.clearAllButton.grid(row = 2, column = 4)

        # treci red
        self.oneButton = Button(self, font=("Helvetica",11), text="1",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("1"))
        self.oneButton.grid(row = 3, column = 0)

        self.twoButton = Button(self, font=("Helvetica",11), text="2",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("2"))
        self.twoButton.grid(row = 3, column = 1)

        self.threeButton = Button(self, font=("Helvetica",11), text="3",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("3"))
        self.threeButton.grid(row = 3, column = 2)

        self.minusButton = Button(self, font=("Helvetica",11), text="-",
                                  borderwidth=0,
                                  command = lambda: self.appendToDisplay("-"))
        self.minusButton.grid(row = 3, column = 3)

        self.percentageButton = Button(self, font=("Helvetica",11),
                                  text="%", borderwidth=0,
                                  command = lambda: self.appendToDisplay("%"))
        self.percentageButton.grid(row = 3, column = 4)
```

Primer 37d. Jednostavni kalkulator

```
        # cetvrti red
        self.zeroButton = Button(self, font=("Helvetica",11), text="0",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("0"))
        self.zeroButton.grid(row = 4, column = 0, columnspan = 2)

        self.dotButton = Button(self, font=("Helvetica",11), text=".",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("."))
        self.dotButton.grid(row = 4, column = 2)

        self.plusButton = Button(self, font=("Helvetica",11), text="+",
                                 borderwidth=0,
                                 command = lambda: self.appendToDisplay("+"))
        self.plusButton.grid(row = 4, column = 3)

        self.equalButton = Button(self, font=("Helvetica",11), text="=",
                                 borderwidth=0,
                                 command = lambda: self.calculateExpression())
        self.equalButton.grid(row = 4, column = 4)

# crtamo komponente na prozoru
# i postavljamo menadzer rasporedjivanja
# (layout manager) na formi
# izabrani menadzer je grid, tj. mreza
app = Application(calculator).grid()
# da bismo korisniku prikazali kreirani
# prozor moramo da pozovemo metod mainloop
calculator.mainloop()
```

Primer 37e. Jednostavni kalkulator

PyGame biblioteka

1. Prazan program.

```
# ovim primerom se generiše prazan prozor (ekran)

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

# inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()
# definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
# definišemo dimenzije ekrana
sirina = 400
visina = 400

# ispis u zaglavlju prozora
pygame.display.set_caption("Prazan ekran")
# poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))
ekran.fill(BELA)
pygame.display.flip()

kraj = False
while not kraj: # glavna petlja
    for dogadjaj in pygame.event.get(): # petlja obrade događaja
        if dogadjaj.type == pygame.QUIT:
            kraj = True
        # poziva se funkcija koja ispisuje trenutni dogadjaj
        # (nije relevantno za ovaj primer, ali je ilustrativno
        # da se vidi kako sta su dogadjaji i kako se interno zapisuju
        # (pomeranje miša, desni klik miša, skrolovanje miša...))
        print(dogadjaj)

# isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 38. Prazan program.

2. Zdravo svete.

```
# ovim programom se generiše ekran / prozor na kome
# je ispisano "Zdravo, svete!". Cilj  je
# usvojiti znanje kako se može pisati tekst na ekranu.

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()
```

Primer 39a. Zdravo svete.

```
#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 500
#uključuje se modul sa slovima
pygame.font.init()

# ovim biramo vrstu i veličinu fonta
myfont = pygame.font.SysFont('Comic Sans MS', 30)
# pomoću funkcije myfont.render( a1, a2, a3) se navodi šta
# se ispisuje (prvi argument), da li je font
# podebljan (boldovan) ili nije (izbor True ili False) (drugi argument)
# i bira se boja slova koja se ispisuju (treći argument)
tekst = myfont.render('Zdravo, svete!', True, (255, 100, 0))
#ispis u zaglavlju prozora
pygame.display.set_caption("Zdravo, svete!")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

#poziv funkcije koja ekran boji u (ovom slučaju) u belu boju
ekran.fill(BELA)


# početak glavne petlje programa
kraj = False
while not kraj: #glavna petlja
    for dogadjaj in pygame.event.get(): #petlja obrade događaja
        if dogadjaj.type == pygame.QUIT:
            kraj = True
    #poziv funkcije za ažuriranje ekrana (bojenje u belo)
    pygame.display.flip()
    #poziv funkcije za ispis teksta na ekranu
    ekran.blit(tekst,(100,100))

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 39b. Zdravo svete.

3. Statični prikaz slike – linijska struktura.

```
#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 450
visina = 200
```

Primer 40a. Statični prikaz slike.

```
#ispis u zaglavlju prozora
pygame.display.set_caption("Slika")

#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

#poziv funkcije koja ekran boji u (ovom slučaju) u belu boju
ekran.fill(BELA)

#kreiranje objekta slika
slika = pygame.image.load('logo.png').convert()

#prikaz slike na ekranu
ekran.blit(slika, ( 10,0))

#poziv funkcije za prikaz promene na ekranu
pygame.display.flip()

kraj = False
while not kraj: #glavna petlja
    for dogadjaj in pygame.event.get(): #petlja obrade događaja
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 40b. Statični prikaz slike.

