

OSNOVE PROGRAMIRANJA PYTHON

Profesor
Milan Paroški

Novi Sad,
2024/2025



Novembarski ispitni rok za starije studente:

smer	Predmet	Datum	Vreme početka ispita	Sala
SII	Osnove programiranja	27.11.2024.	16:00	E002
IT	Osnove programiranja	27.11.2024.	16:00	E002

Za studente koji sada imaju pravo da slušaju predmet - K2:

IT - mTutor 3 min/30 (15+15 ili 12) pitanja/30 ili 27 bodova

SII – praktični deo : 40 minuta 2 zadatka/15 bodova

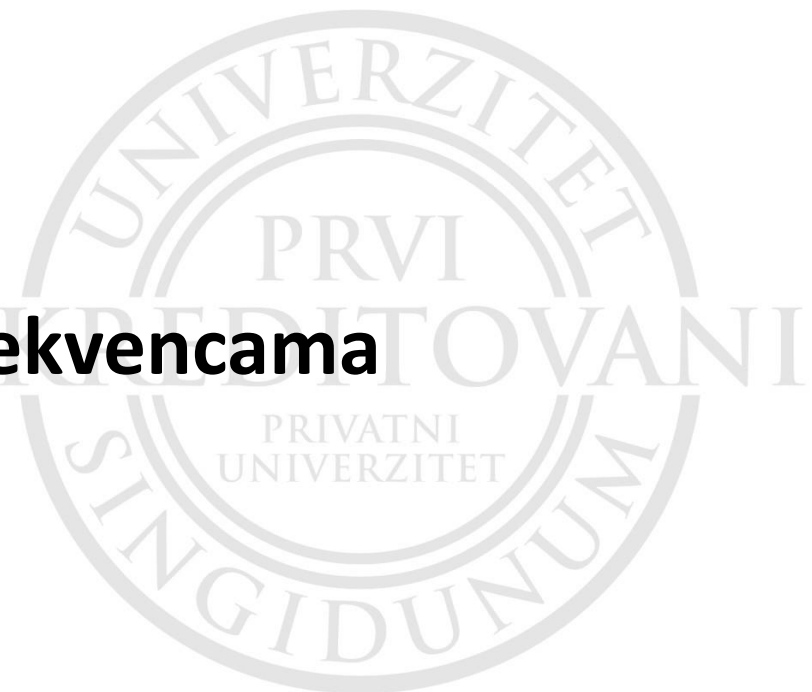
- teorijski deo : 15 minuta/15 ili 12 pitanja/15 ili 12 bodova

Teorija : poglavlja 6,7,8,9

II kolokvijum – Osnovne strukture podataka u jeziku Python

Sadržaj:

1. Uvod
2. Stringovi
3. Liste
4. N-torke
5. Osnovne operacije nad sekvencama
6. Primeri programa



1. Uvod

1.1. Strukture podataka u memoriji

1.2. Osnovne strukture podataka u jeziku Python



1.1. Strukture podataka u memoriji

- **Struktura podataka je ?**

kolekcija više delova podataka strukturiranih na takav način da im se može efikasno pristupati

Koje su strukturirani podaci?

U bazama podataka, excelu itd

Koje su nestrukturirani podaci?

U mejlovima, doc dokumentima itd

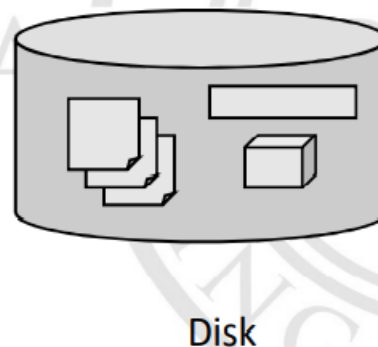
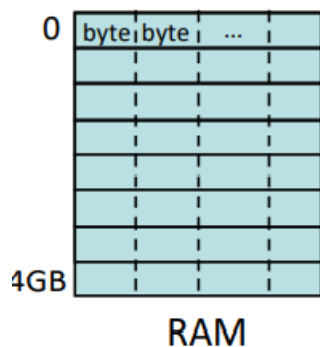








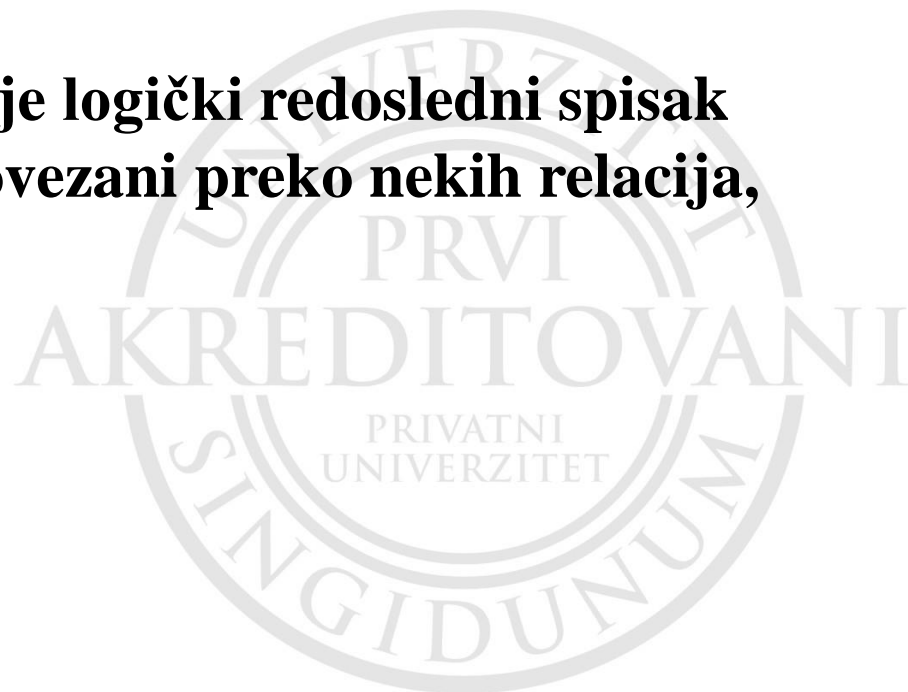
- **Zbog različitog načina pristupa podacima, razlikuju se strukture podataka u ?:**
 - **memoriji s direktnim pristupom (operativna memorija, RAM) i**
 - **na drugim medijima, npr. na disku**



U ovoj temi se razmatraju samo strukture podataka u memoriji

1.2. Osnovne strukture podataka u jeziku Python

- Posmatra se najjednostavnija organizacija podataka u obliku linearne **sekvence** ili niza
- U filmu i u pozorištu pojam sekvence : najmanja filmska zaokružena celina, scena ili epizoda.
- **Prema definiciji sekvenca ili niz je logički redosledni spisak elemenata koji su međusobno povezani preko nekih relacija, najčešće matematičkih.**



- U jeziku Python, sekvence su linearno uređeni nizovi podataka čijim elementima se pristupa pomoću ?
 - numeričke vrednosti ili
 - indeksa
- Sekvence (nizovi) su:
 1. stringovi - nizovi karaktera
`a='Novi Sad'`
 2. liste - nizovi elemenata bilo kog tipa
`a = [10, 20, 30, 40]`
 3. n-torke - nizovi elemenata bilo kog tipa koji se ne mogu menjati
`a = (1,2,3,4)`
- Indeksi elemenata sekvence računaju se unapred od početne pozicije 0 ili unazad, od poslednje pozicije niza koja je n-1

2. Stringovi

2.1. Sintaksa

2.2. Uvodni primeri

2.3. Osnovne funkcije za rad sa stringovim



2.1. Sintaksa

- String je sekvenca znakova u navodnicima ('...' ili "...") , npr.

