



Softverski algoritmi u sistemima automatskog upravljanja

Osnovne strukture podataka

List

? Šta je lista u Pythonu?

- ▶ Lista je **uređena** kolekcija podataka
- ▶ Može sadržati **različite tipove podataka** (ali najčešće su istog tipa)
- ▶ **Promenjiva je** — može se menjati nakon kreiranja

📌 Primer:

```
niz = [4, 7, 1, 9]
```



Osobine sturkture:

- ▶ Elementi se **mogu ponavljati** (dozvoljeni duplikati)
- ▶ Mogu se **ugneždavati** — lista može da sadrži druge liste
- ▶ **Ne koristi se** kao ključ u rečniku (*nije hashable*)
- ▶ **Više memorije** troši u poređenju sa tuple zbog promenjivosti

Operacije nad listom



Pristup i izmena:

```
niz[0]          # Prvi element  
niz[-1]         # Poslednji element  
niz[2] = 10     # Promena vrednosti na indeksu 2
```



Sečenje (slicing):

```
niz[1:3]        # Elementi na indeksima 1 i 2  
niz[:2]         # Prva dva elementa  
niz[::-2]       # Svaki drugi element  
niz[::-1]       # Obrnuta lista
```

+ – Dodavanje i uklanjanje elemenata:

```
niz.append(5)    # Dodaje na kraj  
niz.insert(1, 6)  # Ubacuje 6 na poziciju 1  
niz.extend([7, 8]) # Dodaje više elemenata na kraj  
  
niz.pop()        # Uklanja poslednji element  
niz.pop(2)        # Uklanja element na indeksu 2  
niz.remove(6)      # Uklanja prvi pojavni element sa vrednošću 6  
del niz[1]        # Briše element na poziciji 1  
del niz[1:3]       # Briše elemente na indeksima 1 i 2
```



Dodatno — korisne operacije:

```
len(niz)         # Dužina liste  
min(niz), max(niz) # Najmanji i najveći element  
sum(niz)         # Zbir svih elemenata (ako su brojevi)  
niz.index(9)      # Pronalazi indeks prve pojave vrednosti 9  
niz.count(4)      # Broj pojavljivanja vrednosti 4
```

Iteracija i tipične primene

Iteracija kroz listu:

```
for x in niz:  
    print(x)  
  
for i in range(len(niz)):  
    print(i, niz[i])
```

Tipične primene:

- ▶ Sumiranje, pronalaženje maksimuma/minimuma
- ▶ Pretraživanje i filtriranje
- ▶ Građenje DP tabela:

```
dp = [0] * (n + 1)
```

Napomena:

- ▶ Lista podržava $O(1)$ pristup po indeksu
- ▶ Umetanje/brisanje nije uvek efikasno ako nije na kraju

Česte greške i saveti

⚠ Česte greške:

- ▶ IndexError zbog **nepostojećeg indeksa**
- ▶ Modifikovanje liste **dok se prolazi kroz nju**

💡 Saveti:

- ▶ Za dodavanje **više elemenata**: extend()
- ▶ Za **kopiranje**: niz.copy() ili niz[:]
- ▶ Ako je potrebna **fiksna dužina**:

```
niz = [0] * 10
```

Set

? Šta je set u Pythonu?

- ▶ Set je **kolekcija jedinstvenih elemenata**
- ▶ Sličan je matematičkom skupu
- ▶ Koristi se kada je važna pripadnost, a ne redosled

📌 Primer:

```
# Inicijalizacija seta sa elementima
moj_set = {1, 2, 3}

# Prazan set - obavezno koristi set(), ne {}
prazan_set = set()
```

📦 Osobine strukture:

- ▶ Ne dozvoljava **duplike**
- ▶ **Brza provera** postojanja
- ▶ Nema definisan **redosled** — ne može se osloniti na pozicije
- ▶ Ne podržava **indeksiranje** ni **sečenje**

Operacije nad setom

Dodavanje i uklanjanje elemenata:

```
skup.add(5)          # Dodaje jedan element
skup.update([6, 7])  # Dodaje više elemenata
skup.remove(2)        # Uklanja 2, baca grešku ako ne postoji
skup.discard(10)     # Uklanja 10 ako postoji, bez greške
skup.clear()         # Prazni ceo skup
```