4. Mapiranje pozadine.

```
# ovim primerom se demonstrira kako se može mapirati
# pozadina neke igrice, tačnije, kako možemo pozadinu prekriti
# različitim slikama što nije retka pojava u igricama.
# u ovom primeru na ekran postavljaju dve sličice koje se ponavljaju;
# pozadina je prekrivena sličicama sa po tri u vrsti i tri u koloni.
import pygame

# početak
pygame.init()

# podešavanje visine i širine ekrana
visina = 600
sirina = 400
ekran = pygame.display.set_mode((sirina, visina))
# lista u kojoj se zadaje pozicija sličice1, sličice2 ili praznog polja
slike = ["1 2", "22 ", " 11"];

# ucitavanje slika
slika1 = pygame.image.load('starwars1.png').convert()
slika2 = pygame.image.load('starwars2.png').convert()
# slika1.set_colorkey(BELA)
```

Primer 41a. Mapiranje pozadine.

```
# dimenzije učitanih slika
slika_sirina = slika1.get_width()
slika_visina = slika1.get_height()

# prolazimo kroz listu stringova (prva petlja), zatim kroz svaki
# string (druga petlja) i ukoliko je trenutni karakter 1 prikazaće
# se prva slika, ukoliko je karakter 2, prikazaće se druga slika.
y = 0
for vrsta in slike:
    x = 0
    for c in vrsta:
        if c == "1":
            ekran.blit(slika1, (x, y))
        elif c == "2":
            ekran.blit(slika2, (x, y))
        x += slika_sirina
    y += slika_visina
# poziv funkcije kojom se osvežava ekran i iscrtava sve promene
# iz prethodnih petlji
pygame.display.update()

# glavna petlja programa
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
# kraj programa
pygame.quit()
```

Primer 41b. Mapiranje pozadine.

5. Crtanje – kućica, linijska struktura.

```
# ovim primerom se demonstrira kako se može primenom funkcija za
# generisanje različitih oblika može iscrtati kućica na ekranu.

# uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

# inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

# definišemo boje koje koristimo u programu

BELA = (255, 255, 255)
CRVENA = (255, 0, 0)
ZELENA = (0, 255, 0)
PLAVA = (20, 100, 255)
ZUTA = (240, 240, 40)
CRNA = (0,0,0)
BRAON = (190,42,42)
```

Primer 42a. Crtanje – kućica, linijska struktura.

```
# definišemo dimenzije ekrana
sirina = 500
visina = 500

# ispis u zaglavlju prozora
pygame.display.set_caption("Kućica")

# poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

# poziv funkcija koje iscrtavaju pozadinu (nebo i travu)
pygame.draw.rect(ekran, PLAVA, (0, 0, 500, 350),  0)
pygame.draw.rect(ekran, ZELENA, (0, 350, 500, 150),  0)
# poziv funkcije koja iscrtava sunce
pygame.draw.circle(ekran, ZUTA, (90, 90), 40,  0)
# poziv funkcija koje crtaju kucicu
pygame.draw.rect(ekran, BELA, (200, 250, 200, 200),  0)
# poziv funkcije koja crta krov
pygame.draw.polygon(ekran, CRVENA, [(200, 250), (400, 250), (300, 100)],
0)
# poziv funkcija koje crtaju vrata i prozor
pygame.draw.rect(ekran, CRNA, (300, 350, 60, 100),  0)
pygame.draw.rect(ekran, BRAON, (230, 270, 50, 50),  0)

# poziv funkcije za prikaz promene na ekranu
pygame.display.flip()

kraj = False
while not kraj: #glavna petlja
    for dogadjaj in pygame.event.get(): #petlja obrade događaja
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 42b. Crtanje – kućica, linijska struktura.

6. Crtanje – apstrakcije, petlje (grananje).

```
# primer programa u kome se učenicima može demonstrirati kako se sa malo
# poznavanja korišćenja petlji može iscrtati zanimljiv apstrantan crtež

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

pygame.init()  #inicijalizacija biblioteke pygame (obično na početku
programa)

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 400
```

Primer 43a. Crtanje – apstrakcije, petlje (grananje).

```
#ispis u zaglavlju prozora
pygame.display.set_caption("Apstrakcija")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

# petlja u kojoj se generišu različite rotacije prave
for i in range(0,visina,10):
            pygame.draw.line(ekran, BELA, (0,i), (i, visina))
            pygame.draw.line(ekran, BELA, (0,i), (visina-i, 0))
            pygame.draw.line(ekran, BELA, (i,0), (visina, i))
            pygame.draw.line(ekran, BELA, (visina,i), (visina-i, visina))
            pygame.display.flip()

kraj = False
#glavna petlja
while not kraj:
    #petlja obrade događaja
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 43b. Crtanje – apstrakcije, petlje (grananje).

```
# primer programa u komse demonstrira kako se ekran može popuniti
# (išpartati) kosim paralelnim linijama

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 400

#ispis u zaglavlju prozora
pygame.display.set_caption("Kose linije")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

# petlja u kojoj se opisuju linije. linije su predstavljene dužima
# (svojom početnom i krajnom tačkom)
for i in range(0,2*visina,20):
            pygame.draw.line(ekran, BELA, (0,i), (i, 0))
            # iscrtavanje promena na ekranu
            pygame.display.flip()
```

Primer 44a. Crtanje – kose linije (grananje).

```
kraj = False
#glavna petlja
while not kraj:
    #petlja obrade događaja
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 44b. Crtanje – kose linije (grananje).