`print('Novi Sad')` ili

`print("Novi Sad")`

Rezultat je: Novi Sad

`print('Novi Sad')?`

- Znak `\` Šta znači?

Jedna logička linija može se produžiti u više fizičkih pomoću znake nastavka `"\"`, npr.

`print("\nPrimer \n2")?`

Rezultat:

===

Primer

2



- Elementi stringa se indeksiraju od nule:

'	P	y	t	h	o	n	'
	0	1	2	3	4	5	
	-6	-5	-4	-3	-2	-1	



Uvodni primeri

M
o
n
t
y

P
y
t
h
o
n
>>>

Pristup elementima stringa u petlji i ispis svih znakova teksta, po jedan u redu, može se izvršiti jednom naredbom ponavljanja
Koja blok naredbi izvršava za svaki element niza:

```
for ch in 'Monty Python':
```

```
    print(ch)
```

Šta će se ispisati?

Objašnjenje

1. `for ch in 'Monty Python':`

- `for` petlja iterira kroz svaki karakter (slovo) u nizu 'Monty Python'. Svaki karakter se privremeno smešta u promenljivu `ch` tokom jedne iteracije petlje.

2. `print(ch):`

- Funkcija `print()` ispisuje vrednost `ch`, tj. trenutni karakter. Pošto se svaka `print()` komanda u Pythonu po defaultu završava prelaskom u novi red, svaki karakter će biti ispisan u novom redu.

Ponekad je pogodno u petlji umesto elemenata sekvence koristiti njihove pozicije ili indekse, npr.

```
s = 'Monty Python'
for i in range(len(s)):
    print(s[i])
```

Šta će se ispisati?

A šta će ispisati ako je poslednji red promenjen :

```
s = 'Monty Python'
for i in range(len(s)):
    print(len(s),s[i])
```

=====

M
o
n
t
y

P
y
t
h
o
n

>>>

|


```
s = "python"  
s = s[0]  
print(s)
```

Rezultat: Python

```
s = "Pithon"  
s = s[3:]  
print(s)
```

Rezultat: P

```
s = "Pithon"  
s = s[0:]  
print(s)
```

Rezultat: hon



- Elementi strukture string se ne mogu menjati:

```
s = "Pithon"
```

```
s[1] = "y" #Šta je rezultat?
```

Traceback (most recent call last):

File "", line 1, in s[1] = "y"

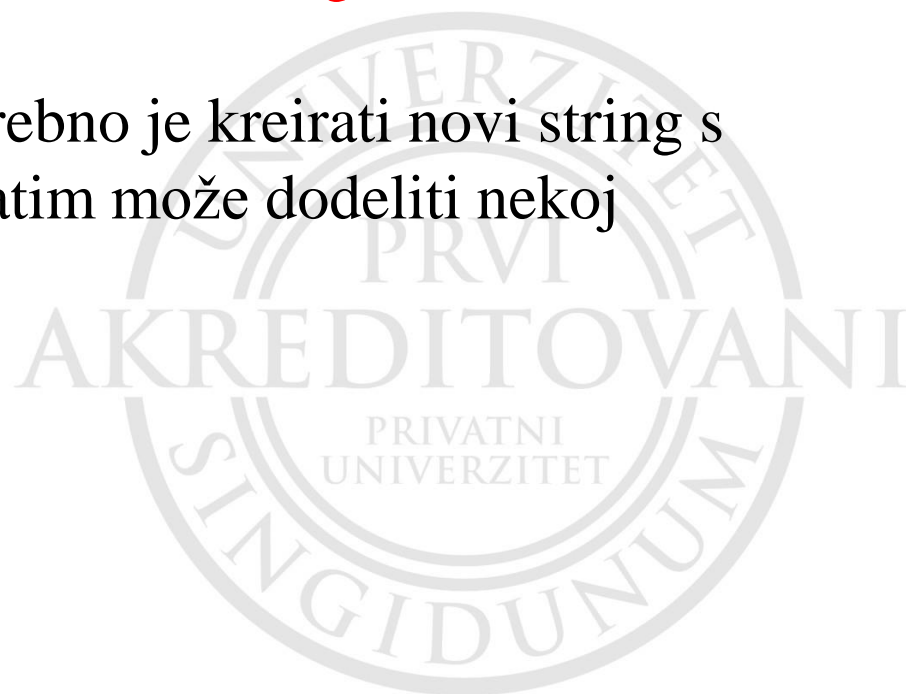
TypeError: 'str' object does not support item assignment

- Za promenu sadržaja stringa potrebno je kreirati novi string s izmenjenim sadržajem, koji se zatim može dodeliti nekoj promenljivoj:

```
s = "Pithon"
```

```
s = s[0] + "y" + s[2:]
```

```
print(s)
```



Osnovne funkcije za rad sa stringovima

1. **len(s)** -?

dužina stringa s

2. **str(x)** ?

konverzija drugih tipova podataka x u tip string

Šta je rezultat:

s = 2

print(s+3)

s1 = str(s)

print(s1+3)

5

Traceback (most recent call last):



3. `s[i]` ?

pristup elementu stringa `i`

4. `s[i:j]` ?

izdvajanje dela stringa od pozicije `i` do pozicije `j`

```
s = 'Python'
```

```
s = s[2:5]
```

```
print(s)
```

tho

5. `+` ?

spajanje (konkatenacija) stringova

Konkatenacija

`"Novi"+" Sad"`



6. Funkcije za konverziju zapisa pojedinačnih znakova :

ord(c) - vraća kod znaka u ASCII tabeli kodova

chr(i) - vraća znak s kodom u ASCII tabeli kodova

```
print(ord('s'))
```

115

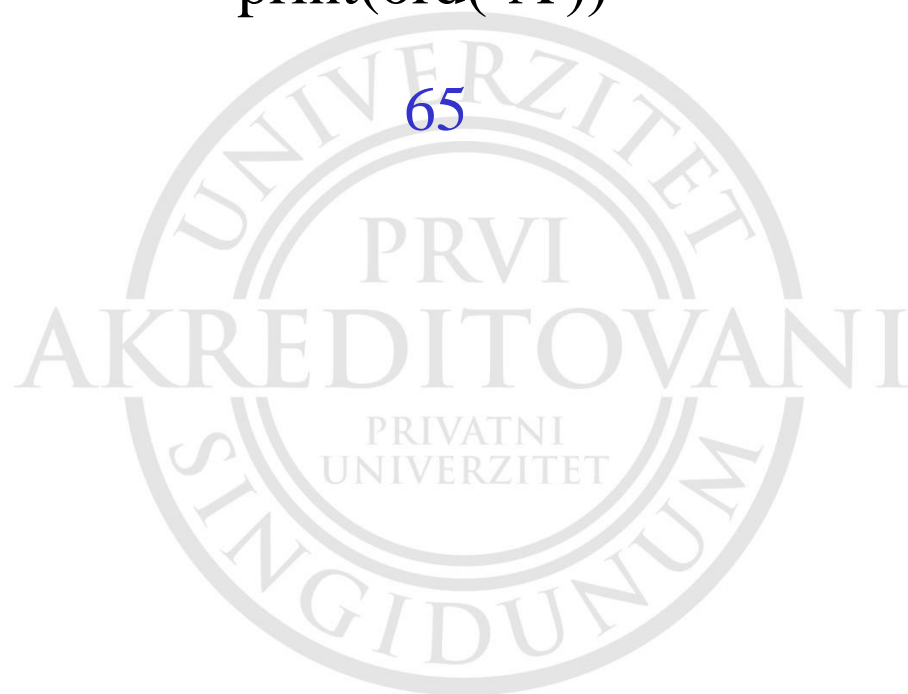
```
# print the character
```

```
print(chr(115))
```

s

```
print(ord('A'))
```

65



ASCII()

