Sekvence: stringovi, liste, fajlovi

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2014.

Ciljevi

- razumevanje stringova i njihove reprezentacije u računaru
- poznavanje operacija nad stringovima
- poznavanje pojma sekvence i indeksa, primena na stringove i liste
- veština formatiranja stringova

Ciljevi 2

- poznavanje osnovnih koncepata rada sa fajlovima
- poznavanje tehnika za čitanje i pisanje fajlova u Pythonu
- razumevanje i pisanje programa koji rukuju tekstualnim fajlovima

Tip podataka string

- obrada teksta je najčešći oblik korišćenja računara
- u programima tekst je predstavljen tipom string
- string je niz (sekvenca) znakova
- navodi se unutar jednostrukih (') ili dvostrukih (") navodnika

Tip podataka string 2

```
>>> str1="Hello"
>>> str2='spam'
>>> print(str1, str2)
Hello spam
>>> type(str1)
<class 'str'>
>>> type(str2)
<class 'str'>
```

Tip podataka string 3

- unos stringa sa tastature

```
>>> firstName = input("Please enter your name: ")
Please enter your name: Žika
>>> print("Hello", firstName)
Hello Žika
```

- uneti string nije eval-uiran – želimo da memorišemo unete znakove, a ne da ih interpretiramo kao Python izraz

String i pojedinačni znakovi

- možemo pristupiti pojedinačnim znakovima u stringu pomoću indeksa
- indeks predstavlja redni broj znaka u stringu
- brojanje počinje od 0
- opšti oblik je `<string>[<expr>]` gde vrednost `expr` određuje izabrani znak u stringu

String i pojedinačni znakovi 2

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```

String i pojedinačni znakovi $_3$

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

- u stringu od n znakova, poslednji znak je na poziciji $n-1$ jer brojanje počinjemo od 0
- možemo brojati i sa desnog kraja pomoću negativnih brojeva

```
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```

String i pojedinačni znakovi $_4$

- indeksiranje vraća string koji sadrži tačno jedan znak
- možemo izdvojiti i duži podniz znakova iz stringa – podstring (substring)
- postupak isecanja (slicing)

String i pojedinačni znakovi 5

- isecanje:
  `<string>[<start>:<end>]`
- `start` i `end` moraju biti tipa `int`
- isečak (slice) sadrži podstring od `start`-nog znaka (uključivo) do `end`-nog (isključivo)

String i pojedinačni znakovi 6

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

```
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```

- ako nedostaju `start` ili `end` podrazumevaju se početak odnosno kraj stringa

Konkatenacija i ponavljanje stringa

- konkatenacija predstavlja „spajanje“ stringova – kraj prvog stringa sa početkom drugog
- operator za konkatenaciju je +

- ponavljanje predstavlja višestruku konkatenaciju stringa sa samim sobom
- operator za ponavljanje je *

Konkatenacija i ponavljanje stringa 2

```
>>> "spam" + "eggs"
'spameggs'
>>> "Spam" + "And" + "Eggs"
'SpamAndEggs'
>>> 3 * "spam"
'spamspamspam'
>>> "spam" * 5
'spamspamspamspamspam'
>>> (3 * "spam") + ("eggs" * 5)
'spamspamspameggseggseggseggseggs'
```

Dužina stringa

- funkcija `len` vraća dužinu (broj znakova) stringa

```
>>> len("spam")
4
>>> for ch in "Spam!":
        print (ch, end=" ")

S p a m !
```

Operacije nad stringom

operator	**značenje**
`+`	konkatenacija
`*`	ponavljanje
`<string>[]`	indeksiranje
`<string>[:]`	isecanje
`for <var> in <string>`	iteracija kroz znakove

Primeri obrade stringova

- korisničko ime u nekom računarskom sistemu: prvo slovo imena i prvih 7 slova prezimena

```
first = input("Unesite ime: ")
last = input("Unesite prezime: ")

username = first[0] + last[:7]
```

Formiranje korisničkog imena

```
Unesite ime: ana
Unesite prezime: tot
username = atot

Unesite ime: dimitrije
Unesite prezime: dimitrijević
username = ddimitri
```