```
# program kojim se demonstrira iscrtavanjem kosih linija

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 400

#ispis u zaglavlju prozora
pygame.display.set_caption("Kose linije")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))
# konstanta 30 znači da interval od 0 do 2*visina deli sa korakom 30.
for i in range(0, 2*visina, 30):
            pygame.draw.line(ekran, BELA, (i,visina), (0, visina-i))
            pygame.display.flip()

kraj = False
#glavna petlja
while not kraj:
    #petlja obrade događaja
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 45. Crtanje – kose linije 2 (grananje).

```
#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 400

#ispis u zaglavlju prozora
pygame.display.set_caption("Linije")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

for i in range(0,2*visina,30):
            pygame.draw.line(ekran, BELA, (0,i), (i, 0))
            pygame.draw.line(ekran, BELA, (i,visina), (0, visina-i))
            pygame.display.flip()

kraj = False
#glavna petlja
while not kraj:
    #petlja obrade događaja
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 46. Crtanje – kvadrati horizontalni (grananje).

```
# ovim programom se demonstrira kako se pozivanjem horizontalnih i
# vertikalnih linija može išpartati ekran

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 400

#ispis u zaglavlju prozora
pygame.display.set_caption("Linije")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))
```

Primer 47a. Crtanje – kvadrati apstrakcija (grananje).

```
# u okviru ove petlje se iscrtavaju vertikalne i horizontalne linije
for i in range(0,visina,20):
            pygame.draw.line(ekran, BELA, (i,0), (i, visina))
            pygame.draw.line(ekran, BELA, (0,i), (sirina, i))
            pygame.display.flip()



kraj = False
#glavna petlja
while not kraj:
    #petlja obrade događaja
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True

#isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 47b. Crtanje – kvadrati apstrakcija (grananje).

```
# u okviru ovog programa se demonstrira kako se može nacrtati samo jedan
# deo apstraktnog cveta iz primera apstrakcija.py

#uključujemo biblioteke programskog jezika python potrebne za rad
import pygame, random, math

#inicijalizacija biblioteke pygame (obično na početku programa)
pygame.init()

#definišemo boje koje koristimo u programu
BELA  = (255, 255, 255)
#definišemo dimenzije ekrana
sirina = 400
visina = 400

#ispis u zaglavlju prozora
pygame.display.set_caption("Linije")
#poziv funkcije za generisanje ekrana / prozora
ekran = pygame.display.set_mode((sirina, visina))

#petlja u kojoj se postepeno rotira duž i time se postiže efekat obvojnice
for i in range(0,visina,10):
            pygame.draw.line(ekran, BELA, (0,i), (i, visina))
            pygame.display.flip()
kraj = False
# glavna petlja
while not kraj:
    # petlja obrade događaja
    for dogadjaj in pygame.event.get(): #petlja obrade događaja
        if dogadjaj.type == pygame.QUIT:
            kraj = True

# isključivanje biblioteke pygame (obično na kraju programa)
pygame.quit()
```

Primer 48. Crtanje – obvojnica (grananje).

7. Krugovi na dijagonali

```
# ovim primerom se demonstrira kako iscrtati n krugova (n mi zadajemo) na
# dijagonali ekrana kome su date širina i visina

# uključena i biblioteka math jer se koriste određene funkcije ove
# biblioteke
import pygame, math

# definišemo boje koje koristimo prilikom kreiranja ekrana
BELA = (255, 255, 255)
CRVENA = (255, 0, 0)

pygame.init()

# zadata visina i širina
visina = 400
sirina = 500
# podešavanje ekrana
ekran = pygame.display.set_mode((sirina, visina))
ekran.fill(BELA)

# n je broj krugova koji želimo da se pojave na dijagonali
n = 10
# funkcija sqrt() vraća realan broj, korišćenjem funkcije int() "sasecamo"
# realan broj
d = int(math.sqrt(sirina**2 + visina**2))
# zadajemo veličinu poluprečnika
r = d // (2 * n)
# zadajemo koordinate centra kruga
kx = (r*sirina) // d
ky = (r*visina) // d

# u okviru ove for petlje se generišu krugovi i nagon što se svi
# izgenerišu iscrtavaju se na ekranu nakon poziva funkcije
# pygame.display.update() koja je ispod for petlje
for i in range(n):
    pygame.draw.circle(ekran, CRVENA, ((2*i+1)*kx, visina - (2*i+1)*ky),
r, 3)

pygame.display.update()

# krugovi koji se vide nakon poziva ovog programa na ekranu su statički i
# zato se generišu i iscrtavaju van glavne petlje.

# glavna petlja programa
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
# kraj programa
pygame.quit()
```

Primer 49. Krugovi na dijagonali.

