Programiranje za umetnike 1

~ 11 ~

Staša Vujičić Stanković

Teme

Funkcije.

Prostor imena
i opseg važenja promenljivih.

Moduli i biblioteke.

Funkcije

- Upotrebom funkcija problem se razbija na manje, slabo međusobno povezane celine
- Omogućava se razvoj i testiranje nezavisnih celina
  - ➡ Rešenje postaje jednostavnije za sagledavanje
    Sistem se lakše održava

Funkcije

- Kada se neki deo koda u istom ili sličnom obliku ponavlja, pogodno ga je definisati u obliku funkcije.
- Funkcije su imenovani delovi koda.
- Podela koda na manje segmente jasno definisane namene naziva se modularizacija
- Modularizacija je poželjna i neophodna iz više razloga
  - Funkcija se definiše na jednom mestu, a poziva proizvoljan broj puta.
  - Funkcija se može testirati nezavisno od koda u kome će se pozivati.
  - Sve izmene treba vršiti na jednom i samo jednom mestu.

Funkcije

- Svaka funkcija je jednoznačno određena svojim nazivom.
- Pozivanjem funkcije od računara se zahteva da se funkcija izvrši.
- Svaka funkcija može imati nula, jedan ili više argumenata.
  Argumenti bliže određuju način na koji se funkcija izvršava.
  Njima se mogu prosleđivati informacije neophodne za izvršenje naredbe.

Funkcije

- Razlikuju se tri vrste funkcija:
  - Ugrađene funkcije – funkcije definisane u samom programskom jeziku
  - Funkcije koje se uvoze iz spoljašnjih paketa i modula
  - Korisničke funkcije – funkcije koje definišu sami programeri
- Primer funkcije:

```
print("Ja sam tekst koji se ispisuje")
```

Funkcije

```
def naziv_funkcije(opcioni_argumenti):
    """ A documentation string - string dokumentacije """
    # kod funkcije
    # kod funkcije
    # ...
    return opciona_vrednost
```

- Funkcija je blok koda čiji početak označava ključna reč `def` iza koje sledi naziv funkcije i zagrade ( ) sa opcionim parametrima funkcije
- Telo funkcije je blok koda koji sledi nakon :

Funkcije

```
def naziv_funkcije():
  print("Zdravo!!!")

naziv_funkcije()
```

- Da bi se pozvala funkcija navodi se njen naziv za kojim slede zagrade (sa odgovarajućim argumentima)

Argumenti (parametri) funkcije

- Informacije koje se u zagradama prosleđuju funkciji nazivaju se argumenti

```
def pozdrav(kome):
  print("Zdravo {}!".format(kome))

pozdrav("Pero")
pozdrav("Lazo")
```

Argumenti (parametri) funkcije

- Funkcija može imati nula ili proizvoljan broj argumenata razdvojenih zarezima
- Podrazumevano, funkcija mora da bude pozvana sa ispravnim brojem argumenata.

```
def pozdrav(ime, prezime):
  print("Ja sam {} {}!".format(ime, prezime))

pozdrav("Pera", "Peric")
pozdrav("Laza", "Lazic")
```

Funkcije

- Telo funkcije opciono sadrži naredbu povratka iz funkcije `return`
  - Uz return se opciono navodi izraz
    čiji rezultat se vraća pozivajućem (pot)programu
  - Naredba `return` uz koju nije naveden izraz
    ekvivalentna je sa `return None`

```
def naziv_funkcije():
  print("Zdravo!!!")

naziv_funkcije()
```

primer

```
def ispisi_drugove(skup_drugova):
    for drug in skup_drugova:
        print(drug)

def dodaj_druga(drug, skup_drugova):
    skup_drugova.add(drug)

def dodaj_vise_drugova(novi_drugovi, skup_drugova):
    skup_drugova.update(novi_drugovi)

drugovi = set(['Mika', 'Pera', 'Laza', 'Pera', 'Mika'])

ispisi_drugove(drugovi)
print("------------")
dodaj_druga('Jovan', drugovi)
ispisi_drugove(drugovi)
print("------------")
novi_drugovi = ['Nikola', 'Vuk', 'Stefan']
dodaj_vise_drugova(novi_drugovi, drugovi)
ispisi_drugove(drugovi)
```

Promenljivi broj argumenata

- Ukoliko se ne zna unapred tačan broj argumenata koji će biti prosleđen funkciji, stavi se * pre naziva parametra u definiciji funkcije

```
def pozdrav(*drugovi):
  print("Zdravo {} {}!".format(drugovi[0], drugovi[1]))

pozdrav("Pera", "Mika")
pozdrav("Pera", "Mika", "Laza")
```

```
def pozdrav(*drugovi):
  print("Drugovi: ", drugovi)

pozdrav("Pera", "Mika")
pozdrav("Pera", "Mika", "Laza")
```

