Funkcije

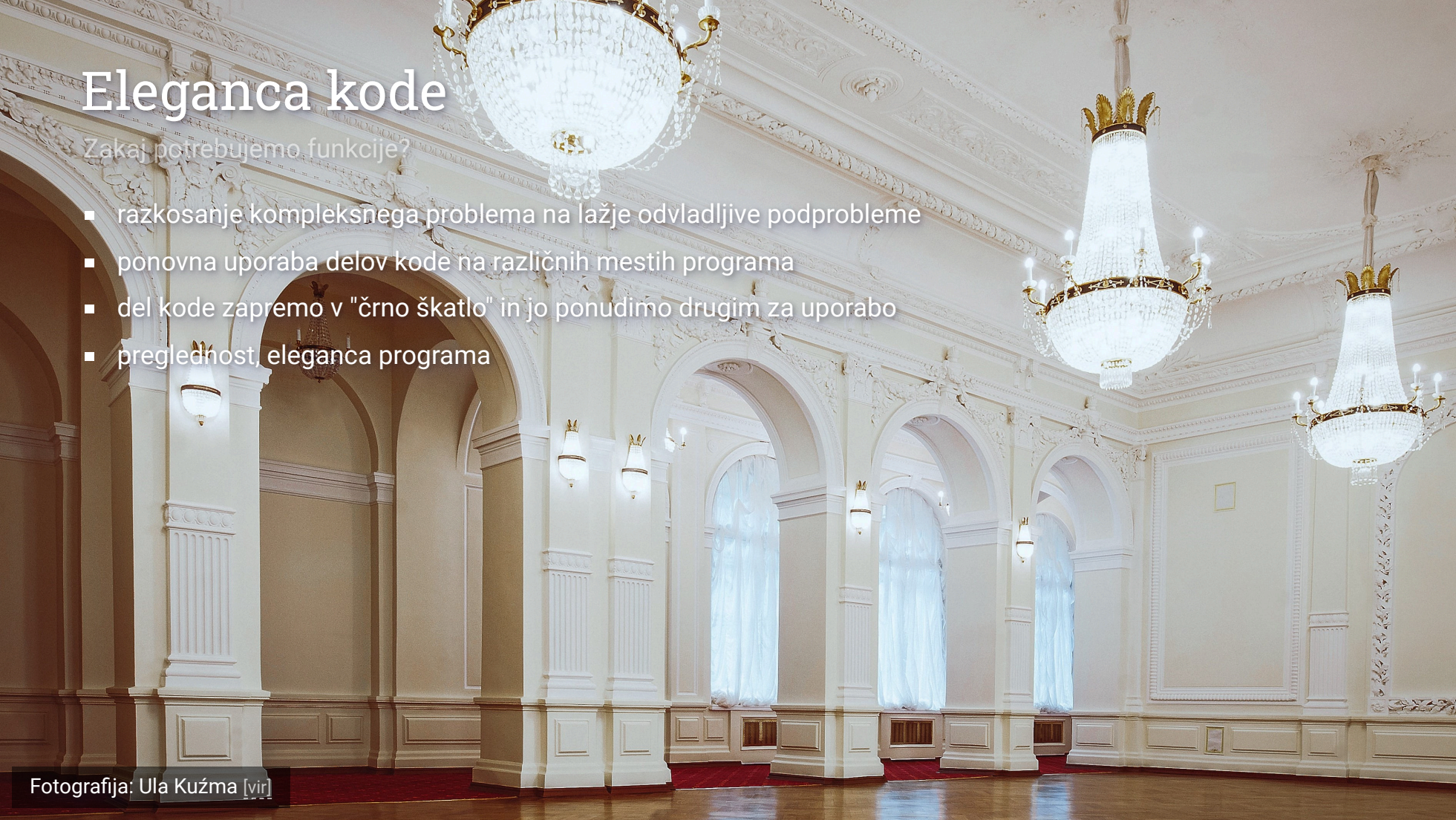
Osnove programiranja

Nejc Ilc

Eleganca kode

Zakaj potrebujemo funkcije?

