

Vežba 2

U drugom terminu vežbi koje se tiču naprednih koncepata programskog jezika *Python*, studenti bi trebalo da se upoznaju sa sledećim tematskim celinama samostalno proučavajući dolenađene materijale:

- *collections*
 - [poglavlje “collections” \(slajdovi 47-52\)](#)
 - [modul `hpc_kolekcije.py`](#)
- Višetruko nasleđivanje
 - [poglavlje “Višetruko nasleđivanje i MRO” \(slajdovi 53-62\)](#)
 - [modul `visetruko_nasledjivanje.py`](#)
- Metaklase
 - [poglavlje “Metaklase” \(slajdovi 74-82\)](#)
 - [modul `metaklase.py`](#)
- Konkurentno i paralelno programiranje
 - [poglavlje “`async/await`” \(slajdovi 63-73\)](#)
 - [paket `konkurentno_programiranje`](#)

Zadaci za samostalni rad

Nakon prolaska kroz gorenavedene materijale, ali i uz dodatno samostalno istraživanje, potrebno je uraditi sledeće zadatke (teorijske i praktične):

1. U čemu je razlika između kolekcije *OrderDict* i *dict*?
2. Šta *defaultdict* prima kao parametar prilikom instanciranja?
3. Koja kolekcija dosta podseća na C-ovsku strukturu? Navesti primer njene upotrebe.
4. Funkcija *most_common_words* prima 2 parametra:
 - *text* – tekst (*string*) u kome prebrojava reči
 - *n* – prirodan broj.

Ova funkcija treba da u jednom iskazu odredi *n* reči koje se najčešće javljaju u tekstu. Funkcija treba da vrati listu reči, ali ne i koliko puta se svaka od njih pojavljuje.

Napomena: Iskazi u kojima se *import*-uju biblioteke koje dolaze sa standardnom *Python* implementacijom se ne računaju.

5. Šta je *Big O* notacija?
6. Koja je vremenska kompleksnost dodavanja elementa na početak liste, a kolika na početak kolekcije *deque*? Obratiti pažnju na funkciju [deque primer](#).
7. Šta je *Diamond* problem u kontekstu višestrukog nasleđivanja?
8. Kako se *Diamond* problem rešava u *Python* programskom jeziku?
9. Navesti dva pravila *MRO*-a?
10. Šta predstavlja *super* u programskom jeziku *Python*?
11. Da li programski jezik *Java* nativno podržava višetruko nasleđivanje? Kako se ono realizuje?

12. Na osnovu dole navedenog isečka koda odrediti *MRO* lanac koji odgovara klasi *F*.

```
1
2 class A:
3     pass
4
5 class B(A):
6     pass
7
8 class C(A):
9     pass
10
```

```
11
12 class D(B, C):
13     pass
14
15 class E(B, C):
16     pass
17
18 class F(E, D):
19     pass
20
```

13. Da li su klase objekti u programskom jeziku *Python*?
14. Ko je zadužen za kreiranje klasa?
15. Za šta se sve koristi *type* u *Python* programskom jeziku? Navesti primere.
16. Za šta je *class* lepša sintaksa (engl. *syntactic sugar*)? Navesti primer.
17. Šta su *callback* funkcije?
18. Koja je razlika između *preemptive* i *non-preemptive multitasking*-a?
19. Šta su korutine (u programskom jeziku *Python*)?
20. Šta je “pumpa događaja” (engl. *event loop*)?
21. Šta je asinhrono programiranje?
22. Šta je konkurentno programiranje?
23. Šta je paralelno programiranje?
24. Koja je razlika između konkurentnog i paralelnog programiranja?
25. Šta je proces?
26. Šta je nit?
27. Koja je razlika između procesa i niti?
28. Kako komuniciraju niti, a kako procesi?
29. Koja je razlika između generatora i korutine?
30. Da li se generatori u programskom jeziku *Python* mogu ulančavati? Ukoliko mogu, navesti primer, u suprotnom navesti razlog zašto ne.
31. Šta je profajler (engl. *profiler*)?
32. Šta je *GIL* u programskom jeziku *Python*?
33. Koliko niti može istovremeno da izvršava kod u jednom *Python* procesu? Zašto?
34. Šta je *race condition*? Navesti primer.
35. Šta je trka do podataka (engl. *data race*)? Navesti primer.
36. Šta je štetno preplitanje? Navesti primer.
37. Šta je *deadlock*? Navesti primer.
38. Da li se kod asinhronog programiranja javlja problem štetnog preplitanja? Zašto?
39. U čemu je razlika između *I/O Bound* i *CPU Bound* programa?
40. Na koje je sve načine moguće optimizovati *I/O Bound* programe u programskom jeziku *Python*? Koji je od njih najbolji i zašto?
41. Na koje sve načine je moguće optimizovati *CPU Bound* programe u programskom jeziku *Python*? Koji je od njih najbolji i zašto?
42. Šta će se desiti ukoliko se prilikom asinhronog programiranja koriste zadaci koji nisu kooperativni (ne komuniciraju sa *event loop*-om)?
43. Šta je jako, a šta slabo skaliranje?
44. Prilikom paralelizacije *CPU Bound* programa *multiprocessing* bibliotekom, na koji način se bira broj procesa? Šta sve treba uzeti u obzir prilikom razmatranja?

45. Dat je sledeći isečak koda.

- (a) Da li je navedeni program *I/O Bound* ili *CPU Bound*?
- (b) Označiti neoptimalan deo koda.
- (c) Na koje se sve načine dati programski kod može optimizovati?
- (d) Implementirati optimalan ekvivalentni programski kod.

```
1  import time
2
3
4  import requests
5
6
7  def download_site(url, session):
8      with session.get(url) as response:
9          print(f"Read {len(response.content)} from {url}")
10
11
12  def download_all_sites(sites):
13      with requests.Session() as session:
14          for url in sites:
15              download_site(url, session)
16
17
18  if __name__ == "__main__":
19      sites = [
20          "http://www.jython.org",
21          "http://olympus.realpython.org/dice",
22          ] * 80
23      start_time = time.time()
24      download_all_sites(sites)
25      duration = time.time() - start_time
26      print(f"Downloaded {len(sites)} in {duration} seconds")
27
```

46. Dat je sledeći isečak koda.

- (a) Da li je navedeni program *I/O Bound* ili *CPU Bound*?
- (b) Označiti neoptimalan deo koda.
- (c) Na koje se sve načine dati programski kod može optimizovati?
- (d) Implementirati optimalan ekvivalentni programski kod.

```
1
2 import time
3
4
5 def find_sum(number):
6     return sum(i * i for i in range(number))
7
8
9 def find_sums(numbers):
10     for number in numbers:
11         find_sum(number)
12
13
14 if __name__ == "__main__":
15     numbers = [5_000_000 + x for x in range(20)]
16
17     start_time = time.time()
18     find_sums(numbers)
19     duration = time.time() - start_time
20     print(f"Duration {duration} seconds")
21
```