Američki standardni kod za razmenu informacija (ASCII) je standard za engleske znakove, brojeve, znakove interpunkcije i druge posebne znakove.

Standardni skup od 128 znakova kodira se sa koliko bita?

7

dok je osmi bit u komunikacijama služio za ?
kontrolu pariteta.

Kontrolni znakovi u ASCII formatu nisu elementi za štampanje, već neke komande ili pokazivača.

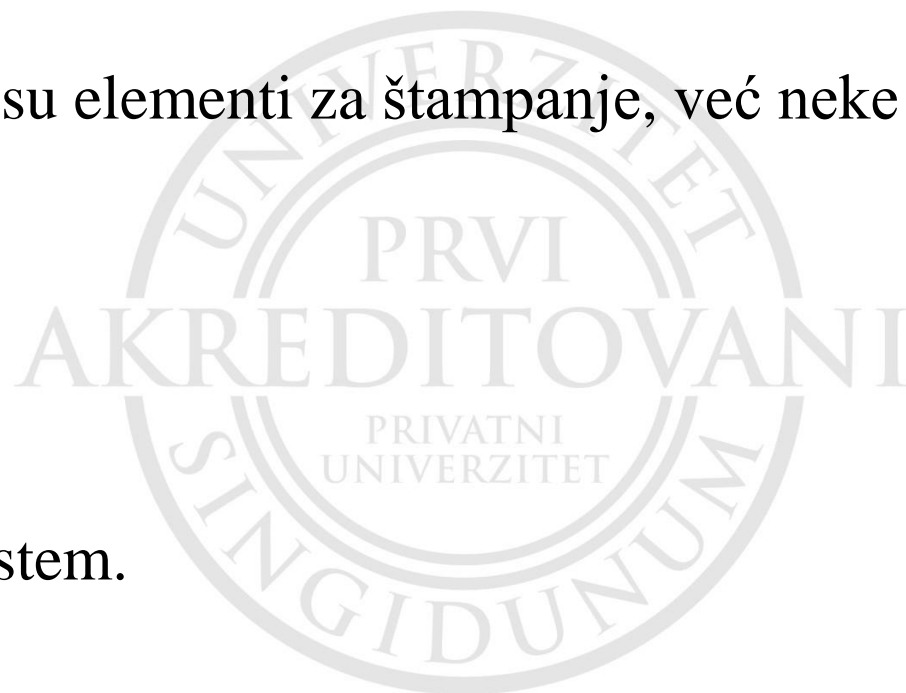
Primeri:

\ e („escape“),

\? („Delete“) i

\ 0 („null character“).

Ovo su znakovi koje čita operativni sistem.



ASCII
tabela
znakova
ASCII
Standard
binarnog
zapisa
različitih
znakova koji
se koriste u
računarstvu
i
telekomunikacijama

Dec	Oct	Hex	Znak	Dec	Oct	Hex	Znak	Dec	Oct	Hex	Znak	Dec	Oct	Hex	Znak
0	0	00	NUL (null)	32	40	20	(space)	64	100	40	@	96	140	60	`
1	1	01	SOH (start of header)	33	41	21	!	65	101	41	A	97	141	61	a
2	2	02	STX (start of text)	34	42	22	"	66	102	42	B	98	142	62	b
3	3	03	ETX (end of text)	35	43	23	#	67	103	43	C	99	143	63	c
4	4	04	EOT (end of transmission)	36	44	24	\$	68	104	44	D	100	144	64	d
5	5	05	ENQ (enquiry)	37	45	25	%	69	105	45	E	101	145	65	e
6	6	06	ACK (acknowledge)	38	46	26	&	70	106	46	F	102	146	66	f
7	7	07	BEL (bell)	39	47	27	'	71	107	47	G	103	147	67	g
8	10	08	BS (backspace)	40	50	28	(72	110	48	H	104	150	68	h
9	11	09	HT (horizontal tab)	41	51	29)	73	111	49	I	105	151	69	i
10	12	0a	LF (line feed - new line)	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	0b	VT (vertical tab)	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	0c	FF (form feed - new page)	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	0d	CR (carriage return)	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	0e	SO (shift out)	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	0f	SI (shift in)	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	DLE (data link escape)	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	DC1 (device control 1)	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	DC2 (device control 2)	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	DC3 (device control 3)	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	DC4 (device control 4)	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	NAK (negative acknowledge)	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	SYN (synchronous idle)	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	ETB (end of transmission block)	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	CAN (cancel)	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	EM (end of medium)	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	SUB (substitute)	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	ESC (escape)	59	73	3b	;	91	133	5b	[123	173	7b	{
28	34	1c	FS (file separator)	60	74	3c	<	92	134	5c	\	124	174	7c	
29	35	1d	GS (group separator)	61	75	3d	=	93	135	5d]	125	175	7d	}
30	36	1e	RS (record separator)	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	US (unit separator)	63	77	3f	?	95	137	5f		127	177	7f	DEL (delete)

Vidljivi deo (printable characters)

Šta znači ASCII?

American Standard Code for Information Interchange

Proizvodjači računara su ASCII standard usvojili za prikaz teksta u memoriji računara, koristeći pri tome svih 8 bita za prikaz ukupno 256 znakova.

Uključena su i slova odredjenih nacionalnih pisama...

Problem:šta sa ostalim nacionalnim pismima?

Razvijen je Unicode :koji može da prikaže znakove nacionalnih pisama

UNICODE 8: $2^8 = 256$ karaktera

Koliko je to bajtova?

1

UNICODE 16: $2^{16} = 65536$ karaktera.

Koliko je to bajtova?