8. Animacije – loptica koja se odbija.

```
# primer demonstrira kako napraviti jednostavnu animaciju.
# cilj animacije je da se loptica kreće u okviru ekrana i
# da se kada "udari" o ivicu ekrana vrati unazad.
import pygame
# početak
pygame.init()

# zadavanje boja
BELA = (255, 255, 255)
CRNA = (0, 0, 0)

# zadavanje dimenzija ekrana i generisanje ekrana datih dimanzija
dim = (sirina, visina) = (400, 450)
ekran = pygame.display.set_mode(dim)

# sat je sistemski objekat koji usporava brzinu izvršavanja
# glavne petlje programa
clock = pygame.time.Clock()

# zadavanje koordinata centra kruga
(x, y) = (sirina // 2, visina // 2)
# vrednost promene koordinata
(dx, dy) = (5, 5)
r = 30

# početa glavne petlje programa
kraj = False
while not kraj:
    ekran.fill(BELA)
    pygame.draw.circle(ekran, CRNA, (x, y), r)
    pygame.display.flip()

    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True

    x += dx
    y += dy
    # proveravamo da li se loptica nalazi unutar samog ekrana
    # i kada dodje do ivice ekrana menjaju se tekuće koordinate centra
    if x - r < 0 or x + r > sirina:
        dx = -dx
    if y - r < 0 or y + r > visina:
        dy = -dy


     # podešavanje sata
    clock.tick(40)
# kraj programa
pygame.quit()
```

Primer 50. Animacije – loptica koja se odbija.

9. Animacije – vojnik koji se šeta.

```
# cilj primera je da prikaze animaciju u oviru koje patrolira strazar.
# naime, uvežena su dve slike stražara jedna slika je stražar okrenut na
# levo, druga slika je isti taj stražar okrenut na desno. animacija se
# sastoji u tome da se stražar kada dođe do ivice prozora okrene na drugu
# stranu (zamene se slike) i krene u drugom smeru
import pygame
# zadavanje boje
BELA = (255, 255, 255)
pygame.init()
# zadavanje dimenzije ekrana i generisanje ekrana
[sirina, visina] = [500, 400]
ekran = pygame.display.set_mode([sirina, visina])
#ispis u zaglavlju prozora
pygame.display.set_caption("Stražar patrolira")
# uvoženje sličica
strazar_levo = pygame.image.load('strazar_levo.png').convert()
strazar_desno = pygame.image.load('strazar_desno.png').convert()
# primenom funkcija get_width() i get_height() dobijamo dimenzije uveženih
# slika
strazar_sirina = strazar_levo.get_width()
strazar_visina = strazar_levo.get_height()

x = 0
y = (visina - strazar_visina) // 2
dx = 2

print(visina, strazar_visina, y)

sat = pygame.time.Clock()

kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
    # ukoliko slika stražara dođe do ivice, menjamo dx, čime menjamo x
    # koordinatu slike
    x += dx
    if x < 0 or x + strazar_sirina > sirina:
        dx = -dx
    # bojenje pozadine prozora u belo
    ekran.fill(BELA)
    # u zavisnosti od vrednosti promenljive dx menjamo sličicu stražara
    if dx > 0:
        ekran.blit(strazar_desno, (x, y))
    else:
        ekran.blit(strazar_levo, (x, y))
    pygame.display.update()
     # podešavanje brzine izvršavanja glavne petlje
    sat.tick(70)
# kraj programa
pygame.quit()
```

Primer 51. Animacije – vojnik koji se šeta.

10. Obrada događaja – šetanje sličice tasterom.

```
# u okviru ovog programa se prvi put susrećemo sa obradom događaja sa
# tastature. na ekran je učitana slika rakete koju je moguće "šetati" po
# ekranu gore, dole, levo, desno pomoću strelica na tastaturi.
import pygame
# zadavanje boja
CRNA = (0, 0, 0)
BELA = (255, 255, 255)
# početak programa
pygame.init()

pygame.key.set_repeat(10, 10)
# zadavanje dimenzije širine i visine ekrana
dim = (sirina, visina) = (800, 600)
ekran = pygame.display.set_mode(dim)

# ucitavanje slike brodic (svemirski brod, tj. raketa)
brodic = pygame.image.load('spaceship.png').convert()
# ovom funkcijom sve što je na slici bilo bele boje postavljamo na
providno
brodic.set_colorkey(BELA)
# promenljive brodic_sirina i brodic_visina postavljamo na
# dimenzije uvežene slike
brodic_sirina = brodic.get_width()
brodic_visina = brodic.get_height()

#inicijalizacija promenljivih x i y
x = sirina // 2
y = visina // 2

# funkcija za iscrtavanje (prikazivanje) slike na poziciji koja je
# određena koordinatama gornjeg levog temena slike
def nacrtaj():
    ekran.fill(CRNA)
    ekran.blit(brodic, (x - brodic_sirina // 2, y - brodic_visina // 2))
    pygame.display.update()

# iscrtavamo sliku
nacrtaj()

# početak glavne petlje u okviru koje reagujemo na događaje sa tastature,
# tačnije, u zavisnosti od toga koji taster sa tastature je pritisnut,
# slika će se pomerati gore, dole, levo, desno. pomeranje je regulisano
# promenom vrednosti promenljivih x i y
#(smanjenjem ili povećanjem za npr. 5)
```

Primer 52a. Obrada događaja – šetanje sličice tasterom.

```
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
        if dogadjaj.type == pygame.KEYDOWN:
            if dogadjaj.key == pygame.K_UP:
                y -= 5
            elif dogadjaj.key == pygame.K_DOWN:
                y += 5
            elif dogadjaj.key == pygame.K_LEFT:
                x -= 5
            elif dogadjaj.key == pygame.K_RIGHT:
                x += 5
            nacrtaj()

# kraj programa
pygame.quit()
```

Primer 52b. Obrada događaja – šetanje sličice tasterom.

