Primjer 1. kolokvija iz kolegija Raspodijeljeni sustavi

Akademska godina 2025./2026.

Definirajte novi radni direktorij pod nazivom `rs-mid-<vaše_ime>-<vaše_prezime>`.

Za svaki zadatak stvorite novi direktorij unutar radnog direktorija naziva `zadatak-<broj_zadatka>`.

Zadatak 1 (7 bodova)

1.1 Potrebno je izraditi novu Python datoteku u koju ćete pohraniti kolekciju `razredi_studenti` iz priloga. Ovu datoteku, proizvoljnog naziva, potrebno je pohraniti u direktorij `./zadatak_1/data` gdje `.` predstavlja korijenski direktorij vašeg radnog direktorija, odnosno `rs-mid-<vaše_ime>-<vaše_prezime>`.

Stvorite `index.py` unutar direktorija `/zadatak-1` te uvezite podatke.

Implementirajte sljedeće funkcije/izraze:

1.2 `dohvati_studente_iz_razreda(razredi_studenti: list, naziv_razreda: str) -> list`: Funkcija prima kolekciju i naziv razreda te vraća listu imena i prezimena svih studenata iz razreda. *Zadatak nema implementacijskih ograničenja.*

1.3 `prosjek_studenta(razredi_studenti: list, ime_prezime: str) -> float`: Funkcija prima kolekciju i *ime_prezime* studenta te vraća prosjek ocjena iz svih kolegija. Ako student nije pronađen, funkcija treba vratiti `None`. *Zadatak nema implementacijskih ograničenja.*

1.4 *comprehension* izraz koji vraća listu n-torki, gdje svaka n-torka ima dvije vrijednosti: naziv razreda i broj studenata u tom razredu.

1.5 jedan ili više *comprehension* izraza s kojima ćete pohraniti imena i prezimena studenata iz razreda `1B` u listu.

Zadatak 2 (10 bodova)

2.1 Definirajte novi Python paket `fakultet` unutar direktorija `zadatak-2`.

Unutar paketa potrebno je izraditi modul `student.py` u kojem ćete definirati klasu `Student` s navedenim atributima i metodama:

- `ime (str)`:ime studenta
- `prezime (str)`: prezime studenta
- `razred (str)`: naziv (šifra) razreda gdje student pripada
- `kolegij_ocjene (dict)`: rječnik gdje je ključ naziv kolegija, a vrijednost ocjena iz tog kolegija

Unutar istog paketa, definirajte i modul `podaci.py` u kojem ćete samo pohraniti kolekciju `razredi_studenti` iz priloga.

2.2 Stvorite `index.py` izvan paketa te uvezite module `student.py` i `podaci.py`.

Stvorite novu instancu klase `Student` za svakog studenta iz kolekcije `razredi_studenti`. Pohranite u listu sve stvorene instance klase `Student` te ispišite podatke u sljedećem formatu za svakog studenta:

```
Student: <ime> <prezime>, Razred: <razred>, Kolegiji i ocjene: {<kolegij1>: <ocjena1>,
<kolegij2>: <ocjena2>, ...}
```

2.3 Dodajte metodu `prosjek_ocjena(self) -> float` koja računa i vraća prosjek ocjena studenta.

- Ako student nema niti jedan kolegij, metoda treba vratiti `0.0`.
- Prosjek ocjena treba biti zaokružen na jednu decimalu.

2.4 Dodajte metodu `promjena_razreda(self, novi_razred: str) -> None` koja mijenja atribut `razred` na vrijednost proslijeđenu kao argument `novi_razred`.

- Ako se vrijednost `novi_razred` ne nalazi u listi mogućih razreda (koju je potrebno izgraditi mapiranjem podataka iz modula `podaci.py`), potrebno je podići iznimku `ValueError` s porukom: `"Razred {novi_razred} nije dopušten."`
- U suprotnom, atribut `razred` treba biti ažuriran na novu vrijednost.

Zadatak 3 (10 bodova)

3.1 Definirajte sinkronu funkciju `zadatak(sekunde: int) -> str` koja blokira izvođenje programa na broj sekundi proslijeđen kao argument `sekunde`. Nakon isteka vremena, funkcija treba vratiti poruku: `"Zadatak završen nakon {sekunde} sekundi."`

- Unutar `main` funkcije, pokrenite tri poziva funkcije `zadatak` s vrijednostima `3`, `2` i `1` sekundu, sekvencijalno. Izmjerite ukupno vrijeme izvođenja `main` funkcije.

3.2 Definirajte ekvivalentnu asinkronu korutinu `asinkroni_zadatak(sekunde: int) -> str` koja simulira čekanje asinkronom metodom čekanja iz modula `asyncio`.