2



- Sada se javlja problem prostora za Unicode poruku na medijumu koji se koristi.
- Ako je reč o nekom dokumentu na disku, on će da zauzima duplo više prostora nego konvencionalan dokument jer će se svaki karakter zapisivati sa dva bajta umesto samo sa jednim.
- Ako je reč o prenosu podataka preko računarske mreže, biće potrebno preneti duplo više podataka, pa će samim tim i prenos da traje duplo više (odnosno da košta duplo više).
- Postavlja se pitanje da li je to suviše velika cena za univerzalno pismo i da li postoji neki način da se taj problem prevaziđe i izbegne.

```
normalText = 'Python is interesting'
```

```
print(ascii(normalText))
```

Rezultat: 'Python is interesting'

```
normalText = 'Пајтон је ОК'
```

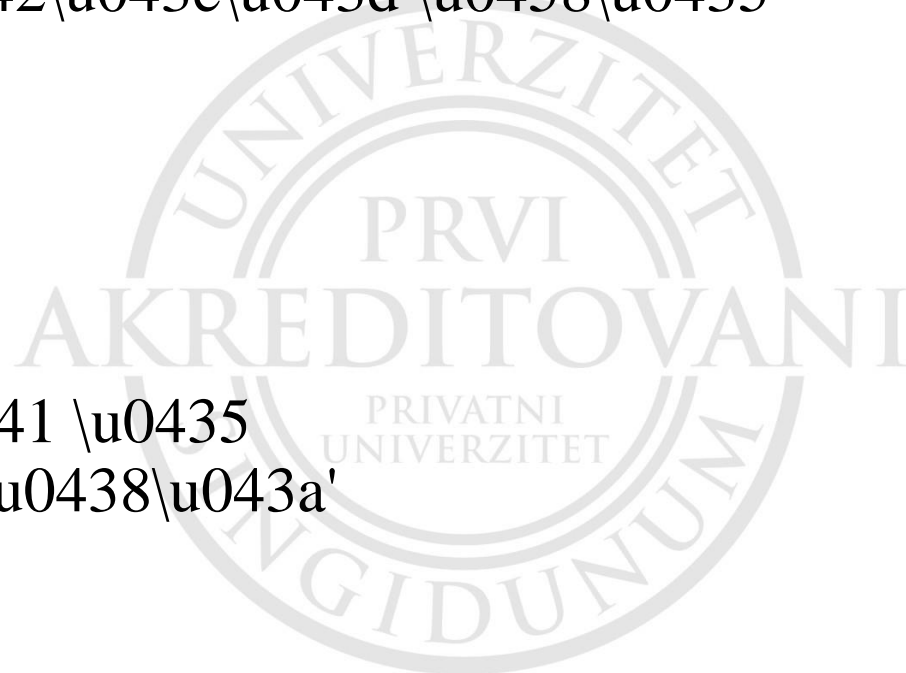
```
print(ascii(normalText))
```

Rezultat: '\u041f\u0430\u0458\u0442\u043e\u043d \u0458\u0435 \u041e\u041a'
\u0415\u0410\u0410'

```
normalText = 'Днес е вторник'
```

```
print(ascii(normalText))
```

Rezultat: '\u0414\u043d\u0435\u0441 \u0435 \u0432\u0442\u043e\u0440\u043d\u0438\u043a'
\u0432\u043e\u0440\u043d\u0438\u043a \u0435 \u0432\u0442\u043e\u0440\u043d\u0438\u043a'



3. LISTE

- 1. Liste u jeziku Python**
- 2. Iteracija nad listama**
- 3. Linearno pretraživanje liste**
- 4. Dodela vrednosti i kopiranje lista**
- 5. Lista kao struktura stek**
- 6. Skraćeno generisanje lista**
- 7. Ugnježdene liste**

Liste su sekvence bilo kakvih vrednosti, uključujući i rezultate izraza.

Najjednostavnije se definiše kao lista vrednosti u uglastim zagradama, npr. : numeričke, nenumeričke i različite:

```
a = [10, 20, 30, 40]  
print(a)
```

Rezultat:
[10, 20, 30, 40]

```
b = ["Војводина", "Спартак",  
     "Пролетер"]  
print(b)
```

Rezultat:
['Војводина', 'Спартак', 'Пролетер']

```
b = ["Војводина", 1, "Спартак", "Пролетер", 2.0]  
print(b)
```

Rezultat: ['Војводина', 1, 'Спартак', 'Пролетер', 2.0]

Operacije nad listama

U jeziku Python predviđene su sledeće operacije nad strukturom liste :

- Kako se dodaje novi element na kraj liste ?

- **append(x)**

```
b=["Војводина",1,"Спартак", "Пролетер",2.0]
```

```
b.append("Срем")
```

```
print(b)
```

`['Војводина', 1, 'Спартак', 'Пролетер', 2.0, 'Срем']`

```
b = ["Војводина",1,"Спартак", "Пролетер",2.0]
```

```
print(len(b))
```

`5`

```
b = ["Војводина",1,"Спартак", "Пролетер",2.0]
```

```
b[len(b):] = ["Срем"]
```

```
print(b)
```

Umesto `b[len(b):]` može: `b[5:] = ["Срем"]`

`['Војводина', 1, 'Спартак', 'Пролетер', 2.0, 'Срем']`

- Proširuje listu dodavanjem svih elemenata zadane liste L na kraj liste ?

extend(L)

```
b = ["Војводина",1,"Спартак", "Пролетер",2.0]
```

```
b.extend("Срем")
```

```
print(b)
```

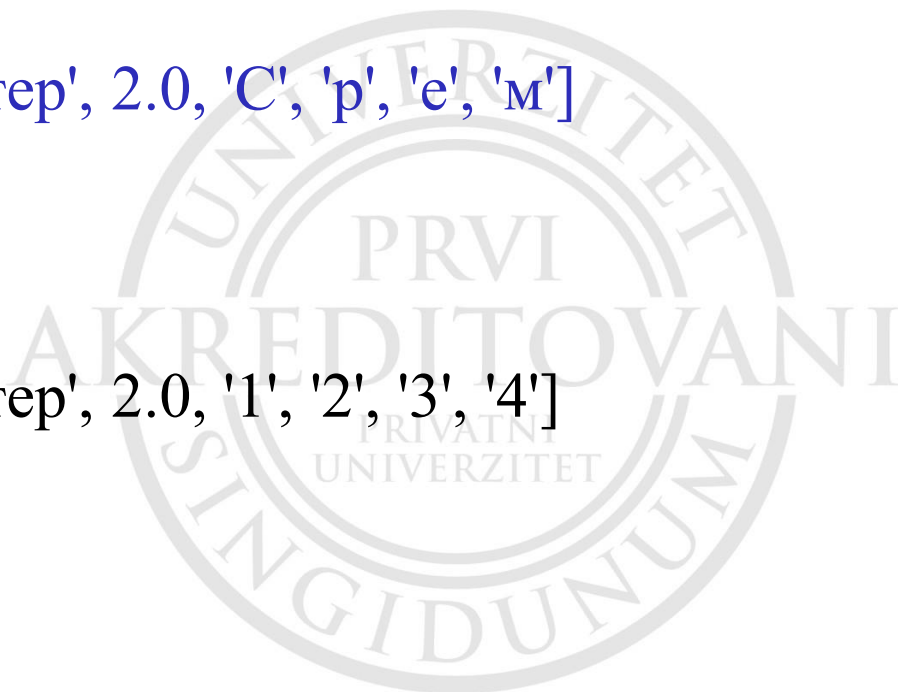
Rezultat?:

```
['Војводина', 1, 'Спартак', 'Пролетер', 2.0, 'С', 'р', 'е', 'м']
```

Kada se stavi : b.extend("1234")

Dobije se ?:

```
['Војводина', 1, 'Спартак', 'Пролетер', 2.0, '1', '2', '3', '4']
```



- Umeće novi element x u listu od zadane pozicije i. Argument i je indeks elementa ispred kog treba umetnuti element x?

insert(i,x)

```
b = ["Војводина",1,"Спартак", "Пролетер",2.0]
```

```
b.insert(0,"Срем")
```

```
print(b)
```

Rezultat: ['Срем', 'Војводина', 1, 'Спартак', 'Пролетер', 2.0]

```
b = ["Војводина",1,"Спартак", "Пролетер",2.0]
```

```
b.insert(3,"Срем")
```

```
print(b)
```

Rezultat: ['Војводина', 1, 'Спартак', 'Срем', 'Пролетер', 2.0]

Gde bi postavio 'Срем' sa naredbom: `b.insert(8,"Срем")`?

a gde sa 18 ili sa -18 ?