11. Krugovi pomoću miša.

```
# cilj primera je da demonstrira kako se nakon kliktanja mišem po ekranu
# iscrtavaju krugovi različitih boja

# uključili smo biblioteku random jer koristimo funkciju randomint() kako
# bi generisali nasumičnu boju
import pygame, random

# definišemo funkciju koja vraća nasumičnu (random) boju
def slucajna_boja():
    return (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))

# ispis u zaglavlju prozora
pygame.display.set_caption("Šareni krugovi")
# definišemo boje koje koristimo u programu
BELA = (255, 255, 255)
# početak programa
pygame.init()
# zadavanje dimanzija ekrana
visina = 400
sirina = 400
ekran = pygame.display.set_mode((visina, sirina))
# bojenje ekrana u belu boju
ekran.fill(BELA)
# pozivanje funkcije display.update() kako bi se sve gore opisane promene
# prikazale na ekranu
pygame.display.update()
```

Primer 53a. Obrada događaja - krugovi pomoću miša.

```
# glavna petlja programa
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
        # ukoliko se desio događaj  MOUSEBUTTONDOWN (kliknuli smo mišem),
        # doćiće do obrade ovog događaja, tj. iscrtaće se krug na ekranu
        # nasumične boje, sa centrom gde je trenutna pozicija
        # miša (dogadjaj.pos) poluprečnika 30.
        if dogadjaj.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.circle(ekran, slucajna_boja(), dogadjaj.pos, 30)
            pygame.display.update()
# kraj programa
pygame.quit()
```

Primer 53bc. Obrada događaja - krugovi pomoću miša.

12. Olovka kojom se može pisati.

```
# cilj ovog programa je da se generiše ekran koji je podloga za
# pisanje dok je klik na levi taster misa olovka koja ostavlja trag
import pygame, random
# definisemo boju koju koristimo, u ovom primeru belom bojom bojimo ekran,
# a olovka ostavlja crni trag
BELA = (255, 255, 255)
CRNA = (0, 0, 0)
#pocetak
pygame.init()
#definisanje sirine i visine ekrana
visina = 400
sirina = 500
ekran = pygame.display.set_mode((visina, sirina))
ekran.fill(BELA)
pygame.display.update()
#glavna petlja
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        print(dogadjaj)
        if dogadjaj.type == pygame.QUIT:
            kraj = True

        if dogadjaj.type == pygame.MOUSEMOTION:
        # ukoliko se pritisnuto levo dugme misa (provera uslova:
        # dogadjaj.buttons[0] == 1)
        # onda ce se u uredjenom paru (px,py) zabeleziti trenutne
        # koordinate misa, a u uredjenom paru (rx, ry) upisujemo vrednosti
        # koje vraca funkcija dogadjaj.rel. ova funkcija vraca za koliko
        # su se promenile koordinate polozaja misa od prethodnog
        # pomeranja do ovog momenta.
```

Primer 54a. Obrada događaja – olovka kojom se može pisati.

```
            if dogadjaj.buttons[0] == 1:
                (px, py) = dogadjaj.pos
                (rx, ry) = dogadjaj.rel
                # ostavljanje traga olovke prilikom kretanja misa
                # postizemo tako sto iscrtavamo linije (tacnije duzi)
                # kojima je pocetak tacka sa koordinatama (px, py),
                # a kraj tacka sa koordinatama (px + rx, py + ry)
                pygame.draw.line(ekran, CRNA, (px, py), (px + rx, py +
                                                                ry), 2)
                pygame.display.update()

pygame.quit()
```

Primer 54b. Obrada događaja – olovka kojom se može pisati.

13. Krugovi na dijagonali – mišem.

```
# cilj primera je da demonstrira kako se već iscrtani krugovi na
# dijagonali mogu uvećati ukoliko na mišu pritisnemo levi taster, odnosno
# mogu se smanjiti ukoliko na mišu pritisnemo desni taster.
# krugovi na dijagonali su različitih boja.

# ukljucili smo pored pygame biblioteke, jos dve potrebne biblioteke za
# rad tokom ovog zadatka
import pygame, math, random

# funkcija kojom se generiše nasumična boja
def slucajna_boja():
    return (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))

# definisemo boje koje koristimo tokom programa
BELA = (255, 255, 255)
CRNA = (0, 0, 0)

# definisanje funkcije za generisanje i crtanje kruga čiji poluprečnik
# zavisi od broja krugova na dijagonali
def krugovi_na_dijagonali(n):
    r = d / (2 * n)
    kx = (r*sirina) / d
    ky = (r*visina) / d
    ekran.fill(BELA)
    for i in range(n):
        pygame.draw.circle(ekran, slucajna_boja(), (int((2*i+1)*kx),
visina - int((2*i+1)*ky)), int(r), 2)
    pygame.display.update()


pygame.init()
# definisanje sirine i visine ekrana
visina = 400
sirina = 500
```