Redni broj meseca u godini → troslovna oznaka

- smestimo sve troslovne oznake u jedan veliki string: `"JanFebMarAprMajJunJulAvgSepOktNovDec"`
- koristimo redni broj meseca kao indeks za string: `mesecSkr = meseci[pos:pos+3]`
- da bismo izračunali `pos`, treba od broja meseca oduzeti 1 i pomnožiti to sa 3

mesec	**broj**	`pos`
Jan	1	0
Feb	2	3
Mar	3	6
Apr	4	9

Redni broj meseca u godini → troslovna oznaka 2

```
# month.py
# Štampa skraćeno ime meseca za dati redni broj

# ovde čuvamo nazive
meseci = "JanFebMarAprMajJunJulAvgSepOktNovDec"

n = eval(raw_input("Unesite mesec (1-12): "))

# izračunaj indeks u stringu
pos = (n-1) * 3

# iseci podstring
mesecSkr = meseci[pos:pos+3]

# ispiši rezultat
print ("Skraćenica je", mesecSkr + ".")
```

Redni broj meseca u godini → troslovna oznaka 3

```
$ python month.py
Unesite mesec (1-12): 1
Skraćenica je Jan.
$ python month.py
Unesite mesec (1-12): 12
Skraćenica je Dec.
```

- ovo rešenje je ispravno samo ako su nazivi svih meseci iste dužine
- kako da to prevaziđemo?

Stringovi, liste, sekvence

- string je specijalni oblik sekvence
- string operacije postoje za sve sekvence!

```
>>> [1,2] + [3,4]
[1, 2, 3, 4]
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
>>> len(grades)
5
```

Stringovi, liste, sekvence 2

- string je uvek sekvenca znakova
- lista može da sadrži bilo kakve podatke – brojeve, stringove, ili oba

```
myList = [1, "Spam ", 4, "U"]
```

Redni broj meseca u godini $\longrightarrow$ troslovna oznaka $_4$

- string je uvek sekvenca znakova
- lista može da sadrži bilo kakve podatke – brojeve, stringove, ili oba

```
meseci = ["Jan", "Feb", "Mar", "Apr", "Maj", "Jun",
    "Jul", "Avg", "Sep", "Okt", "Nov", "Dec"]
```

- naredba dodele vrednosti se proteže preko dva reda
- Python zna da nije završena u prvom redu pa dalje čita tekst programa

Redni broj meseca u godini → troslovna oznaka 5

- sada se naziv meseca dobija ovako:

```
mesecSkr = meseci[n-1]
```

- umesto ovako:

```
mesecSkr = meseci[pos:pos+3]
```

Redni broj meseca u godini → troslovna oznaka 6

```
# month2.py
# Štampa skraćeno ime meseca za dati redni broj

# ovde čuvamo nazive
meseci = ["Jan", "Feb", "Mar", "Apr", "Maj", "Jun",
    "Jul", "Avg", "Sep", "Okt", "Nov", "Dec"]

n = eval(raw_input("Unesite mesec (1-12): "))

# ispiši rezultat
print ("Skraćenica je", meseci[n-1] + ".")
```

- pošto je indeks lako izračunati, nije nam neophodna pomoćna promenljiva

Redni broj meseca u godini → naziv meseca

- ova verzija programa se lako menja za rad sa punim nazivima

```
meseci = ["Januar", "Februar", "Mart", "April", "Maj", "Jun",
    "Jul", "Avgust", "Septembar", "Oktobar", "Novembar",
    "Decembar"]
```

Menjanje sekvence, stringa i liste

- sekvenca je lista koja se ne može menjati
- string je sekvenca ⇒ ne može se menjati
- lista se može menjati

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = "p"

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    myString[2] = "p"
TypeError: object doesn't support item assignment
```

Kodni rasporedi

- kodni raspored: mapiranje slova na brojeve radi reprezentacije slova u memoriji
- u davna vremena, svaki proizvođač računara je imao sopstveni kodni raspored
- ASCII standard: 7-bitni kodni raspored (127 slova)
- Unicode standard: preko 100.000 različitih slova
- Python radi sa Unicode slovima (default od verzije 3)

Konverzija slovo ↔ broj

- funkcija `ord` vraća numerički kod za dato slovo
- funkcija `chr` vraća slovo za dati numerički kod