Promenljivi broj argumenata

- Prvi argument koji odgovara parametru `pocetna_vrednost` je obavezan, dok je proizvoljan broj argumenata u nastavku neobavezan;
  * ispred parametra `ostale_vrednosti` to označava!

```
def suma(pocetna_vrednost, *ostale_vrednosti):
    suma = pocetna_vrednost
    for broj in ostale_vrednosti:
        suma += broj
    return suma

print(suma(0))
print(suma(0, 1, 2))
```

Navođenje naziva argumenta u pozivu

- Dozvoljeno je da redosled argumenata u pozivu funkcije bude drugačiji nego u definiciji funkcije, ukoliko se u pozivu funkcije eksplicitno navede naziv argumenta

```
def pozdrav(ime, prezime, broj_godina):
  print("Ja sam {} {}. Imam {} godina.".format(ime, prezime, broj_godina))

pozdrav(prezime = "Peric", broj_godina = 20, ime = "Pera")
pozdrav("Laza", "Lazic", 15)
```

Izostavljanje argumenta

- Dozvoljeno je da se u pozivu funkcije neki argument izostavi, ukoliko se eksplicitno navede njegova podrazumevana vrednost u definiciji funkcije

```
def pozdrav(ime, prezime, broj_godina=30):
  print("Ja sam {} {}. Imam {} godina.".format(ime, prezime, broj_godina))

pozdrav(prezime = "Peric", broj_godina = 20, ime = "Pera")
pozdrav("Laza", "Lazic")
```

Funkcija kao parametar druge funkcije

```
def racunarska_operacija(operacija, broj):
    rezultat = operacija(broj)
    return rezultat

def uvecanje_za_1(broj):
     return broj+1

def umanjenje_za_1(broj):
     return broj-1

print(racunarska_operacija(uvecanje_za_1, 5))
print(racunarska_operacija(umanjenje_za_1, 5))
```

Još jedan primer

Funkcija bez sadržaja

- Definicija funkcija ne može da bude prazna
- Prema potrebi se može koristiti `pass` da bi se izbeglo dobijanje greške

```
def naziv_funkcije():
    pass
```

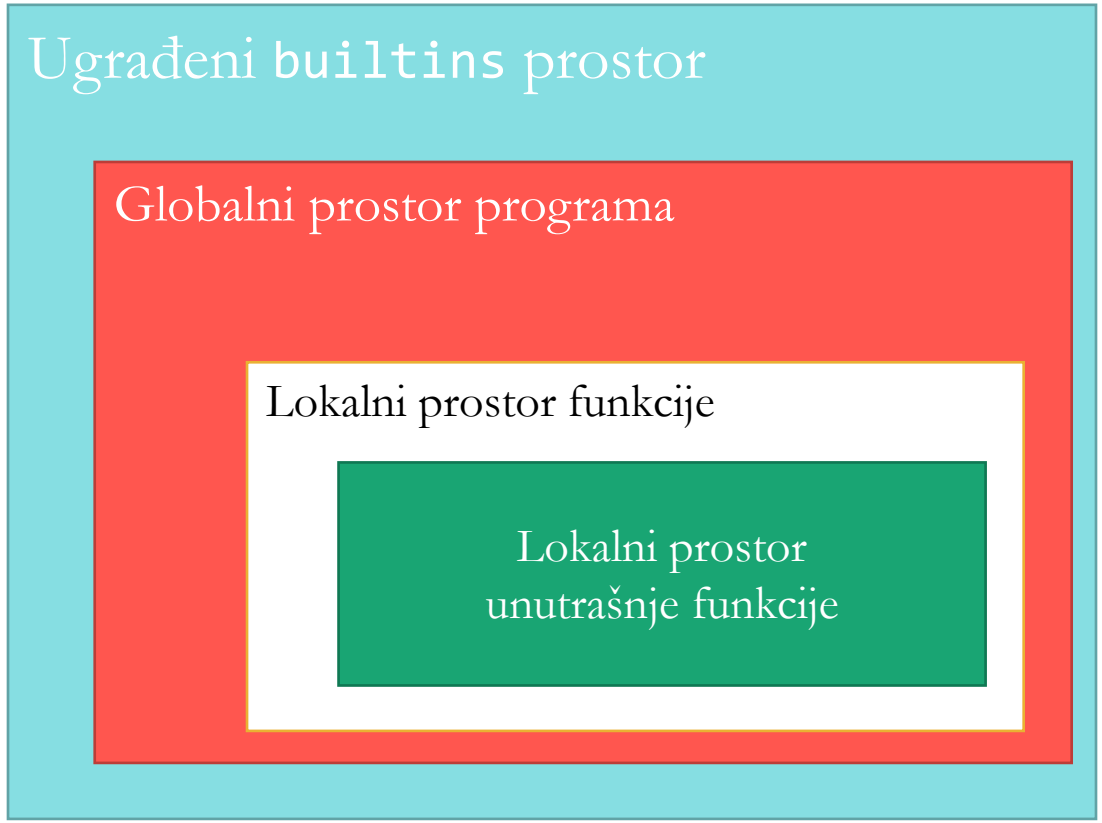
Prostor imena i opseg važenja promenljivih