Skupovske operacije:

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)    # Unija: {1, 2, 3, 4, 5}
print(a & b)    # Presek: {3}
print(a - b)    # Razlika: {1, 2}
print(a ^ b)    # Simetrična razlika: {1, 2, 4, 5}
```

Iteracija i tipične primene

Iteracija kroz set:

```
for x in skup:  
    print(x)
```

Tipične primene:

- ▶ Uklanjanje duplikata: set(lista)
- ▶ Brza provera postojanja
- ▶ Operacije skupova (unija, presek, razlika)

```
# Uklanjanje duplikata  
lista = [1, 2, 2, 3]  
bez_duplikata = set(lista)  
  
# Provera postojanja  
if 3 in skup:  
    print("Postoji")  
  
# Operacije  
a = {1, 2}  
b = {2, 3}  
print(a & b) # Presek: {2}
```

Česte greške i saveti

⚠ Česte greške:

- ▶ `remove()` prijavljuje **grešku ako element ne postoji**
- ▶ Oslanjanje na redosled (koji ne postoji)

💡 Saveti:

- ▶ Koristiti `discard()` umesto `remove()` ako **nije sigurno** da li element postoji
- ▶ Za efikasnu deduplikaciju: `set(lista)`

Tuple

? Šta je tuple u Pythonu?

- ▶ Tuple je **uređena** kolekcija podataka
- ▶ **Ne može se menjati** — tuple je nepromenjiv (immutable)
- ▶ Može sadržati **različite tipove podataka**

📌 Primer:

```
koordinate = (4, 7)
```



Osobine strukture:

- ▶ Ima **fiksan broj elemenata**
- ▶ Podržava **pristup po indeksu**
- ▶ **Hashable** je — može se koristiti kao ključ u rečniku
- ▶ **Manje memorije** koristi od liste (zbog nemogućnosti izmene)

Operacije nad tuple-om



Pristup elementima:

```
koordinate[0]      # Prvi element  
koordinate[-1]    # Poslednji element
```



Sečenje (slicing):

```
koordinate[0:2]    # Prva dva elementa  
koordinate[::-1]   # Obrnuti redosled
```



Korisne funkcije:

```
len(koordinate)     # Broj elemenata  
min(koordinate)    # Najmanji element (ako su svi brojevi)  
max(koordinate)    # Najveći element
```

Iteracija i tipične primene

Iteracija kroz set:

```
for x in koordinate:  
    print(x)  
  
for i in range(len(koordinate)):  
    print(i, koordinate[i])
```

Tipične primene:

- ▶ Vraćanje više vrednosti iz funkcije:

```
def vrati_par():  
    return (1, 2)
```
- ▶ Fiksirane konfiguracije, podešavanja
- ▶ Lista koordinata: $[(x_1, y_1), (x_2, y_2), \dots]$

Napomena:

- ▶ Tuple podržava $O(1)$ pristup po indeksu i koristi se kada **izmene nisu poželjne**

Česte greške i saveti

⚠ Česte greške:

- ▶ TypeError pri pokušaju izmene:

```
koordinate[0] = 10    # Greška!
```

💡 Saveti:

- ▶ Ako je potrebna **slična struktura, ali promenjiva** — koristiti listu
- ▶ Ako treba **fiksni sadržaj** — koristiti tuple
- ▶ Pretvaranje:

```
tuple(lista)    # Lista → Tuple
list(tuple)     # Tuple → Lista
```

Dict

? Šta je rečnik u Pythonu?

- ▶ Rečnik je **neuređena kolekcija parova ključ-vrednost**
- ▶ Ključevi moraju biti **jedinstveni i hashable**(npr. string, broj, tuple)
- ▶ Vrednosti mogu **biti bilo koji tip podataka**
- ▶ Može se manjati nakon kreiranja

📌 Primer:

```
osoba = {  
    "ime": "Ana",  
    "godine": 25,  
    "student": True  
}
```



Osobine sturkture:

- ▶ **Direktan pristup** vrednostima preko ključeva `dict["ključ"]`
- ▶ **Veoma efikasna pretraga** podataka po ključu
- ▶ Odličan izbor za **mapiranje ili čuvanje strukturiranih podataka**

Operacije nad rečnikom



Dodavanje/ Ažuriranje:

```
osoba["prezime"] = "Petrović"      # Dodavanje novog ključa  
osoba["godine"] = 26                # Ažuriranje postojeće vrednosti
```



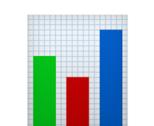
Brisanje:

```
del osoba["student"]  
osoba.pop("godine")  
  
# Brisanje ključa i vrednosti  
# Takođe briše, ali može i da vrati vrednost
```



Pristup:

```
print(osoba["ime"])          # Direktan pristup  
print(osoba.get("visina"))    # Vraća None ako ne postoji
```



Pregled sadržaja:

```
osoba.keys()  
osoba.values()  
osoba.items()  
  
# Svi ključevi  
# Sve vrednosti  
# Parovi (ključ, vrednost)
```



Prolazak kroz dict:

```
for k, v in osoba.items():  
    print(f"{k}: {v}")
```



Ostale korisne metode:

```
osoba.update({"drzava": "Srbija"})    # Dodavanje više vrednosti  
"ime" in osoba                         # Provera da li ključ postoji
```

Iteracija i tipične primene

Iteracija kroz ključeve:

```
for kljuc in osoba:  
    print(kljuc)
```

Iteracija kroz vrednosti:

```
for vrednost in osoba.values():  
    print(vrednost)
```

Iteracija kroz parove:

```
for kljuc, vrednost in osoba.items():  
    print(f'{kljuc}: {vrednost}')
```

Tipične primene:

- ▶ Brojanje pojavljivanja
- ▶ Čuvanje grafova
- ▶ Mapiranje ID-eva, indeksa, vrednosti itd.
- ▶ Keširanje rezultata u dinamičkom programiranju (*memoization*)

Česte greške i saveti

⚠ Česte greške:

- ▶ Pristupanje nepostojećem ključu **bez provere**
- ▶ Ne može se korisititi **promenjiva sturktura** kao ključ (npr. lista)

💡 Saveti:

- ▶ Korisitit `.get()` kada nije sigurno da ključ postoji
- ▶ `.items()` omogućeva elegemntno iteriranje kroz parove
- ▶ **Veoma brzi za pertragu** — idealni za *lookup* tabele
- ▶ Ključevi u *dict* moraju biti **jedinstveni i hashable**



Softverski algoritmi u sistemima automatskog upravljanja

Funkcionalne operacije nad kolekcijama

Lambda funkcije

Šta je lambda funkcija?

- ▶ To je **anonimna (bez imena)** funkcija koja se definiše u jednom redu
- ▶ Koristi se kada je potrebna **karatka funkcija bez posebnog imena**, obično kao argument drugoj funkciji

Osnovna sintaksa:

```
lambda argumenti: izraz
```

- ▶ Primer:

```
f = lambda x: x + 1
print(f(5)) # Output: 6
```

- ▶ Ovo je ekvivalentno sledećem:

```
def f(x):
    return x + 1
```

Česti slučajevi korišćenja lambda funkcija



map() — transformacija elemnata u kolekciji:

```
brojevi = [1, 2, 3, 4]
kvadrati = list(map(lambda x: x**2, brojevi))
# → [1, 4, 9, 16]
```



filter() — filtriranje elemenata po uslovu:

```
brojevi = [1, 2, 3, 4, 5]
parni = list(filter(lambda x: x % 2 == 0, brojevi))
# → [2, 4]
```



sorted() — sortiranje po složenim pravilima:

```
parovi = [(1, 'a'), (3, 'c'), (2, 'b')]
sortirani = sorted(parovi, key=lambda x: x[1])
# → [(1, 'a'), (2, 'b'), (3, 'c')]
```

Kombinovanje kolekcija

Šta radi zip funkcija?

- ▶ Kombinuje više **iterabilnih objekata** tako da formira **parove** (ili **torke**) njihovih elemenata po indeksima

Sintaksa:

```
zip(iterable1, iterable2, ...)
```

Primer upotrebe:

```
imena = ['Ana', 'Marko', 'Ivana']
godine = [23, 30, 27]
spojeno = list(zip(imena, godine))
# [('Ana', 23), ('Marko', 30), ('Ivana', 27)]
```

Tipične primene:

- ▶ Pravljenje parova ključ - vrednost (npr. *dict*)
- ▶ Kombinovanje koordinata (*x,y*)
- ▶ Sinhrono iteriranje kroz više lista (ukoliko liste nisu jednakih dužina, *zip* staje kod **najkratce**)

Kombinovanje kolekcija

🧠 Šta radi enumerate funckija?

- ▶ Dodaje **indeks svakom elementu** iz kolecije tokom iteracije



Sintaksa:

```
enumerate(iterable, start=0)
```



Primer upotrebe:

```
imena = ['Ana', 'Marko', 'Ivana']
for i, ime in enumerate(imena):
    print(f"{i}: {ime}")
# 0: Ana
# 1: Marko
# 2: Ivana
```



Tipične primene:

- ▶ Potreba za **indeksima** pri iteraciji
- ▶ Pravljenje **numerisanih prikaza** (npr. redni brojevi)
- ▶ U kombinaciji **sa uslovima** (if i % 2 == 0 ...)



Softverski algoritmi u sistemima automatskog upravljanja

Zadaci za vežbu

Zadatak 1 – Suma dva broja

Opis:

Dat je niz celih brojeva `nums` i ciljna vrednost `target`. **Naći indekse dva broja iz niza čiji zbir je jednak target.**

Prepostavlja se da postoji **tačno jedno rešenje** i da se **ne može koristiti isti element više puta**. **Redosled indeksa u povratnom rezultatu nije bitan.**

Example 1:

Input: `nums = [2,7,11,15]`, `target = 9`

Output: `[0,1]`

Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

Example 2:

Input: `nums = [3,2,4]`, `target = 6`

Output: `[1,2]`

Example 3:

Input: `nums = [3,3]`, `target = 6`

Output: `[0,1]`

Zadatak 2 – Uklanjanje duplikata

Opis:

Imate niz nums sortiran u ne-opadajućem redosledu. Uklonite sve duplike **bez korišćenja dodatne memorije**, tako da se svaki jedinstveni element pojavi samo jednom u početku niza. Vratite broj jedinstvenih elemenata k.

- 1.Izmenite niz nums tako da prvih k elemenata sadrži sve jedinstvene vrednosti (u originalnom redosledu).
- 2.Vratite k.

Napomena: Sve posle k elemenata nije važno.

Example 1:

Input: nums = [1,1,2]

Output: 2, nums = [1,2,_]

Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:

Input: nums = [0,0,1,1,1,2,2,3,3,4]

Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]

Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Zadatak 3 – Plus 1

Opis:

Data vam je velika cifra predstavljena nizom `digits`, gde svaki element `digits[i]` predstavlja cifru broja, poređanu od **najznačajnije ka najmanje značajnoj** (s leva na desno).

Broj **nema vodeće nule**.

Uvećajte taj broj za **1** i vratite **novi niz cifara** koji predstavlja rezultat.

Example 1:

```
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].
```

Example 2:

```
Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
Incrementing by one gives 4321 + 1 = 4322.
Thus, the result should be [4,3,2,2].
```

Example 3:

```
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].
```

Zadatak 4 – Spajanje soritranih nizova

Opis:

Data su vam dva **celobrojna niza** nums1 i nums2 , sortirana **u ne-opadajućem redosledu**. Takođe su dati brojevi m i n , koji označavaju koliko elemenata iz svakog niza treba uzeti u obzir:

- nums1 ima **dovoljno prostora** da u sebi čuva svih $m + n$ elemenata (tj. prvih m su validni brojevi, a poslednjih n su nule koje treba ignorisati pre spajanja).
- nums2 ima n elemenata koji treba da se spoje sa prvih m elemenata iz nums1 .

Zadatak je da spojite nizove u nums1 tako da dobijete jedan sortiran niz.

Example 1:

Input: $\text{nums1} = [1, 2, 3, 0, 0, 0]$, $m = 3$, $\text{nums2} = [2, 5, 6]$, $n = 3$

Output: $[1, 2, 2, 3, 5, 6]$

Explanation: The arrays we are merging are $[1, 2, 3]$ and $[2, 5, 6]$.

The result of the merge is $[1, \underline{2}, 2, \underline{3}, 5, 6]$ with the underlined elements coming from nums1 .

Example 2:

Input: $\text{nums1} = [1]$, $m = 1$, $\text{nums2} = []$, $n = 0$

Output: $[1]$

Explanation: The arrays we are merging are $[1]$ and $[]$.

The result of the merge is $[1]$.

Zadatak 5 – Većinski element

Opis:

Dat je niz celih brojeva `nums` dužine n .

Vaš zadatak je da pronađete i vratite većinski element — tj. element koji se pojavljuje više od $\lfloor n / 2 \rfloor$ puta (zaokruženo nadole).

Možete pretpostaviti da većinski element uvek postoji u nizu.

Example 1:

```
| Input: nums = [3,2,3]
| Output: 3
```

Example 2:

```
| Input: nums = [2,2,1,1,1,2,2]
| Output: 2
```

Zadatak 6 – Nedostajući broj

Opis:

Dat vam je niz `nums` koji sadrži **n različitih brojeva** iz opsega **[0, n]**.

Vaš zadatak je da **pronađete jedini broj iz tog opsega koji nedostaje** u nizu i vratite ga.

Example 1:

Input: `nums = [3, 0, 1]`

Output: 2

Explanation:

`n = 3` since there are 3 numbers, so all numbers are in the range `[0, 3]`. 2 is the missing number in the range since it does not appear in `nums`.

Example 2:

Input: `nums = [0, 1]`

Output: 2

Explanation:

`n = 2` since there are 2 numbers, so all numbers are in the range `[0, 2]`. 2 is the missing number in the range since it does not appear in `nums`.

Example 3:

Input: `nums = [9, 6, 4, 2, 3, 5, 7, 0, 1]`

Output: 8

Explanation:

`n = 9` since there are 9 numbers, so all numbers are in the range `[0, 9]`. 8 is the missing number in the range since it does not appear in `nums`.

Zadatak 7 – Teemov napad

Opis:

Naš heroj **Teemo** napada neprijateljskog šampiona **Ashe** otrovnim napadima! Svaki put kada Teemo napadne, Ashe ostaje **otrovan tačno duration sekundi**.

Formalno:

- Napad u trenutku t izaziva da Ashe bude otrovana tokom **intervala $[t, t + duration - 1]$** .
- Ako Teemo ponovo napadne **pre nego što prethodni otrov prođe**, tajmer se resetuje — novi otrov traje duration sekundi od novog napada.

Data vam je lista timeSeries (u rastućem redosledu), koja označava **vremena napada**, i broj duration.

Zadatak: **izračunati ukupno koliko je sekundi Ashe bila otrovana**.

Zadatak 7 — Teemov napad

Example 1:

Input: timeSeries = [1,4], duration = 2

Output: 4

Explanation: Teemo's attacks on Ashe go as follows:

- At second 1, Teemo attacks, and Ashe is poisoned for seconds 1 and 2.
- At second 4, Teemo attacks, and Ashe is poisoned for seconds 4 and 5.

Ashe is poisoned for seconds 1, 2, 4, and 5, which is 4 seconds in total.

Example 2:

Input: timeSeries = [1,2], duration = 2

Output: 3

Explanation: Teemo's attacks on Ashe go as follows:

- At second 1, Teemo attacks, and Ashe is poisoned for seconds 1 and 2.
- At second 2 however, Teemo attacks again and resets the poison timer. Ashe is poisoned for seconds 2 and 3.

Ashe is poisoned for seconds 1, 2, and 3, which is 3 seconds in total.