```
>>> ord("A")
65
>>> ord("a")
97
>>> chr(97)
'a'
>>> chr(65)
'A'
```

Primer: konverzija string $\leftrightarrow$ niz brojeva

- algoritam je jednostavan:

1 unesi string za konverziju

2 za svaki znak u stringu:

2a ispiši numerički kod znaka

- `for` petlja ide kroz sekvencu, a string je sekvenca:
  `for <ch> in <string>:`

Primer: konverzija string $\leftrightarrow$ niz brojeva $_2$

```
# text2numbers.py
# Konvertuje string u niz brojeva koristeći Unicode
# kodni raspored

print("Konvertuje string u niz brojeva koristeći")
print("Unicode kodni raspored.\n")

# unesi tekst
message = raw_input("Unesite tekst za kodiranje: ")

print("\nOvo su Unicode kodovi:")

# za svaki znak u stringu ispiši njegov kod
for ch in message:
    print(ord(ch),  end=" ")

print()
```

Primer: konverzija niz brojeva $\leftrightarrow$ string

- bilo bi lepo imati i program koji radi inverziju prethodnog
- algoritam za dekoder bi mogao da glasi:

1 unesi niz brojeva za dekodiranje
2 message $\leftarrow$ ""
3 za svaki broj u nizu:
3a konvertuj broj u znak
3b dodaj znak na kraj message
4 ispiši message

Primer: konverzija niz brojeva $\leftrightarrow$ string $_2$

- promenljiva `message` je akumulator
- inicijalno postavljena na prazan string
- u svakom prolazu petlje jedan broj se konvertuje u znak
- i dodaje na kraj akumulatora

Primer: konverzija niz brojeva $\leftrightarrow$ string $_2$

- kako da unesemo sa tastature niz brojeva promenljive dužine?
- učitamo ga kao string, pa podelimo na delove koji predstavljaju pojedinačne brojeve
- novi algoritam:

1 unesi niz brojeva kao string
2 message $\leftarrow$ ""
3 za svaki od podstringova:
3a konvertuj niz znakova koji predstavljaju broj u broj
3b konvertuj taj broj u znak
3c dodaj znak na kraj message
4 ispiši message

Funkcija split

- podeli string na podstringove, u odnosu na neki separator-znak

```
>>> "Hello string methods!".split()
['Hello', 'string', 'methods!']

>>> "32,24,25,57".split(",")
['32', '24', '25', '57']
>>>
```

Funkcija eval

- konverzija string → broj pomoću funkcije `eval`

```
>>> numStr = "500"
>>> eval(numStr)
500
>>> x = eval(raw_input("Enter a number "))
Enter a number 3.14
>>> print x
3.14
>>> type (x)
<type 'float'>
```

Primer: konverzija niz brojeva ↔ string 2

```
# numbers2text.py
# Konvertuje niz Unicode brojeva u string

print("Konvertuje niz Unicode brojeva u string.\n")

# unesi poruku
inString = input("Unesite poruku kao Unicode niz: ")

# za svaki podstring odredi znak i dodaj ga u rezultat
message = ""
for numStr in inString.split(" "):
    # konvertuj podstring u broj
    codeNum = eval(numStr)
    # dodaj znak na kraj rezultata
    message = message + chr(codeNum)

print("\nDekodirana poruka je:", message)
```

Primer: konverzija niz brojeva $\leftrightarrow$ string $_2$

```
$ python text2numbers.py
Konvertuje string u niz brojeva koristeći
Unicode kodni raspored.

Unesite tekst za kodiranje: Pajton je zakon

Ovo su Unicode kodovi:
80 97 106 116 111 110 32 106 101 32 122 97 107 111 110

$ python numbers2text.py
Konvertuje niz Unicode brojeva u string.