Primer 55a. Obrada događaja – krugovi na dijagonali.

```
# ispis u zaglavlju prozora
pygame.display.set_caption("Krugovi na dijagonali!")
ekran = pygame.display.set_mode((sirina, visina))
# dužina dijagonale ekrana
d = math.sqrt(sirina**2 + visina**2)

# inicijalizacija broja krigova na dijagonali na 10
n = 10
# generisanje i iscrtavanje krugova na dijagonali sa datim brojem krugova
# na samom početku
krugovi_na_dijagonali(n)
# pozivamo objekat sat kojim upravljamo brzinom izvrsavanja glavne petlje
# programa
sat = pygame.time.Clock()

# glavna petlja
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
        # provera koje dugme miša je pritusnuto i obrada dva dugmeta (levi
        # i desni)
        if dogadjaj.type == pygame.MOUSEBUTTONDOWN:
            # oznaka za levi taster miša je 1
            if dogadjaj.button == 1:
                n += 1
                krugovi_na_dijagonali(n)
            # oznaka za desni taster miša je 3
            if dogadjaj.button == 3 and n > 1:
                n -= 1
                krugovi_na_dijagonali(n)
    sat.tick(60)
# kraj programa
pygame.quit()
```

Primer 55bc. Obrada događaja – krugovi na dijagonali.

14. Menjanje sličica mišem.

```
# ovim primerom se demonstrira kako se ekran moze prekriti
# odredenim slikama i kako se kada se klikne na neku od ovih
# slika misem ta ista slika zameni drugom slikom. u prvom delu programa
# je iskoriscen kod iz primera slicice.py kako bi se staticki
# ekran prekrio slicicama (objasnjenje se nalazi u fajlu slicice.py)
import pygame

BELA = (255, 255, 255)

pygame.init()

visina = 600
sirina = 400
ekran = pygame.display.set_mode((sirina, visina))
```

Primer 56a. Menjanje sličica mišem.

```
slika_sirina = slika1.get_width()
slika_visina = slika1.get_height()

#početno postavljanje slika na ekran
y = 0
for vrsta in slike:
    x = 0
    for c in vrsta:
        if c == "1":
            ekran.blit(slika1, (x, y))
        elif c == "2":
            ekran.blit(slika2, (x, y))
        x += slika_sirina
    y += slika_visina

pygame.display.update()

# pocetak glavne petlje
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
        # obradjujemo dogadjaj kada se misem klikne na slicicu
        if dogadjaj.type == pygame.MOUSEBUTTONDOWN:
            # uredjeni par (mx, my) inicijalizujemo na trenutne coordinate
            # misa
            (mx, my) = dogadjaj.pos
            # uredjeni par (k,v) se inicijalizuje na poziciju misa u
            # odgovarajucoj koloni, odnosno vrsti --
            # u redu ispred smo odredili u kojoj koloni, tj vrsti je
            # kliknuto misem
            (k, v) = (mx // slika_sirina, my // slika_visina)
            # ako  vrednost promenljive v, koja je postavljena na vrednost
            # vrste u kojoj se trenutno nalazi zadovoljava navedene uslove
            # (v je manje od ukupnog broja vrsta slike) i ako k,
            # promenljiva u kojoj se belezi kolona u kojoj se trenutno
            # nalazi mis i ako je k manje od duzine stringa kojim je
            # obelezen raspored slika na mapi izvrsice se zamena slika.
            if v < len(slike) and k < len(slike[v]):
                if slike[v][k] == "1":
                    # print("Bilo 1")
                    # slika sa brojem 1 se menja u sliku broj 2, sto se
                    # postize upisivanjem novog karaktera u listu
                    # stringova kojom je opisan raspored slika. tacnije
                    # umesto karaktera 1 na trenutnoj poziciji upisace se
                    # karakter 2
                    slike[v][k] = "2"
                    # slika se prikazuje na ekranu navodjenjem slike koju
                    # zelimo da prikazemo u koordinata gornjeg levog ugla
                    # slike
                    ekran.blit(slika2, (k*slika_sirina, v*slika_visina))
                    pygame.display.update()
```

Primer 56b. Menjanje sličica mišem.

```
                elif slike[v][k] == "2":
                    # print("Bilo 2")
                    # slika sa brojem 2 se menja u sliku broj 1
                    slike[v][k] = "1"
                    # slika se prikazuje na ekranu navodjenjem slike koju
                    # zelimo da prikazemo u koordinata gornjeg levog ugla
                    # slike
                    ekran.blit(slika1, (k*slika_sirina, v*slika_visina))
                    pygame.display.update()
#kraj programa
pygame.quit()
```

Primer 56c. Menjanje sličica mišem.