- Uklanja prvi element liste čija je vrednost jednaka x?
- **remove(x)**

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]  
b.remove(1)  
print(b)
```

Rezultat:['Војводина', 'Спартак', 1, 'Пролетер', 2.0]

a sa 11?

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]  
b.remove(11)  
print(b)
```

Ukoliko u listi ne postoji takav element, dobija se poruka o grešci

- Uklanja element na zadanoj poziciji i liste , zatim vraća njegovu vrednost?

pop([3])

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]
```

```
b.pop(3)
```

```
print(b)
```

Rezultat: ['Војводина', 1, 'Спартак', 'Пролетер', 2.0]

- Ako indeks nije naveden, **b.pop()** uklanja poslednji elemenat u listi

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]
```

```
b.pop()
```

```
print(b)
```

Rezultat: ['Војводина', 1, 'Спартак', 1, 'Пролетер']

- Uklanja sve elemente liste ?

clear()

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]
```

```
b.clear()
```

```
print(b)
```

Rezultat:[]

- ekvivalentno **del b[:]**

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]
```

```
del b[:]
```

```
print(b)
```

Rezultat:[]



- Враћа индекс liste prvog elementa čija je vrednost x?

index(x)

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]  
print(b.index("Спартак"))
```

- Rezultat: 2

```
b = ["Спартак","Војводина",1,"Спартак",1,"Пролетер",2.0]  
print(b.index("Спартак"))
```

- Rezultat: 0

- Ukoliko takav element ne postoji?

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]  
print(b.index("Војвдина"))
```

Traceback (most recent call last): ValueError: 'Војвдина' is not in list

- Vraća broj pojavljivanja vrednosti x u listi?

count(x)

```
b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]  
print(b.count(1))
```

- Rezultat: 2
- Šta je rezultat?
- b = ["Војводина",1,"Спартак",1,"Пролетер",2.0]
- print(b.count(1))
- Rezultat: 0



SORT

- `sort(key=None, reverse=False)`
- Sortira elemente liste u zadanom redosledu, definisanom argumentima `key` i `reverse`

```
b = ["Војводина", "Спартак", "Пролетер"]
```

```
b.sort()
```

```
print(b)
```

Rezultat: ['Војводина', 'Пролетер', 'Спартак']

Da li može:

```
print(b.sort()) ?
```

None



2.čas



```
brojevi = [33, 21, 5, 15, -3, 0]
```

```
brojevi.sort()
```

```
print(brojevi)
```

```
slova = ['A','a', 'b','E']
```

```
slova.sort()
```

```
print(slova)
```

[-3, 0, 5, 15, 21, 33]

['A', 'E', 'a', 'b']

Zašto je prvo A pa a?

ASCII kod A je 65 a ASCII kod a je 97



REVERSE

- `lista.reverse()`
- Postavlja elemente liste u obrnutom redosledu

```
b = ["Војводина", "Спартак", "Пролетер"]
```

```
b.reverse()
```

```
print(b)
```

- Rezultat: ['Пролетер', 'Спартак', 'Војводина']



COPY

- lista.copy()
- Vraća kopiju cele liste

```
b = ["Војводина", "Спартак", "Пролетер"]
```

```
b.copy()
```

```
print(b)
```

Rezultat: ['Војводина', 'Спартак', 'Пролетер']

- ekvivalentno a[:]

```
b = ["Војводина", "Спартак", "Пролетер"]
```

```
a=b[:]
```

```
print(a)
```

Rezultat: ['Војводина', 'Спартак', 'Пролетер']



3.2. Iteracija nad listama

- Pristup svim elementima liste pojedinačno može se realizovati pomoću petlje, npr. za :

- lista = [10,20,30,40,60,90]

- pristup pomoću petlje for je, po definiciji petlje :

for n in lista:

 print(n)

- pristup pomoću petlje while je programski nešto složeniji ?

n = 0

while n < len(lista):

 print(lista[n])

 n = n + 1

10

20

30

40

60

90

3.3. Linerano pretraživanje liste

- Ponekad je potrebno u nizu vrednosti pronaći određenu vrednost, što se može realizovati pomoću while petlje, npr.

```
lista = [10,30,90,40,50,80]
```

```
podatak = 40
```

```
pronadjen = False
```

```
i = 0
```

```
while i < len(lista) and not pronadjen:
```

```
    if lista[i] == podatak:
```

```
        pronadjen = True
```

```
    else:
```

```
        i = i + 1
```

```
if pronadjen:
```

```
    print("Podatak:", podatak, " je pronadjen")
```

```
else:
```

```
    print("Podatak:", podatak, " nije pronadjen")
```

Podatak: 40 je pronadjen

Podatak: 41 nije pronadjen

Šta bi bio rezultat ?

```
lista = [10,30,90,40,50,80]
```

```
podatak = 40
```

```
pronadjen = False
```

```
i = 0
```

```
while i < len(lista) and not pronadjen:
```

```
    if lista[i] == podatak:
```

```
        pronadjen = True
```

```
        print("Podatak:",podatak, " je pronadjen")
```

```
    else:
```

```
        i = i + 1
```

```
        print("Podatak:",podatak, " nije pronadjen")
```

Podatak: 40 nije pronadjen

Podatak: 40 nije pronadjen

Podatak: 40 nije pronadjen

Podatak: 40 je pronadjen



3.4. Dodela vrednosti i kopiranje lista

- Dodela vrednosti jedne liste drugoj ne stvara kopiju vrednosti, tako da se promena elemenata jedne liste odnosi i na drugu

```
lista1 = [10, 20, 30, 40]
```

```
lista2 = lista1
```

```
lista1[0] = 5
```

```
print(lista1)
```

```
print(lista2)
```