Unesite poruku kao Unicode niz: 80 97 106 116 111 110 32 106 101 32 12

Dekodirana poruka je: Pajton je zakon
```

Još funkcija za stringove

- postoji puno funkcija koje operišu nad stringovima
- `s.capitalize()` – kopija `s` sa prvim znakom pretvorenim u veliko slovo
- `s.title()` – kopija `s` sa prvim znakom svake reči pretvorenim u veliko slovo
- `s.center(width)` – centriraj `s` u string dužine `width`
- `s.ljust(width)` – kao `center`, samo je tekst poravnat sa leve strane
- `s.rjust(width)` – kao `center`, samo je tekst poravnat sa desne strane

Još funkcija za stringove 2

- `s.count(sub)` – broj pojavljivanja podstringa `sub` u stringu `s`
- `s.find(sub)` – indeks prvog mesta gde se podstring `sub` pojavljuje u stringu `s`
- `s.rfind(sub)` – kao `find`, samo traži podstring sa desne strane
- `s.join(list)` – konkatenraj stringove iz liste `list` koristeći `s` kao separator

Još funkcija za stringove 3

- `s.lower()` – konvertuje sve u mala slova
- `s.upper()` – konvertuje sve u velika slova
- `s.lstrip()` – ukloni razmake (whitespace) sa početka stringa
- `s.rstrip()` – ukloni razmake (whitespace) sa kraja stringa
- `s.split(sep)` – podeli `s` na delove oko znakova `sep`
- `s.replace(oldsub, newsub)` – zameni sve podstringove `oldsub` u stringu `s` sa `newsub`

Formatiranje stringova

- često moramo da „ulepšamo“ string pre ispisa
- recimo da hoćemo da konvertujemo datum iz oblika "15.10.2011." u oblik "15. oktobar 2011."

1. unesi datum u formatu dd.mm.gggg.
2. podeli string oko tačke u dan, mesec i godinu
3. konvertuj broj meseca u naziv
4. formiraj novi string u obliku dd. mesec gggg.
5. ispiši novi string

Formatiranje stringova 2

- prva dva reda su jednostavna:

```
dateStr = input('Unesite datum (dd.mm.gggg.): ')
dayStr, monthStr, yearStr, x = dateStr.split('.')
```

- datum se učita kao string, pa se podeli oko tačke na 4 dela
- prva tri dela upišemo u bitne promenljive

Formatiranje stringova 3

- sledeći korak: konvertovati `monthStr` u broj
- možemo da koristimo funkciju `int`
  `int("05") = 5`

Formatiranje stringova 4

- `eval` neće raditi za brojeve sa vodećim nulama (vodeća nula je nekad označavala oktalni broj)

```
>>> int("05")
5
>>> eval("05")
 Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
eval("05")
File "<string>", line 1
05
     ^
SyntaxError: invalid token
```

Formatiranje stringova 5

```
months = ["januar", "februar", ..., "decembar"]
monthStr = months[int(monthStr) - 1]
```

- indeks u listi počinje od nula, zato oduzimamo 1 od broja meseca
- ostaje još konkatenacija rezultujućeg stringa

Formatiranje stringova 6

```
print ("Konvertovani datum je:", dayStr+".",
    monthStr, yearStr+".")
```

- tačka je dodata na dayStr i yearStr konkatenacijom

```
Unesite datum (dd.mmm.gggg.): 15.10.2011.
Konvertovani datum je: 15. oktobar 2011.
```

Broj → string

- za konverziju broj → string možemo da koristimo funkciju `str`

```
>>> str(500)
'500'
>>> value = 3.14
>>> str(value)
'3.14'
>>> print("Vednost je", str(value) + ".")
Vrednost je 3.14.
```

Broj → string i konkatenacija

- ako je neka vrednost string, možemo konkatenirati tačku na njen kraj
- ako je ta vrednost int, šta onda?

```
>>> value = 3.14
>>> print("Vrednost je", value + ".")
Vrednost je

Traceback (most recent call last):
  File "<pyshell#10>", line 1, in -toplevel-
    print "The value is", value + "."
TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

Operacije za konverziju

funkcija	**značenje**
`float(expr)`	konvertuj `expr` u `float` vrednost
`int(expr)`	konvertuj `expr` u `int` vrednost
`str(expr)`	vrati string reprezentaciju od `expr`
`eval(str)`	izračunaj `str` kao Python izraz

Formatiranje stringova

```
Change Counter

Please enter the count of each coin type.
Quarters: 6
Dimes: 0
Nickels: 0
Pennies: 0

The total value of your change is 1.5
```

- bilo bi lepše da na kraju piše `$1.50`

Formatiranje stringova 2

- možemo da preradimo ispis rezultata ovako:

```
print("Your change is ${0:0.2f}".format(total))
```