15. Flappy bird.

```
import pygame, random, math

BELA = (255, 255, 255)
PLAVA = (0, 0, 255)
ZELENA = (0, 255, 0)

pygame.init()
# Podrazumevano, kada se pritisne taster tastature,
# generise se samo jedan dogadjaj KEY_DOWN, a dogadjaj
# KEY_UP se generise nakon otpustanja tastera.
# Narednim pozivom menjamo to podrazumevano ponasanje.
# Prilikom duzeg drzanja tastera generisace se vise
# dogadjaja KEY_DOWN, a nakon otpustanja tastera
# generisace se dogadjaj KEY_UP. Prvi dogadjaj KEY_DOWN
# se generise cim je pritisnut taster. Nakon toga se ceka
# broj milisekundi zadat prvim parametrom funckije set_repeat
# (ovom slucaju 10), a zatim se naredni dogadjaji KEY_DOWN
# generisnu u pravilnim vremenskim intervalima odredjenim
# drugim parametrom funckije key_repeat (u ovom slucaju ce
# se dogadjaji KEY_DOWN generisati na svakih 10 milisekundi,
# sve dok je taster pritisnut).
pygame.key.set_repeat(10, 10)
# sat koji odredjuje ucestalost iscrtavanja
# (i time i brzinu kretanja prepreka)
sat = pygame.time.Clock()

# dimenzija ekrana
dim = (sirina, visina) = (800, 400)
ekran = pygame.display.set_mode(dim)

# parametri ptice koja prolazi kroz prepreke -
# ona je kruznog oblika
ptica_x = sirina // 2
ptica_y = visina // 2
ptica_r = 30

# parametri prepreka koje se pojavljuju
sirina_prepreke = 60
visina_otvora = 130
```

Primer 57a. Flappy bird.

```
# funkcija generise dva pravougaonika koji zajedno
# predstavljaju prepreku kroz koju ptica prolazi
# pravougaonici su odredjeni koordinatama gornjeg
# levog temena, sirinom i visinom
def napravi_prepreku():
    vrh_otvora = random.randint(0, visina - visina_otvora)
    return [[sirina, 0, sirina_prepreke, vrh_otvora],
                [sirina,
                 vrh_otvora + visina_otvora,
                 sirina_prepreke,
                 visina - vrh_otvora - visina_otvora]]
# funkcija kojom se odredjuje predjeni put tekuce
# prepreke nakon kojeg se pojavljuje nova prepreka
def odredi_rastojanje_nove_prepreke():
    # najbrze sto moze, nova prepreka se pojavljuje
    # nakon sto je tekuca presla trecinu sirine ekrana
    # a najredje sto moze, nova prepreka se pojavljuje
    # nakon sto je tekuca prepreka presla ceo ekran
    return random.randint(sirina // 3, sirina)
# provera da li krug sa datim centrom (cx, cy) i
# poluprecnikom r sece horizontalnu duz (x0, y) (x1, y)
def krug_sece_horizontalnu_duz(cx, cy, r, x0, x1, y):
    if cy - r <= y and y <= cy + r:
        d = math.sqrt(r**2 - (cy - y)**2)
        if x0 - d <= cx and cx <= x1 + d:
            return True
# provera da li krug sa datim centrom i poluprecnikom
# sece vertikalnu duz (x, y0) (x, y1)
def krug_sece_vertikalnu_duz(cx, cy, r, x, y0, y1):
    if cx - r <= x and x <= cx + r:
        d = math.sqrt(r**2 - (cx - x)**2)
        if y0 - d <= cy and cy <= y1 + d:
            return True
# provera da li krug sa datim centrom i poluprecnikom
# sece pravougaonik zadat koordinatama
# gornjeg levog temena, sirinom i visinom
def krug_sece_pravougaonik(krug, pravougaonik):
    (cx, cy, r) = krug
    (x0, y0, w, h) = pravougaonik

    # provera sudara sa prednjom ivicom
    if krug_sece_vertikalnu_duz(cx, cy, r, x0, y0, y0 + h):
        return True
    # provera sudara sa donjom ivicom
    if krug_sece_horizontalnu_duz(cx, cy, r, x0, x0 + w, y0 + h):
        return True
    # provera sudara sa gornjom ivicom
    if krug_sece_horizontalnu_duz(cx, cy, r, x0, x0 + w, y0):
        return True

    # proveru preseka sa desnom ivicom preskacemo, jer u igri
    # to nije moguce
    return False
```

Primer 57b. Flappy bird.

```
# krecemo sa jednom pocetnom preprekom
prepreke = napravi_prepreku()
# odredjujemo kada treba da se pojavi naredna prepreka
rastojanje_nove_prepreke = odredi_rastojanje_nove_prepreke()

# glavna petlja
kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
        if dogadjaj.type == pygame.KEYDOWN:
            # strelicom na gore ili razamkom podizemo pticu
            if dogadjaj.key == pygame.K_UP
                  or dogadjaj.key == pygame.K_SPACE:
                ptica_y -= 8
    # Crtamo scenu
    ekran.fill(BELA)
    # Crtamo pticu
    pygame.draw.circle(ekran, PLAVA, (ptica_x, ptica_y), ptica_r)
    # Crtamo prepreke
    for pravougaonik in prepreke:
        pygame.draw.rect(ekran, ZELENA, pravougaonik)
    pygame.display.flip()
    # Ptica pada
    ptica_y += 3
    # Prepreke se pomeraju nalevo
    for pravougaonik in prepreke:
        pravougaonik[0] -= 3
    # Proveravamo da li se poslednja prepreka dovoljno pomerila da bi se
    # ubacila nova prepreka
    if prepreke[-1][0] + prepreke[-1][2] + rastojanje_nove_prepreke
                  < sirina:
        # ubacujemo novu prepreku
        prepreke = prepreke + napravi_prepreku()
        # odredjujemo kada treba da se pojavi naredna prepreka
        rastojanje_nove_prepreke = odredi_rastojanje_nove_prepreke()

    # Izbacujemo prepreke koje su ispale sa levog dela ekrana
    while prepreke[0][0] + prepreke[0][2] < 0:
       prepreke.pop(0)
    # Proveravamo da li je ptica pala na zemlju
    if ptica_y + ptica_r > visina:
        kraj = True
    # Proveravamo da li je ptica udarila u neku prepreku
    for pravougaonik in prepreke:
        if krug_sece_pravougaonik((ptica_x, ptica_y, ptica_r),
                                  pravougaonik):
            kraj = True

    # ogranicavamo brzinu na 50 fps
    sat.tick(50)
# kraj programa
pygame.quit()
```