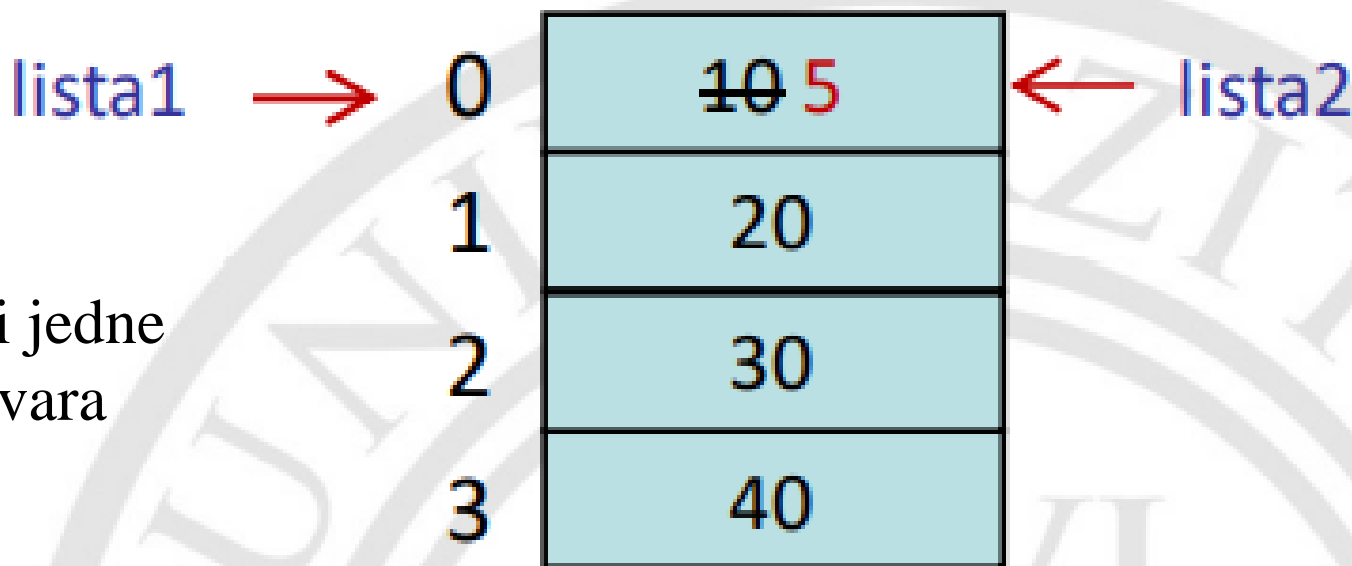
Rezultat?

[5, 20, 30, 40]

[5, 20, 30, 40]



nakon dodele vrednosti promenljivoj
lista2 obe promenljive pokazuju na
istu strukturu podataka



Dodela vrednosti jedne
liste drugoj ne stvara
kopiju vrednosti

- Zasebna kopija strukture podataka koja bi se nezavisno koristila pod nazivom lista2, dobija se naredbom :
- `lista2 = lista1[:]`

```
lista1 = [10, 20, 30, 40]
```

```
lista2 = lista1[:]
```

```
lista1[0] = 5
```

```
print(lista1)
```

```
print(lista2)
```

```
[5, 20, 30, 40]
```

```
[10, 20, 30, 40]
```



3.5. Lista kao struktura stek

Lista se jednostavno koristi kao struktura stek (stack), u kojoj se pristupa samo jednom elementu liste koji se nalazi na kraju (vrh steka, top), a izmene vrše metodima append i pop:

```
stek = [3, 4, 5]
```

```
stek.append(6)
```

```
stek.append(7)
```

```
print(stek)
```

```
print(stek.pop())
```

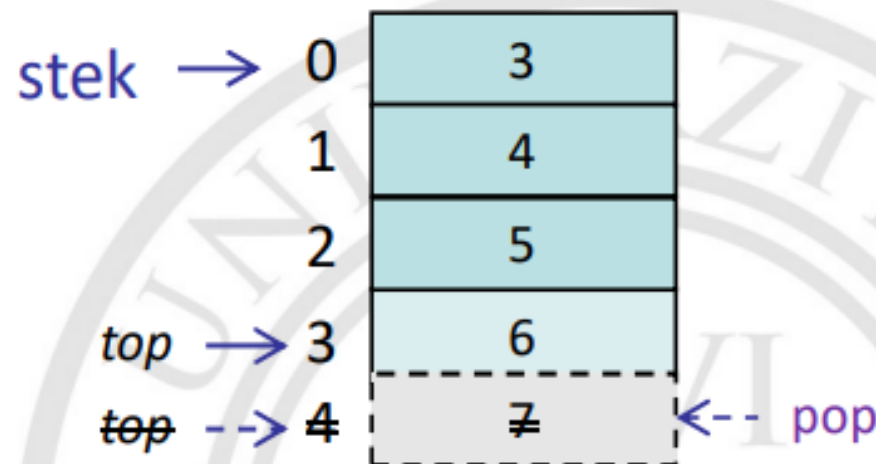
```
print(stek)
```

Rezultat:

```
[3, 4, 5, 6, 7]
```

```
7
```

```
[3, 4, 5, 6]
```



Šta bi bio rezultat ovoga?

```
stek = [3, 4, 5]  
stek.append(6)  
stek.append(7)  
stek.pop(3)
```

```
print(stek)  
print(stek.pop())
```

[3, 4, 5, 7]

7

pop([3])-Uklanja element na
zadanoj poziciji.



3.6. Skraćeno generisanje lista

Skraćeni **zapis izraza koji generiše elemente liste** sastoji se od :

- **[]** - uglastih zagrada u kojima je
- **izraz**
- **for** klauzula
- (iza koje može biti još for ili if klauzula):

[<izraz> for in]

- Rezultat **evaluacije izraza** je lista koja nastaje evaluacijom izraza na svakom koraku izvršavanja petlje for, npr.

```
print([x**3 for x in [1, 2, 3]])
```

šta je rezultat?

Rezultat: [1, 8, 27]

- **Izrazi se mogu koristiti za skraćeni zapis lista u naredbama jezika Python**

Primeri izraza za skraćeno generisanje lista

```
lista = [-1, 1, -2, 2, -3, 3, -4, 4]  
print([x for x in lista if x >= 0])
```

[1, 2, 3, 4]

#Pronalaženje svih pojava samoglasnika u tekstu:

```
samoglasnici = ('a','e','i','o','u')
```

```
tekst = 'Programiranje'
```

```
print([ch for ch in tekst if ch in samoglasnici])
```

['o', 'a', 'i', 'a', 'e']

#Unija elemenata dve liste: x ima vrednosti [1,2,3] a y [3,1,4] kako?

```
print([(x, y) for x in [1,2,3] for y in [3,1,4] if x != y])
```

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

3.7. Ugnježdene liste

- Rekurzivne strukture podataka: **lista kao element liste**:

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
print(x)
```

Rezultat: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

- Jezik Python omogućava pregledniji zapis iste naredbe, gde se liste razmatraju kao redovi matrice:

```
matrica = [[1, 2, 3],  
            [4, 5, 6],  
            [7, 8, 9]]
```

```
print(matrica)
```

Rezultat: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

4. N-torke

- **N-torke su nepromenljive liste** – zadaju se u običnim zagradama, a **elementi se ne mogu menjati**, kao ni struktura (nema odgovarajućih metoda)

```
a = (1,2,3,4)
```

```
print(a)
```

Rezultat: (1, 2, 3, 4)

Ako se doda nov red : `a[1]=1`

Šta se dobije?

Traceback (most recent call last):

File "C:/Users/Milan/Documents/test.py", line 3, in <module>

