

# Računari i programi

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2018.

# Ciljevi

- razumevanje uloge hardvera i softvera u računarskom sistemu
- upoznavanje sa predmetom izučavanja računarstva i tehnikama koje se koriste
- razumevanje osnovnih principa rada modernog računara
- razumevanje oblika i svrhe programskih jezika
- početak korišćenja Python programskog jezika

# Univerzalna mašina

- moderan računar:  
„mašina koja skladišti i manipuliše informacijama pod kontrolom izmenljivog programa“
- dva ključna elementa:
  - računari su uređaji za manipulisanje informacijama
  - računari funkcionišu pod kontrolom izmenljivog programa

# Univerzalna mašina <sub>2</sub>

- šta je računarski program?
  - detaljan, korak-po-korak skup instrukcija koje govore računaru šta da radi
  - ako izmenimo program, računar će izvršavati drugačiji skup operacija
  - mašina ostaje ista, ali se program menja

# Univerzalna mašina <sub>3</sub>

- programi se izvršavaju
- svi računari imaju istu moć uz odgovarajuće programiranje, tj. svaki računar može da uradi ono što i bilo koji drugi
  - ...u principu

# Moć programa

- **softver** (programi) upravlja **hardverom** (fizičkom mašinom)
- proces kreiranja softvera zove se **programiranje**
- zašto učiti programiranje?
  - bazično znanje iz računarstva
  - poznavanje programiranja pomaže razumevanju mogućnosti i ograničenja računara

# Moć programa 2

- zašto učiti programiranje?
  - pomaže čoveku da bude bolji korisnik računara
  - može biti zabavno
  - oblik izražavanja
  - pomaže razvoju veštine rešavanja složenih problema, naročito kod analize složenih sistema i njihovog rastavljanja na manje delove
  - tražena profesija

# Način razmišljanja

Pošalje žena muža programera u prodavnicu...

- Kupi margarin, a ako ima jaja - kupi 10!

Muž ode u radnju, vraća se kući, stavi na sto 10 margarina i kaže: Ima jaja.



# Šta je računarstvo?

- nije izučavanje računara:  
„Computers are to computer science what telescopes are to astronomy“ – E. Dijkstra
- → kakve procese možemo opisati?
- → šta se može izračunati?

# Šta je računarstvo? <sub>2</sub>

- dizajn
  - način da se dokaže da se neki problem može rešiti je da se napravi rešenje
  - to se radi razvojem **algoritma**, korak-po-korak niza instrukcija koje dovode do rešenja
  - problem – na ovaj način možemo dokazati da rešenje postoji, ali ne i da ne postoji

# Šta je računarstvo? <sub>3</sub>

- analiza
  - analiza je matematički postupak ispitivanja problema i algoritama
  - neki naizgled jednostavni problemi ne mogu se rešiti algoritmom – to su **nerešivi** problemi
  - problemi mogu biti **neizračunljivi** ako se njihova rešenja izvršavaju suviše dugo ili koriste previše memorije

# Šta je računarstvo? <sub>4</sub>

- eksperimenti
  - neki problemi su suviše komplikovani za analizu
  - napravi računarski sistem (hw+sw) pa posmatraj njegovo ponašanje

# Osnove hardvera

- procesor (**centralna procesna jedinica**, CPU) je „mozak“ računara
  - CPU izvršava sve osnovne operacije nad podacima
  - npr. jednostavne aritmetičke operacije, poređenje brojeva

# Osnove hardvera 2

- memorija skladišti programe i podatke
  - CPU može da direktno pristupi samo podacima u **RAM** (random access memory)
  - RAM je brza, ali ne pamti podatke nakog gubitka napajanja
  - sekundarna memorija trajno pamti podatke, veći kapacitet, sporija – magnetna, optička, elektronska

# Osnove hardvera <sub>3</sub>

- ulazni uređaji
  - podaci se unose u računar putem tastature, miša, itd.
- izlazni uređaji
  - obrađeni podaci se prikazuju čoveku putem monitora, štampača, itd.