- sada ćemo dobiti nešto ovako:

```
The total value of your change is $1.50
```

Funkcija `format`

- `<šablon>.format(<vrednosti>)`
- vitičaste zagrade {...} označavaju mesta u koja se upisuju vrednosti
- svako mesto ima format

Funkcija `format` 2

```
print("Your change is ${0:0.2f}".format(total))
```

- šablon sadrži jedno mesto sa opisom `0:0.2f`
- opis ima format `<index>:<format-spec>`
- `index` je redni broj parametra koga treba upisati
- `format-spec` ima oblik: `<width>.<precision><type>`
- `width` je broj znakova koju vrednost treba da zauzima; 0 znači da zauzme koliko je potrebno
- `precision` je broj decimala
- `f` znači broj u „fiksnom zarezu“

Funkcija format 3

```
>>> "Hello {0} {1}, you may have won ${2}" .format("Mr.", "Smith", 10000)
'Hello Mr. Smith, you may have won $10000'

>>> 'This int, {0:5}, was placed in a field of width 5'.format(7)
'This int,     7, was placed in a field of width 5'

>>> 'This int, {0:10}, was placed in a field of witdh 10'.format(10)
'This int,         10, was placed in a field of witdh 10'

>>> 'This float, {0:10.5}, has width 10 and precision 5.'.format(3.1415926)
'This float,    3.1416, has width 10 and precision 5.'

>>> 'This float, {0:10.5f}, is fixed at 5 decimal places.'.format(3.1415926)
'This float,   3.14159, has width 0 and precision 5.'
```

Funkcija `format` 4

- ako je `width` veči nego što je potrebno
  - podrazumevano poravnanje:
  - numerički podaci se poravnavaju po desnoj strani
  - stringovi se poravnavaju po levoj strani
- poravnanje se može eksplicitno navesti pre `width`

```
>>> "levo poravnato: {0:<5}.format("Hi!")
'levo poravnato: Hi!  '
>>> "desno poravnato: {0:>5}.format("Hi!")
'desno poravnato:   Hi!'
>>> "centrirano: {0:^5}".format("Hi!")
'centrirano:  Hi! '
```

Novčani iznosi

- brojevi u pokretnom zarezu su nezgodni za čuvanje novčanih iznosa
- možemo čuvati novac u parama/centima kao `int` i konvertovati u dinare/evre i pare/cente prilikom ispisa
- ako je `total` vrednost u evro centima:

```
euros = total // 100
cents = total % 100
```

- centi se ispisuju sa širinom 0>2 da bismo dobili desno poravnat broj sa vodećim nulama
- tako se 5 centi ispisuje kao 05

Novčani iznosi 2

```
# change2.py

print ("Change Counter\n")
print ("Please enter the count of each coin type.")
quarters = eval(raw_input("Quarters: "))
dimes = eval(raw_input("Dimes: "))
nickels = eval(raw_input("Nickels: "))
pennies = eval(raw_input("Pennies: "))
total = quarters * 25 + dimes * 10 + nickels * 5 + pennies

print ("The total value of your change is ${0}.{1:0>2}"
    .format(total//100, total%100))
```

Novčani iznosi 3

```
Change Counter

Please enter the count of
each coin type.
Quarters: 0
Dimes: 0
Nickels: 0
Pennies: 1

The total value of your change
is $0.01
```

```
Change Counter

Please enter the count of
each coin type.
Quarters: 12
Dimes: 1
Nickels: 0
Pennies: 4

The total value of your change
is $3.14
```

Fajlovi

- fajl (datoteka) je sekvenca podataka koji se čuva na sekundarnoj memoriji (disku)
- fajlovi mogu da sadrže bilo koji tip podataka
- najlakše je raditi sa tekstom
- tekstualni fajl tipično sadrži više redova teksta
- Python koristi line feed znak (`\n`, str(10)) da označi kraj reda

Višelinijski stringovi

```
Hello
World

Goodbye 32
```

- kada se ovaj string snimi u fajl, on izgleda ovako:

```
Hello\nWorld\n\nGoodbye 32\n
```

Višelinijski stringovi

- ovo radi na isti način kao ugrađivanje `\n` u `print` naredbe
- ovi znakovi utiču samo na ispis; ne utiču na izračunavanje izraza

Obrađivanje fajlova

- proces otvaranja fajla uključuje povezivanje fajla na disku sa objektom u memoriji
- nakon otvaranja možemo menjati fajl putem tog objekta
  - čitanje iz fajla
  - pisanje u fajl
- nakon obavljenog posla, fajl je potrebno zatvoriti

Zatvaranje fajla

- zatvaranje fajla će pokrenuti sve operacije nad fajlom koje do tada nisu izvršene
  - snimanje podataka
  - pražnjenje bafera, itd.
- ako zaboravimo da zatvorimo fajl, moguće je da će doći do gubitka nekih podataka!