```
    a[1]=1
```

TypeError: 'tuple' object does not support item assignment

5. Osnovne operacije nad sekvencama

- 1. Pregled osnovnih operacija nad sekvencama u jeziku Python**
- 2. Sekvence kao argumenti funkcije**
- 3. Sekvence kao rezultati funkcija**



5.1 Pregled osnovnih operacija nad sekvencama u jeziku Python

Osnovne operacije nad sekvencama u jeziku Python su

- Dužina sekvence *s* (Length) `len(s)`
- Pristup elementu *i* sekvence (Select) `s[i]`
- Izdvajanje dela sekvence *i..j* (Slice) `s[i:j]`
- Brojanje elemenata sekvence (Count) `s.count()`
- Pronalaženje pozicije elementa *x* (Index) `s.index(x)`
- Provera pripadnosti elementa *v* (Membership) `v in s`
- Spajanje sekvenci *s* i *w* (Concatenation) `s + w`
- Najmanja vrednost u sekvenci (Minimum Value) `min(s)`
- Najveća vrednost u sekvenci (Maximum Value) `max(s)`
- Zbir svih elemenata sekvence (Sum) `sum(s)`

5.2. Sekvence kao argumenti funkcije

- **Sekvence se mogu prenositi kao argumenti funkcije. Pri tome se u stek ne kopira cela struktura, već samo pokazivač na originalnu strukturu podataka**
- **Na taj način se sadržaj liste može menjati u samoj funkciji**
- **Stringovi i n-torke su nepromenljivi objekti, pa im se iz funkcije može pristupati, ali se elementi ne mogu menjati**
- **Struktura koja se zadaje kao eksplicitni niz vrednosti je anonimna, npr.**

`fun([10,20,30], ...)`

- **Takav argument se kreira i nestaje nakon okončanja funkcije, jer ne postoji promenljiva koja bi vrednost pamtila**


```
def fun(s):
```

```
    s[0] = "*" # nova vrednost prvog elementa sekvence s
```

```
    return s
```

```
a = ["a", "b", "c", "d"]
```

Rezultat: ['*', 'b', 'c', 'd']

```
print(fun(a))
```

Šta će se dobiti ako se a zameni sa ovim?

```
def fun(s):
```

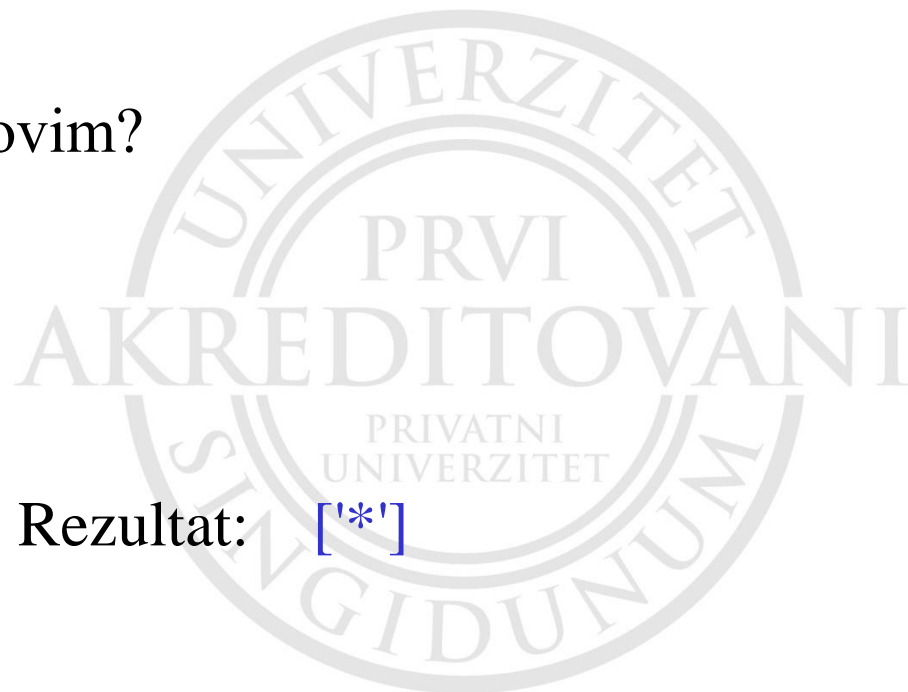
```
    s[0] = "*" # nova vrednost prvog  
    elementa sekvence s
```

```
    return s
```

```
a=["Python"]
```

```
print(fun(a))
```

Rezultat: ['*']



Šta bi se odštamalo?

```
def fun(s):  
    s[0] = "*"   
    return
```

```
a = ["a", "b", "c", "d"]  
print(a)
```

`['a', 'b', 'c', 'd']`

malo pre je bilo: `print(fun(a))`



```
def fun(s):
```

```
    s[0] = "****" # nova vrednost prvog elementa sekvence s
```

```
    return
```

```
a = ["a", "b", "c", "d"]
```

```
print(fun(a))
```

```
print(a)
```

None

['****', 'b', 'c', 'd']

```
def fun(s):
```

```
    s[0] = "****" # nova vrednost
```

```
    prvog elementa sekvence s
```

```
    return a
```

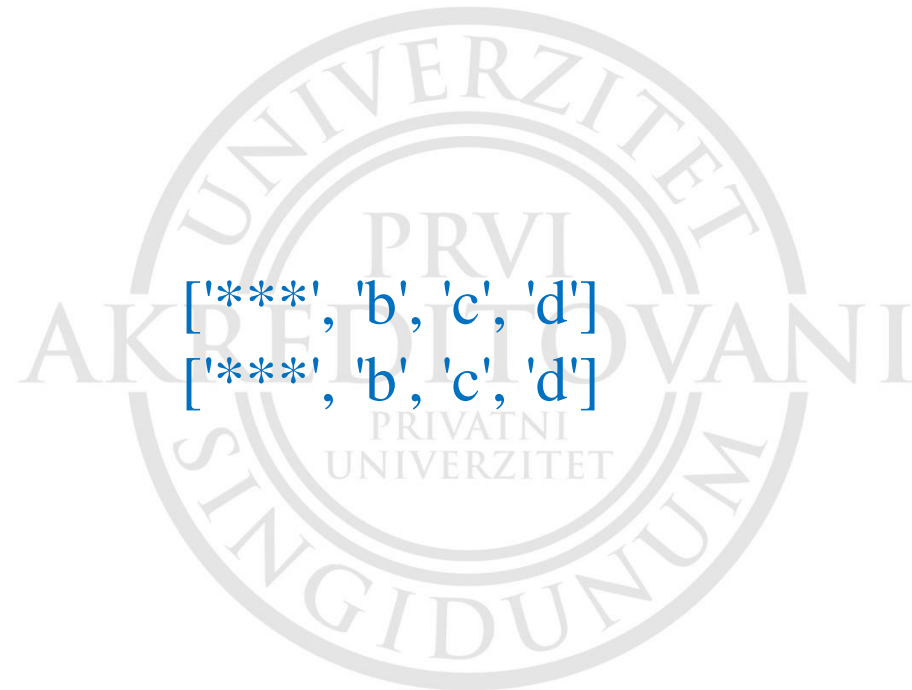
```
a = ["a", "b", "c", "d"]
```