- razkosanje kompleksnega problema na lažje odvladljive podprobleme
- ponovna uporaba delov kode na različnih mestih programa
- del kode zapremo v "črno škatlo" in jo ponudimo drugim za uporabo
- preglednost, eleganca programa



Fotografija: Ula Kuźma [vir]

Sintaksa

Osnovna definicija funkcije

```
def ime_funkcije():
    stavek_1
    ...
    stavek_n
```

- Funkcijo definiramo z rezervirano besedo `def`.
- Funkciji damo kakšno lepo ime. Gotovo kaj bolj spevnega kot `ime_funkcije`.
- Okrogli oklepaji za imenom so obvezni.
- Prav tako dvopičje.
- V telesu funkcije napišemo zaporedje stavkov, ki naj se izvedejo, ko to funkcijo pokličemo.

Primer

Pozdravimo publiko

```
def pozdrav_vsem():
    print('Dober dan, vsi!')

# Glavni del programa
pozdrav_vsem()
```

Kaj se dogaja ob zagonu programa?

1. Python prebere definicijo funkcije (prvi dve vrstici), vendar je ne izvede.
2. Preskoči komentar in izvede stavek, ki pokliče funkcijo `pozdrav_vsem()`.
3. Vstopimo v funkcijo, ki izpiše `Dober dan, vsi!`.
4. Funkcija se zaključi in prav tako program.

Sintaksa

V definicijo funkcije dodajmo argumente

```
def ime_funkcije(arg1,..., argN):
    stavek_1
    ...
    stavek_n
```

- Funkcija lahko sprejme argumente. To so spremenljivke, ki smo jih zgoraj poimenovali `arg1` do `argN`.
- Argumente lahko uporabimo v stavkih znotraj funkcije.

Primer `pozdravcek`

Pozdravimo konkretno osebo

```
def pozdrav(ime):
    print('Dober dan,', ime)

pozdrav('Ana')
```

1. Definicija funkcije `pozdrav()` pravi, da funkcija potrebuje en argument, ki ima splošno ime `ime`.
2. Ob klicu funkcije moramo zato določiti ta argument. Odločili smo se za vrednost `'Ana'`.
3. Ko vstopimo v funkcijo, spremenljivka `ime` dobi vrednost `'Ana'` in izpiše: `Dober dan, Ana`.

Sintaksa

Argumenti imajo lahko privzete vrednosti.

```
def ime_funkcije(arg1, arg2=dfl):
    stavek_1
    ...
    stavek_n
```

- Argumenti funkcije so lahko opcijski, se pravi, da imajo že neko privzeto vrednost. Če pri klicu funkcije določimo vrednost argumenta, se privzeta vrednost ne upošteva.
- Zgoraj je argument `arg1` obvezen - ob klicu funkcije ga moramo določiti.
- Argument `arg2` je opcijski, ob klicu dobi vrednost `dfl`, razen, če smo v klicu funkcije določili drugače.

Primer `pozdravcek`

Pozdrav prilagodimo uri

```
def pozdrav_ura(ime, ura=12):
    if 5 <= ura < 9:
        print('Dobro jutro,', ime)
    elif 9 <= ura <= 17:
        print('Dober dan,', ime)
    else:
        print('Dober večer,', ime)

pozdrav_ura('Ana')    # Dober dan, Ana
pozdrav_ura(ime='Ana')
pozdrav_ura('Ana', 7) # Dobro jutro, Ana
pozdrav_ura(ura='7', ime='Ana')
```

Ob klicu funkcije lahko argumente poimenujemo. Če uporabljamo imena, lahko vrstni red argumentov tudi mešamo.

Sintaksa

Funkcija vrne rezultat

```
def ime_funkcije(arg1, arg2=dfl):
    stavek_1
    ...
    stavek_n
    return <vrednost>
```

- Funkcija se konča, ko:
  - izvede vse stavke v telesu ali
  - ko pride do stavka `return`.
- `return` je podoben `break`, ki prekine zanko
- Funkcija vedno nekaj vrne.
- Če napišemo samo `return` ali če ga sploh ne omenjamo, bo funkcija vrnila `None`. Če napišemo `return vrednost`, bo vrnila `vrednost`.