- Promenljive predstavljaju imena za objekte
- Najčešće se veza imena i objekta uspostavlja dodelom vrednosti
- Prostor imena realizuje se interno tabelom imena
- Tokom izvršavanja programa postoje bar dva (osnovna) prostora imena
- Isto ime može postojati u više prostora imena i tada se odnosi na različite objekte

Prostor imena i opseg važenja promenljivih

- Po pokretanju interpretera, formira se prostor imena ugrađenog `builtins` prostora
- On sadrži predefinisana imena (npr. `print`).
- Kada se program pokrene, kreira se globalni prostor programa
- Pri pozivu svake funkcije kreira se lokalni prostor imena te funkcije
- Svaki od ovih prostora ima svoj životni vek, koji traje tokom izvršavanja odgovarajućeg koda

Prostor imena i opseg važenja promenljivih



- Dozvoljeno je da se funkcija definiše unutar druge funkcije
- Ime se prvo traži u lokalnom prostoru imena
- Ako se ne nađe, ide se nivo više u hijerarhiji prostora imena

Lokalne i globalne promenljive

```
def nova_funkcija():
    promenljiva = 10
    def unutrasnja_funkcija():
        promenljiva = 20
        print("Unutrasnja funkcija:", promenljiva)

    unutrasnja_funkcija()
    print("Spoljasnja funkcija:", promenljiva)

nova_funkcija()
```

Lokalne i globalne promenljive

- Globalnoj promenljivoj može se pristupiti iz svih funkcija
- Deklariše se ključnom rečju `global`
- Oprezno upotrebljavati globalne promenljive!

Lokalne i globalne promenljive

```
def nova_funkcija():
    global promenljiva
    promenljiva = 10
    def unutrasnja_funkcija():
        global promenljiva
        promenljiva = 20
        print("Unutrasnja funkcija:", promenljiva)

    unutrasnja_funkcija()
    print("Spoljasnja funkcija:", promenljiva)

nova_funkcija()
```

Modularnost

- Rekli smo, ukoliko je programski sistem složen
  može se podeliti na manje delove, na logičke celine kojima se lakše rukuje
- Modularne sisteme je lakše testirati, nadograđivati i održavati
- Dekompozicija može biti:
  - Na funkcionalnom nivou (funkcije)
  - Na sistemskom nivou (klase, odnosno objekti)
  - Na organizacionom nivou (moduli i biblioteke)

Modularnost

- Skripta je program smešten u jednoj datoteci koji se pokreće u komandnom režimu operativnog sistema
- Modul je datoteka sa ekstenzijom .py koja sadrži proizvoljan izvorni kod
  - Najčešće sadrži definicije funkcija i definicije globalnih promenljivih
  - Nije namenjena samostalnom izvršavanju,
    već kao skup funkcija za složenije programe
- Potrebno je omogućiti da se funkcija definisana u jednom modulu poziva iz programa

Modularnost

- Da bi se modul koristio u programu upotrebljava se naredba `import`
- Učitavaju se naredbe modula i kreira novi imenski prostor za taj modul (`dir(naziv_modula)`)
- Funkcije i globalne promenljive postaju vidljive upotrebom notacije

```
import math

rezultat = math.sqrt(100)
print(rezultat)
```