Čitanje fajla

- učitavanje fajla u tekst-editor
  - otvaranje fajla
  - učitavanje sadržaja fajla u memoriju
  - zatvaranje fajla
  - izmene nad sadržajem se obavljaju u memoriji, fajl se ne menja!

Snimanje fajla

- snimanje fajla iz memorije na disk
  - originalni fajl je otvara u režimu koji omogućuje upis podataka – to će obrisati stari sadržaj!
  - operacije za upis podataka će iskopirati sadržaj iz memorije na disk
  - zatvaranje fajla (obavezno!)

Tekstualni fajlovi i Python

- otvaranje fajla (kreiranje objekta u memoriji koji predstavlja fajl)
  `<filevar> = open(<name>, <mode>)`
- `name` je string koji predstavlja ime fajla na disku
- `mode` je `"r"` ili `"w"` zavisno od toga da li čitamo ili pišemo u fajl

```
infile = open("numbers.dat", "r")
outfile = open("numbers2.dat", "w")
```

Operacije nad fajlom

- `<file>.read()` – vraća preostali sadržaj fajla kao jedan (veliki!) višelinijski string `<filevar> = open(<name>, <mode>)`
- `<file>.readline()` – vraća narednu liniju teksta iz fajla uključujući i `\n`
- `<file>.readlines()` – vraća listu preostalih linija teksta iz fajla; elementi liste su redovi teksta koji se završavaju sa `\n`

Primer operacija nad fajlom

```
# printfile.py

fname = input("Enter filename: ")
infile = open(fname,'r')
data = infile.read()
print(data)
infile.close()
```

1. korisnik unosi ime fajla
2. fajl se otvara za čitanje
3. ceo fajl se učitava i smešta u string `data`
4. string `data` se ispisuje

Čitanje celog fajla

- `readline` se može upotrebiti za čitanje sledećeg reda iz fajla, uključujući i `\n`

```
infile = open(someFile, "r")
for i in range(5):
    line = infile.readline()
    print(line[:-1])
infile.close()
```

- čita prvih 5 redova iz fajla
- uklanjanje `\n` se obavlja pomoću `split`

Čitanje celog fajla 2

- drugi način je da pročitamo sve redove pomoću `readlines`
- i da u petlji prođemo kroz listu stringova

```
infile = open(someFile, "r")
for line in infile.readlines():
    # uradi nešto sa tekućim redom
infile.close()
```

Čitanje celog fajla $_3$

- Python tretira fajl-objekat kao sekvencu redova!

```
infile = open(someFile, "r")
for line in infile:
    # uradi nešto sa tekućim redom
infile.close()
```

Pisanje u fajl

- otvaranje fajla za upis priprema fajl za prijem podataka
- ako se otvori postojeći fajl, njegov sadržaj će biti obrisan
- ako fajl prethodno ne postoji, biće kreiran novi fajl

```
outfile = open(someFile, "w")
print(sadržaj, file=outfile)
```

Batch režim rada

- batch režim rada: nema neposrednog unosa/ispisa podataka
- ulaz i izlaz se odvija preko fajlova
- primer: kreiranje korisničkih imena (username) na osnovu imena i prezimena

Batch režim rada 2

```
infileName = input("Input file? ")
outfileName = input("Output file? ")

infile = open(infileName, 'r')
outfile = open(outfileName, 'w')

for line in infile:
    first, last = line.split()
    uname = (first[0]+last[:7]).lower()
    print(uname, file=outfile)

infile.close()
outfile.close()

print("Written to", outfileName)
```

Batch režim rada 3

- programi često imaju otvoreno više fajlova
- funkcija `lower` je zgodna za konverziju imena fajlova u mala slova