Primer `ploscina`

Izračunajmo ploščino likov

```
from math import pi

def ploscina_kvadrat(a):
    s = a*a
    return s

def ploscina_krog(r):
    s = pi * r**2
    return s

def ploscina(x, lik):
    if lik == 'kvadrat':
        # U, funkcija kliče drugo funkcijo!
        return ploscina_kvadrat(x)
    elif lik == 'krog':
        return ploscina_krog(x)
    else:
        print('Ne znam za', lik, ':(')
        # Tu ni stavka `return` in funkcija vrne `None`
```

Enosmerno steklo: ko si zunaj, ne vidiš noter; ko si notri, vidiš tudi ven. [vir]

Globalno in lokalno `globalno_lokalno`

"Razmišljaj globalno, deluj lokalno" :)

"Prostor" zunaj funkcije se imenuje globalni imenski prostor. Funkcija ima svoj imenski prostor, ki mu pravimo lokalni imenski prostor.

```
a = 1

def f(x):
    b = 2
    return x + a + b

print(f(1)) # izpiše 4

a = 2
print(f(1)) # izpiše 5

print(b)    # Napaka: NameError: name 'b' is not defined
```

Vizualizacija s Python Tutor

Enosmerno ogledalo

Funkcija je kot stranišče sredi ulice na prejšnji strani. Obdano je s posebnim steklom, ki prepušča svetlobo samo v eni smeri.

- Ko si **zunaj** funkcije, ne vidiš njenih lokalnih spremenljivk. Vidiš pa globalne spremenljivke.
- Ko si **znotraj** funkcije, vidiš njene lokalne spremenljivke in tudi vse, kar je zunaj, torej ves globalni imenski prostor.
- Znotraj funkcije lahko spreminjaš lokalne (itak) in globalne spremenljivke (uau!).

Python 3.6
known limitations

```
1  a = 1
2
3  def f(x):
4      b = 2
5      return x + a + b
6
7  print(f(1))
8  a = 2
9
10 print(f(1))
11
12 print(b)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 13 of 14

Vizualizacija s Python Tutor [vir]

Print output (drag lower right corner to resize)

```
4
```

Frames Objects

Global frame

a	2
f	

function
f(x)

f

x	1
b	2
Return value	5

Spreminjanje globalne spremenljivke v funkciji

```
a = 1

def f(x):
    b = 2
    # Ali lahko spremenimo globalno spremenljivko v funkciji?
    # a = a + 1 # Tole ne bo šlo, napaka.
    # a = 2 # To gre, vendar naredimo novo lokalno spremenljivko
          # z istim imenom, zato tudi izgubimo dostop do globalne
    # Tako je prav, povedati moramo, da mislimo globalno
    global a
    a = a + 1
    return x + a + b

print('f(1):', f(1)) # f(1): 4
print('a: ', a)      # a: 2

a = 2
print('f(1):', f(1)) # f(1): 6
print('a: ', a)      # a: 3
```

Praštevila

prastevila_funkcija

Povadimo koncepte

```
def je_prastevilo(x):
    """Preverimo, ali je število `x` praštevilo."""

    if x < 1:
        print('Število mora biti večje od 0.')
        return False

    for i in range(2,x):
        # Če je x deljiv z i (ki gre od 2 do x-1),
        # potem x ni praštevilo.
        if x % i == 0:
            # Ni praštevilo
            return False
    else:
        # Sem pridemo, ko se zanka konča,
        # kar pomeni, da x je praštevilo
        # (sicer bi že prej klicali return)
        return True
```

```
def izpisi_prastevila():
    """
    Izpiše vsa praštevila do n.
    Število n je definirano globalno
    """

    print('Vsa praštevila do', n)
    for m in range(1, n+1):
        if je_prastevilo(m):
            print(m)

n = int(input('Vnesi število n: '))
izpisi_prastevila()
```

Fotografija: Farzan Lelinwalla [vir]

Otroško igrišče

Ko se otroci zabavajo na toboganu, mi merimo kamenčke ...

Primer postopne nadgradnje programa:

- `povprecje_v1`: program za računanje povprečja brez ene same funkcije. Šparta!
- `povprecje_v2`: program za računanje povprečja z vgrajenimi funkcijami. Olajšanje!
- `povprecje_v3`: napišemo lastno funkcijo za računanje povprečja. Razodetje!
- `povprecje_v4`: dodamo funkcijo za računanje prostornine kamenčkov in ta dela tudi na seznamih. Hudo!