```
print(fun(a))
```

```
print(a)
```

['****', 'b', 'c', 'd']

['****', 'b', 'c', 'd']



3.čas



A šta bi se odšampalo kada bi a bilo:

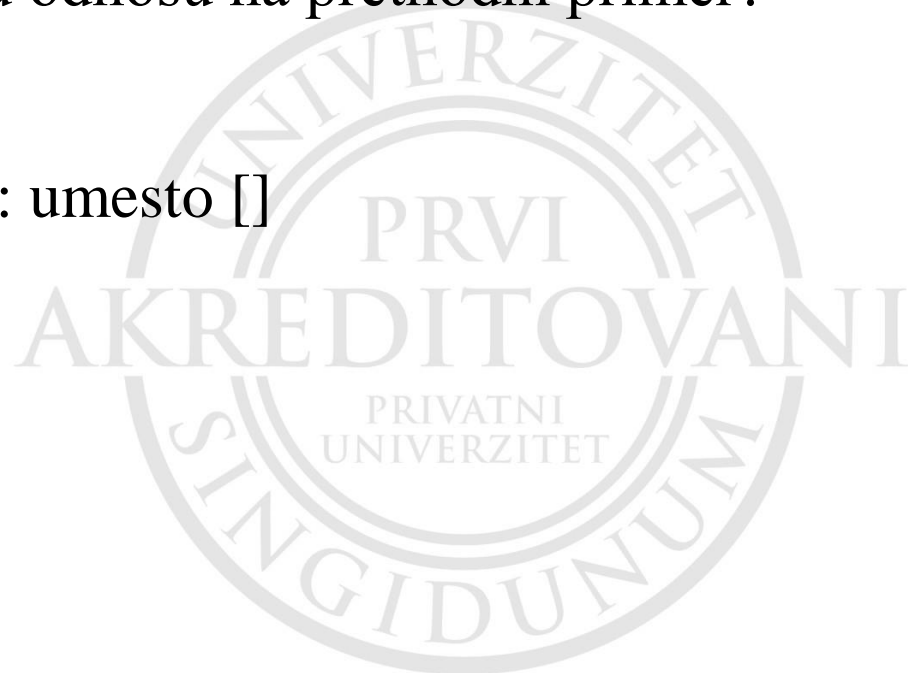
`a = ("a", "b", "c", "d")`

Rezultat: Traceback (most recent call last):

Zašto ne radi? Koje su promene u odnosu na prethodni primer?

Promene su : zagrade kod `a = ()` : umesto `[]`

Sada je a: Ntorka a ne LISTA



```
def fun(s):  
    s[0] = "*" # nova vrednost prvog elementa sekvence s  
    return
```

```
a = "abcd"  
print(fun(a),a)
```

Rezultat: ?

Traceback (most recent call last):

File "C:/Users/Milan/Documents/test.py", line 6, in <module>
 print(fun(a),a)

File "C:/Users/Milan/Documents/test.py", line 2, in fun
 s[0] = "*" # nova vrednost prvog elementa sekvence s

TypeError: 'str' object does not support item assignment

a je string a ne LISTA

```
def fun(broj, lista_brojeva):
```

```
    broj = 100
```

```
    lista_brojeva[0] = 100 # nova vrednost prvog elementa liste
```

```
    return
```

```
broj = 1
```

```
lista = [1, 2, 3, 4]
```

```
print(broj, lista) #Šta se ispiše?
```

Ako se doda još jedan red:

```
fun(broj, lista)
```

```
print(broj, lista)
```

Rezultat: 1 [1, 2, 3, 4]

Ne poziva se uopšte FUN

```
broj = 100 # nova vrednost lokalne promenljive broj
```



Sekvence i podrazumevane vrednosti argumenata

- Sekvence koje se prenose kao argumenti funkcija mogu imati podrazumevajuće vrednosti
- Za razliku od vrednosti osnovnih tipova, dodela definisane podrazumevane vrednosti izostavljenom argumentu vrši se samo kod prvog poziva, npr.

```
def dodaj(x, lista=[]):
```

```
    if x not in lista:
```

```
        lista.append(x)
```

```
    return lista
```

```
a = dodaj(1) # broj se dodaje u default listu []
```

```
print(a)
```

Rezultat: [1]



Šta kada bi dodali još jedan argument?

```
a = dodaj(1,2) # broj se dodaje u default listu []  
print(a)
```

Javi se greška. Zašto?

Očekuje se da je drugi argument LISTA

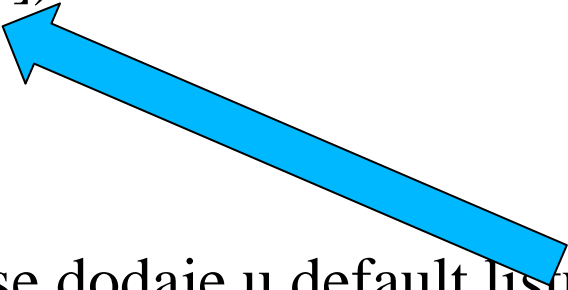
```
def dodaj(x, lista=[]):  
    if x not in lista:  
        lista.append(x)  
    return lista
```

```
a = dodaj(1,[2]) # broj se dodaje u default listu []  
print(a) #šta će sada ispisati?
```

[2, 1]



```
def dodaj(x, lista=[3]):  
    if x not in lista:  
        lista.append(x)  
    return lista  
a = dodaj(1) # broj se dodaje u default listu []  
print(a)
```



Šta će biti rezultat?

[3, 1]



- Ako se dodaju na kraj još 2 reda:

```
def dodaj(x, lista=[3]):
```

```
    if x not in lista:
```

```
        lista.append(x)
```

```
    return lista
```

```
a = dodaj(1) # broj se dodaje u default listu []
```

```
print(a)
```

```
b= dodaj(2) # broj se dodaje u default listu []
```

```
print(b)
```

Rezultat:

[3, 1]

[3, 1, 2]



5.3 Sekvence kao rezultati funkcija

- Funkcija može da kao rezultat vrati više vrednosti
- `return <r1>, <r2>`
- Vrednosti mogu biti osnovnog tipa ili neka od struktura podataka
- Npr. funkcija koja vraća listu u kojoj su elementi u obrnutom redosledu od elemenata zadane liste:

```
def obrnuto(lista):
```

```
    rezultat = [] # kreiranje nove liste
```

```
    for element in lista:
```

```
        print("element=", element)
```

```
        rezultat.insert(0, element) # rezultat je nova lista
```

```
    print("Rezultat=", rezultat)
```

```
    return rezultat
```

```
print(obrnuto([1,2,3,4,5])) # prikaz elemenata nove liste
```

Ključ je u `rezultat.insert(0, ..)` probati 1 ili 2 ili 3 umesto 0

element= 1

Rezultat= [1]

element= 2

Rezultat= [2, 1]

element= 3

Rezultat= [3, 2, 1]

element= 4

Rezultat= [4, 3, 2, 1]

element= 5

Rezultat= [5, 4, 3, 2, 1]

[5, 4, 3, 2, 1]



Kako usporiti izvršavanje programa?

```
import time
```

```
def obrnuto(lista):
```

```
    rezultat = [] # kreiranje nove liste
```

```
    for element in lista:
```

```
        print("element=", element)
```

```
        rezultat.insert(0, element) # rezultat je nova lista
```

```
        print("Rezultat=", rezultat)
```

```
        time.sleep(5)
```

```
    return rezultat
```

```
print(obrnuto([1,2,3,4,5]))
```