Modularnost

- Može se uvesti i alternativno ime za modul

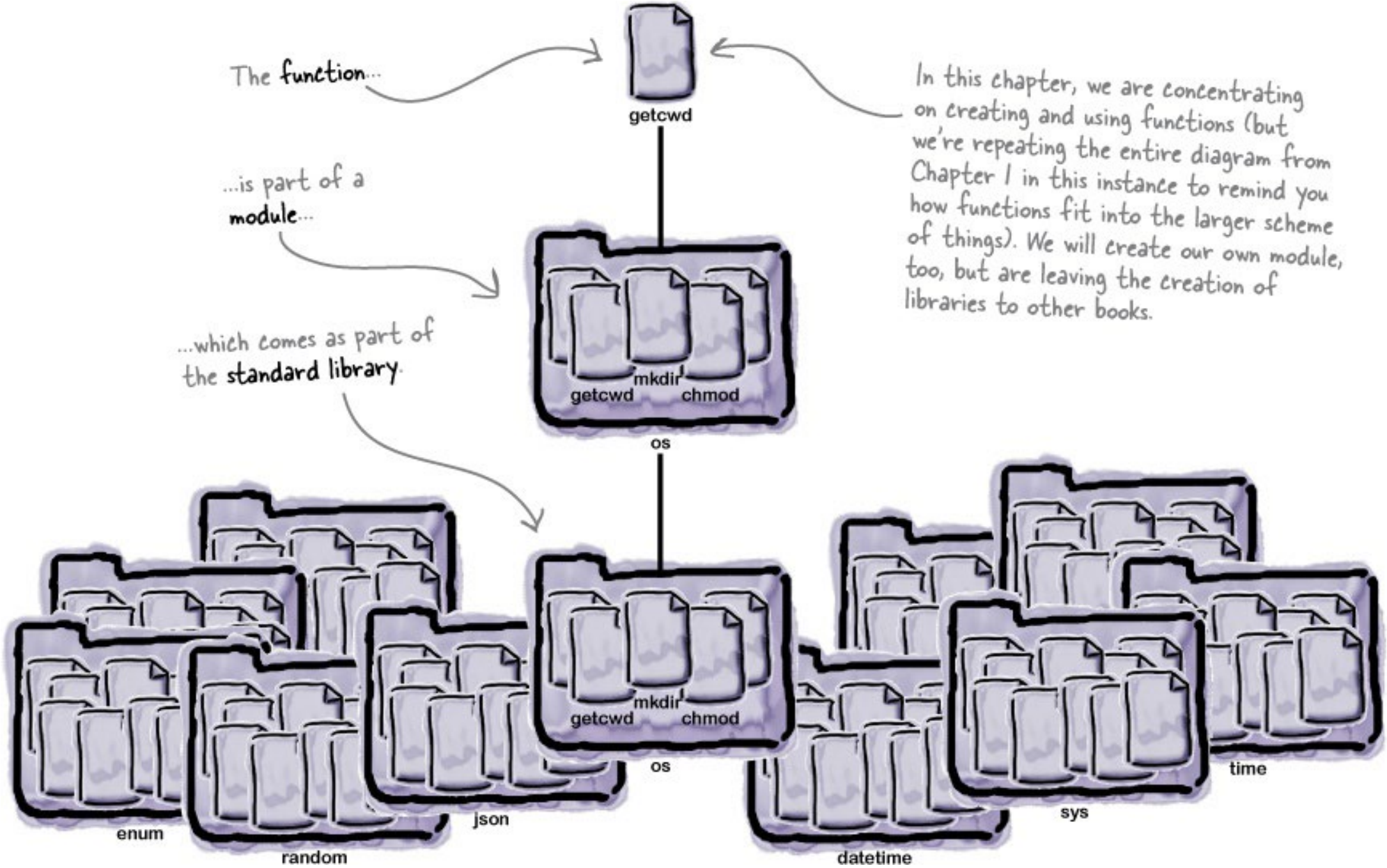
```
import math as m

rezultat = m.sqrt(100)
print(rezultat)
```

Koncept biblioteka

- Biblioteka organizuje više modula u jedinstvenu imensku hijerarhiju
- Moduli se nalaze u zajedničkom direktorijumu kao .py fajlovi
  Direktorijum sadrži obavezno i datoteku `__init__.py`
- Datoteka `__init__.py` može da bude i prazna
  (inače se automatski izvrši pri uvođenju paketa)
- Omogućena je upotreba različitih biblioteka
  koje sadrže module sa istim imenom
- Na primer `paket.modul.fja(arg)`

Python paketi



Slika preuzeta iz knjige „Head First Python“ LINK

Literatura

- Python 3.10.0 documentation
- Barry, Paul. Head First Python, 2nd Edition. O'Reilly Media, Inc., 2016.
  (postoji prevod na srpski jezik u izdanju CET-a)
  *** pogledajte prvo poglavlje ***

Hvala

Staša Vujičić Stanković

stasa.vujicic.stankovic@math.rs

www.matf.bg.ac.rs/~stasa.vujicic.stankovic