- Unutar `main` korutine, pozovite konkurentno tri puta korutinu `asinkroni_zadatak` s istim vrijednostima iz prethodnog zadatka. Konkurentnost implementirajte sa i bez korištenja funkcije `asyncio.gather()`. Izmjerite ukupno vrijeme izvođenja `main` korutine. Objasnite ponašanje *asyncio event loopa* na primjeru ovog zadatka.

3.3 Definirajte sinkronu funkciju `posalji_zahtjev(url: str) -> dict` koja šalje HTTP GET zahtjev na URL proslijeđen kao argument `url` koristeći sinkronu biblioteku za slanje HTTP zahtjeva.

- Unutar `main` funkcije, pozovite funkciju `posalji_zahtjev` tri puta prema servisu: `https://jsonplaceholder.typicode.com/todos/1`. Pohranite `title` iz tijela svakog HTTP odgovora u listu te ispišite tu listu. Izmjerite ukupno vrijeme izvođenja `main` funkcije - zaokružite vrijeme na dvije decimale.

3.4 Definirajte asinkronu korutinu `asinkroni_posalji_zahtjev(url: str) -> dict` koja šalje HTTP GET zahtjev na isti URL koristeći asinkronu biblioteku za slanje HTTP zahtjeva.

- Unutar `main` korutine, pozovite korutinu `asinkroni_posalji_zahtjev` tri puta konkurentno prema istom servisu. Pohranite `title` iz tijela svakog HTTP odgovora u listu te ispišite tu listu. Izmjerite i usporedite vrijeme izvođenja `main` korutine s `main` funkcijom iz *Zadataka 3.3*. Koji model je brži i koliko puta? Objasnite.

Zadatak 4 (13 bodova)

4.1 Razvijate raspodijeljeni sustav za praćenje protoka motornih vozila na autocestama. Sustav se sastoji od više kamera smještenih na različitim lokacijama duž nadzirane dionice. Svaka kamera komunicira s centralnim poslužiteljem putem mreže i šalje podatke o broju detektiranih vozila, jednom kad poslužitelj zatraži te podatke.

Definirajte korutinu `get_camera_data(camera_id: int) -> dict` koja simulira dohvat podataka s kamere identificirane parametrom `camera_id`. Kako mrežni prijenos može biti podložan kašnjenjima, potrebno je simulirati slučajno kašnjenje u rasponu od `0.1` do `1` sekunde (`random.uniform(0.1, 1)`). Ovo nasumično kašnjenje treba se nadodati na konstantno vrijeme obrade od `0.5` sekundi prije povratka rezultata.

Podaci koje korutina vraća trebaju biti u obliku rječnika s ključevima:

- `camera_id`: identifikator kamere
- `timestamp`: trenutak kad je kamera generirala podatke (koristite `datetime.now().isoformat()`)
- `vehicle_count`: slučajan broj vozila (primjerice `random.randint(5, 20)`)

4.2 Definirajte glavnu korutinu `main()` koja predstavlja izvršavanje programa na centralnom poslužitelju. Poslužitelj treba, u jednom ciklusu istovremeno dohvaćati podatke s ukupno 5 kamera (`camera_id` od `1` do `5`). Pretpostavlja se da poslužitelj prikuplja podatke svakih 5 sekundi (vrijeme trajanja jednog ciklusa), tijekom ukupnog vremena izvođenja `main` korutine od 30 sekundi.

Za svaki 5-sekundni ciklus potrebno je:

- dohvatiti podatke svih 5 kamera konkurentno
- pohraniti sve prikupljene rezultate u listu
- nakon završetka svakog ciklusa ispisati:
    - broj ciklusa (npr. "Ciklus 1", "Ciklus 2", ...)
    - *timestamp* kamere koja je prva vratila podatke i kamere koja je posljednja poslala podatke (koristite `datetime.fromisoformat` za parsiranje *timestamp* stringa)
    - ukupan broj detektiranih vozila na svim kamerama

Na kraju izvođenja ispisati prosječan broj vozila po kameri.

4.3 Izmijenite korutinu `get_camera_data` na način da povećate kašnjenje simuliranog prijenosa podataka na maksimalnih 5 sekundi.

U korutinu `main()` dodajte mehanizam vremenskog ograničenja (*timeout*) od 3 sekunde za dohvat podataka s pojedine kamere. Ako dohvat ne završi unutar tog vremena, centralni poslužitelj treba preskočiti rezultat te kamere i ispisati upozorenje: `"Upozorenje: Dohvat podataka s kamere {camera_id} je istekao."`. Takav neuspjeli dohvat ne ulazi u izračun statistika za taj ciklus.