# Osnove hardvera 4

- ciklus **dobavi-i-izvrši**
  - pročitaj instrukciju iz RAM
  - dekodiraj instrukciju – da se odredi šta ona predstavlja
  - izvrši potrebnu akciju
  - naredna instrukcija dobavljena, dekodirana, izvršena
  - ...ponavlja



# Programski jezici

- prirodni jezik može biti neprecizan i dvosmislen kada se opisuju složeni algoritmi
  - programi koji su pisani precizno i nedvosmisleno koriste [programske jezike](#)
  - svaka struktura u programskom jeziku ima precizan oblik – [sintaksu](#)
  - svaka struktura u programskom jeziku ima precizno značenje – [semantiku](#)

# Programski jezici 2

- programski jezik je kao **kôd** za zapisivanje instrukcija
  - programeri ga obično zovu **programski kôd**
  - pisanje algoritama na programskom jeziku se zove **kodiranje**

# Programski jezici <sub>3</sub>

- programski jezici visokog nivoa
  - dizajnirani tako da ih razume i koristi čovek
- programski jezici niskog nivoa
  - hardver može da razume samo mašinski jezik

# Programski jezici 4

- saberi dva broja
  - učitaj broj iz memorijske lokacije br. 2500 u CPU
  - učitaj broj iz memorijske lokacije br. 2501 u CPU
  - saberi dva broja u CPU
  - uskladišti rezultat u lokaciju br. 2502
- ovakve instrukcije niskog nivoa su predstavljene binarnim kodom  $\{0,1\}$

# Programski jezici 5

- na jeziku visokog nivoa:  
 $c = a + b$
- ovo se mora prevesti na mašinski jezik koga računar može da izvrši
- **kompajleri** (prevodioci) prevode program sa jezika visokog nivoa na mašinski jezik nekog računara

# Programski jezici 6

- **interpreteri simuliraju računar koji razume jezik visokog nivoa**
- izvorni program se ne prevodi na mašinski jezik odjednom
- interpreter analizira i izvršava izvorni program instrukciju po instrukciju

# Programski jezici 7

## • kompajleri vs interpreteri

- jednom prevedeni program se može kasnije izvršavati bez izvornog koda i kompajlera
- za interpretirani program svaki put je potreban izvorni kod i interpreter
- kompajlirani programi se (u principu) brže izvršavaju jer se prevođenje obavlja samo jednom

# Programski jezici 8

- **kompajleri** vs **interpreteri**
  - interpreterski jezici su fleksibilniji jer se mogu pisati i izvršavati interaktivno (naredbu po naredbu)
  - interpretirani programi su **prenosivi** – isti program se može izvršavati na različitim računarima, dok postoji interpreter



# Python

- kada se pokrene Python, vidi se nešto kao:

```
Python 3.7.4 (default, Sep  7 2019, 18:27:02)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# Python 2

- znaci >>> predstavljaju **prompt** – Python je spreman da primi naredbu

```
>>> print("Hello, world")
```

```
Hello, world
```

```
>>> print(2+3)
```

```
5
```

```
>>> print("2+3=", 2+3)
```

```
2+3= 5
```

```
>>>
```

# Python 3

- često nam je potrebno da izvršimo nekoliko naredbi odjednom da bismo rešili neki problem
- jedan način da to uradimo je da napravimo **funkciju**:

```
>>> def zdravo():  
    print("Zdravo")  
    print("Kako si?")
```

# Python 4

```
>>> def zdravo():  
    print("Zdravo")  
    print("Kako si?")
```

```
>>>
```

- prva linija kaže da definišemo funkciju koja se zove zdravo
- naredne linije su uvučene da pokažu da su deo funkcije zdravo
- prazan red (pritisnut enter dva puta) kaže Pythonu da je definicija završena

# Python 5

```
>>> def zdravo():  
    print("Zdravo")  
    print("Kako si?")
```

```
>>>
```

- ništa se do sada nije desilo!
- definisali smo funkciju ali nismo rekli Pythonu da je izvrši
- funkcija se **poziva** po imenu:

```
>>> zdravo()  
Zdravo  
Kako si?  
>>>
```

# Python 6

- čemu služe zagrade?
- funkcije mogu zavisiti od nekih podataka koje zovemo **parametri** koje navodimo između ( i )

```
>>> def greet(person):  
    print("Hello", person)  
    print("How are you?")
```

```
>>>
```

# Python 7

```
>>> greet("Pera")
Hello Pera
How are you?
>>> greet("Mika")
Hello Mika
How are you?
```

- kada koristimo parametre, rezultat rada funkcije može da zavisi od njih

# Python 8

- kada prekinemo Python, funkcije koje smo napravili više ne postoje!
- programi se tipično sastoje iz funkcija, **modula** ili **skriptova** koji se čuvaju na disku da bi se mogli ponovo koristiti
- **modul** fajl je tekst-fajl napisan u editoru teksta koji sadrži programski kod
- **programersko okruženje** (programming environment) je softver koji pomaže programerima da efikasnije pišu programe



# Python 9

```
# chaos.py  
# Ovaj program simulira haotično ponašanje  
  
print("Ovaj program simulira haotično ponašanje")  
x = eval(input("Unesite broj između 0 i 1: "))  
for i in range(10):  
    x = 3.9 * x * (1 - x)  
    print(x)
```

- snimićemo fajl sa ekstenzijom `.py` da označimo Python program
- pokrenućemo program sa `python chaos.py`

# Python 10

```
$ python chaos.py
```

Ovaj program simulira haotično ponašanje

Unesite broj između 0 i 1: 0.5

0.975

0.09506250000000008

0.33549992226562525

0.8694649252590003

0.44263310911310905

0.962165255336889

0.1419727793616139

0.4750843861996143

0.9725789275369049

0.1040097132674683

# Analiza Python programa

```
# chaos.py
```

```
# Ovaj program ilustruje haotično ponašanje.
```

- redovi koji počinju sa # su komentari
- namenjene čoveku, Python ih ignoriše
- Python preskače tekst od # do kraja reda

# Analiza Python programa 2

```
print("Ovaj program ilustruje haotično ponašanje")
```

- ovaj red kaže Pythonu da ispiše poruku koja opisuje program

# Analiza Python programa <sub>3</sub>

```
x = eval(input("Unesite broj između 0 i 1: "))
```

- x je primer **promenljive**
- promenljiva se koristi kako bi se nekom **imenu** dodelila **vrednost**
- možemo se na tu promenljivu kasnije pozvati po imenu
- tekst pod navodnicima se ispisuje, i uneti broj se upisuje u promenljivu x.

# Analiza Python programa 4

```
for i in range(10):
```

- for predstavlja **petlju**
- petlja govori Pythonu da komande ponavlja više puta
- u ovom primeru, kod koji sledi će se ponoviti 10 puta

# Analiza Python programa <sub>5</sub>

```
x = 3.9 * x * (1 - x)
print(x)
```

- ovi redovi predstavljaju **telo petlje**
- telo petlje je ono što se izvršava više puta
- naredbe koje čine telo petlje su uvučene u desno
- efekat petlje je kao da smo ove dve naredbe ponovili 10 puta

# Analiza Python programa <sub>6</sub>

- ovo je ekvivalentno

```
for i in range(10):  
    x = 3.9 * x * (1 - x)  
    print(x)
```

```
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)  
print x  
x = 3.9 * x * (1 - x)
```



# Analiza Python programa 7

```
x = 3.9 * x * (1 - x)
```

- ovo je naredba **dodele**
- izraz sa desne strane = je matematički izraz
- \* označava množenje
- kada se vrednost na desnoj strani izračuna, **dodeljuje se** (upisuje se) promenljivoj x

# Analiza Python programa <sub>8</sub>

```
def main():  
    print("Ovaj program ilustruje haotično ponašanje")  
    x = eval(input("Unesite broj između 0 i 1: "))  
    for i in range(10):  
        x = 3.9 * x * (1 - x)  
        print(x)
```

main()

- sada smo ceo prethodni program smestili u funkciju main
- program sadrži definiciju funkcije i njen poziv