Primer 57c. Flappy bird.

16. Mačka juri miša

```
import pygame, random, math

# funkcija rotira sliku image oko njenog centra za dati ugao u stepenima
image
def rot_center(image, angle):
    loc = image.get_rect().center
    rot_image = pygame.transform.rotate(image, angle)
    rot_image.get_rect().center = loc
    return rot_image

# inicijalizacija biblioteke pygame
pygame.init()

# sat koji kontrolise brzinu igre
sat = pygame.time.Clock()

# boje i fontovi koje se koriste
BELA = (255, 255, 255)
CRNA = (0, 0, 0)
font = pygame.font.SysFont("Arial", 72)

# ukljucujemo pomeranje likova duzim drzanjem tastera na tastaturi
pygame.key.set_repeat(50, 50)

# kreiramo ekran
(sirina, visina) = (400, 300)
ekran = pygame.display.set_mode((sirina, visina))

# ucitavamo slike misa i macke i postavljamo nijansu providnosti
macka = pygame.image.load('cat.png').convert()
macka.set_colorkey(BELA)
mis = pygame.image.load('mis.png').convert()
mis.set_colorkey(BELA)

# mis je inicijalno postavljen u centar ekrana i okrenut nadesno
mis_x = sirina // 2 - mis.get_width() // 2
mis_y = visina // 2 - mis.get_height() // 2
mis_ugao = 0

# macka je inicijalno postavljena u gornji levi ugao ekrana
macka_x = 0
macka_y = 0

kraj_igre = False

kraj = False
while not kraj:
    for dogadjaj in pygame.event.get():
        if dogadjaj.type == pygame.QUIT:
            kraj = True
```

Primer 58a. Mačka juri miša.

```
        if dogadjaj.type == pygame.KEYDOWN:
            # tasterima na tastaturi pomeramo macku
            macka_brzina = 5
            if dogadjaj.key == pygame.K_LEFT:
                macka_x -= macka_brzina
            elif dogadjaj.key == pygame.K_RIGHT:
                macka_x += macka_brzina
            elif dogadjaj.key == pygame.K_UP:
                macka_y -= macka_brzina
            elif dogadjaj.key == pygame.K_DOWN:
                macka_y += macka_brzina
    if not kraj_igre:
        # pomeramo misa za 5 koraka u smeru u kojem je okrenut
        mis_brzina = 5
        mis_x += int(mis_brzina * math.cos(math.radians(mis_ugao)))
        mis_y -= int(mis_brzina * math.sin(math.radians(mis_ugao)))
        mis_rot = rot_center(mis, mis_ugao)
        # u 10% slucajeva mis menja svoj smer kretanja za +- 20 stepeni
        if (random.randint(1, 100) <= 10):
            mis_ugao += random.randint(-20, -20)
        # proveravamo da li je mis ispao iz ekrana - ako jeste,okrecemo ga
        if mis_x < 0 or mis_x + mis_rot.get_width() > sirina:
            mis_ugao = 180 - mis_ugao
        if mis_y < 0 or mis_y + mis_rot.get_height() > visina:
            mis_ugao = mis_ugao + 180
        # iscrtavamo misa i macku
        ekran.fill(BELA)
        ekran.blit(mis_rot, (mis_x, mis_y))
        ekran.blit(macka, (macka_x, macka_y))
        pygame.display.flip()
        # proveravamo da li je macka dosla do misa tako sto
        # proveravamo da li se njihovi pravougaonici seku
        macka_rect = pygame.Rect(macka_x, macka_y, macka.get_width(),
                                 macka.get_height())
        mis_rect = pygame.Rect(mis_x, mis_y, mis_rot.get_width(),
                               mis_rot.get_height())
        if macka_rect.colliderect(mis_rect):
            # ako jeste, proglasava se kraj igre
            kraj_igre = True
    else:
        # Na centru ekrana se prikazuje poruka da je igra zavrsena
        ekran.fill(BELA)
        tekst = font.render("Kraj!!!", True, CRNA)
        tekst_x = sirina // 2 - tekst.get_width() // 2
        tekst_y = visina // 2 - tekst.get_height() // 2
        ekran.blit(tekst, (tekst_x, tekst_y))
        pygame.display.flip()
    # ogranicavamo brzinu na 24 slike u sekundi
    sat.tick(24)

# iskljucujemo biblioteku pygame
pygame.quit()
```

Primer 58a. Mačka juri miša.