Raspodijeljeni sustavi (RS)

Nositelj: doc. dr. sc. Nikola Tanković
Asistent: Luka Blašković, mag. inf.

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli

(4) Asinkroni Python: Slanje konkurentnih HTTP zahtjeva

HTTP (Hypertext Transfer Protocol) je protokol koji omogućuje prijenos podataka između klijenta i servera na webu. Asinkroni Python omogućuje nam da programiramo poslužitelje koji mogu istovremeno obrađivati više zahtjeva bez blokiranja glavnog toka programa, čime se postiže bolja učinkovitost, osobito u aplikacijama koje zahtijevaju visoku propusnost, kao što su web servisi i API klijenti. Korištenjem biblioteke kao što je *aiohttp*, možemo jednostavno implementirati asinkrone HTTP klijente i poslužitelje u Pythonu. U prošloj skripti upoznali ste se s asinkronim programiranjem u Pythonu pomoću *asyncio* biblioteke, a u ovoj ćete naučiti kako kombinirati asinkrone funkcionalnosti s HTTP zahtjevima i odgovorima.

Posljednje ažurirano: 21.11.2024.

1. Ponavljanje HTTP protokola

HTTP (*eng. Hypertext Transfer Protocol*) odnosi se na protokol koji se koristi za prijenos podataka putem weba. Omogućuje web preglednicima, udaljenim poslužiteljima i ostalim dijelovima sustavne cjeline da komuniciraju međusobno. HTTP je protokol bez stanja (eng. *stateless*), što znači da svaki zahtjev klijenta poslužitelju ne ovisi o prethodnim zahtjevima. Svaki zahtjev se tretira kao zaseban zahtjev, bez obzira na prethodne.

Tipična HTTP komunikacijski model (**klijent** ↔ **poslužitelj**) sastoji se HTTP zahtjeva (eng. *request*) i HTTP odgovora (eng. *response*).

- **HTTP zahtjev** (*eng. HTTP request*): odnosi se na **zahtjev koji klijent šalje poslužitelju**. Npr. web preglednik šalje zahtjev za resurs udaljenom poslužitelju
- **HTTP odgovor** (*eng. HTTP response*): odnosi se na **odgovor koji poslužitelj šalje klijentu**. Npr. poslužitelj šalje odgovor s JSON podacima u tijelu odgovora

1.1. Struktura HTTP zahtjeva (eng. HTTP request)

- **Metoda** (*eng. method*): odnosi se na vrstu zahtjeva (npr. `GET`, `POST`, `PUT`, `PATCH`, `DELETE`)
- **URL** (*eng. Uniform Resource Locator*): odnosi se na adresu resursa na poslužitelju (npr. `https://api.github.com/users/neki_korisnik`)
  - **Shema** (*eng. scheme*): odnosi se na protokol koji se koristi (npr. `https`)
  - **Domena** (*eng. domain*): odnosi se na ime domene poslužitelja (npr. `api.github.com`)
  - **Route parametar** (*eng. route*): odnosi se na dinamički dio URL-a, najčešće za identifikaciju pojedinog resursa (npr. `/users/:id`)
  - **Query parametar** (*eng. query*): odnosi se na dodatne parametre upita, najčešće za filtriranja, sortiranja i sl. (npr. `?page=1&limit=10`)
  - **Fragment** (*eng. fragment*): odnosi se na oznaku dijela resursa (npr. `#section1`)
- **Zaglavlja** (*eng. headers*): odnose se na dodatne informacije o zahtjevu (npr. `Content-Type: text/html; charset=utf-8`)
- **Tijelo** (*eng. body*): odnosi se na podatke koji se šalju s zahtjevom (npr. `JSON`)
- **Verzija protokola** (*eng. protocol version*): odnosi se na verziju HTTP protokola (npr. `HTTP/1.1`)

HTTP metode koje se najčešće koriste su:

- **GET**: dohvaća resurs/resurse s poslužitelja (npr. podatke o korisniku)
- **POST**: šalje podatke na poslužitelj (npr. podatke iz forme)
- **PUT**: ažurira resurs na poslužitelju u cijelosti (npr. zamjenjuje postojeće podatke o korisniku novima)
- **PATCH**: ažurira resurs na poslužitelju djelomično (npr. izmjenjuje lozinku korisnika)
- **DELETE**: briše resurs s poslužitelja (npr. briše korisnika)

> **Podsjetnik**: Uobičajeno je koristiti **GET** metodu za dohvaćanje podataka, s dodatnim query parametrima za filtriranje, sortiranje, paginaciju i sl. ili s dodatnim route parametrima za identifikaciju pojedinog resursa, npr. kada želimo dohvatiti samo jednog korisnika. Nije po standardu slati **tijelo zahtjeva** unutar **GET** metode.

> **Podsjetnik**: Metodu **POST** koristimo kada želimo poslati podatke na poslužitelj, npr. kada šaljemo podatke iz forme ili kada želimo izraditi novi resurs na poslužitelju. U tom slučaju, **šaljemo podatke u tijelu zahtjeva**, najčešće u JSON formatu iako je moguće koristiti i druge. Razlog zašto šaljemo podatke u tijelu zahtjeva, a ne kao query parametre, jest što je tijelo zahtjeva skriveno od korisnika - ne pojavljuje se u URL-u.

> **Podsjetnik**: Metode **PUT** i **PATCH** koristimo kada želimo ažurirati resurs na poslužitelju. Razlika između njih je što **PUT** zamjenjuje cijeli resurs novim podacima, dok **PATCH** ažurira resurs djelomično. Primjerice, kada ažuriramo korisnika, **PUT** metodom zamijenili bismo sve podatke o korisniku novima, dok bismo **PATCH** metodom mogli ažurirati samo određeni podatak ili više njih. Podatke koje ažuriramo također šaljemo u tijelu zahtjeva.

Primjer: HTTP zahtjev koji dohvaća podatke o korisniku s GitHuba `pero_peric`:

```
GET https://api.github.com/users/pero_peric HTTP/1.1
```

Primjer: HTTP zahtjev koji šalje podatke o novom korisniku na poslužitelj:

```
POST https://api.github.com/users HTTP/1.1
Content-Type: application/json

{
  "username": "pero_peric",
  "email": "pperic@gmail.com",
  "password": "pero123"
}
```

Primjer: HTTP zahtjev koji ažurira username korisnika `pero_peric` na poslužitelju:

```
PATCH https://api.github.com/users/pero_peric HTTP/1.1

{
  "username": "pero_peric_2"
}
```

Primjer: HTTP zahtjev koji briše korisnika `pero_peric` s poslužitelja:

```
DELETE https://api.github.com/users/pero_peric HTTP/1.1
```

Primjer: HTTP zahtjev koji dohvaća samo korisnike s imenom `pero`:

```
GET https://api.github.com/users?name=pero HTTP/1.1
```

Primjer: HTTP zahtjev koji zamjenjuje sve podatke o korisniku `pero_peric` novima:

```
PUT https://api.github.com/users/pero_peric HTTP/1.1

{
  "username": "pero_peric_2",
  "email": "pperic2@gmail.com",
  "password": "ppppero1234"
}
```

1.2 Struktura HTTP odgovora (eng. HTTP response)

- **Statusna linija** (*eng. status line*): odnosi se na **statusni kod** i **poruku** (npr. `200 OK`) gdje je `200` statusni kod, a `OK` poruka
- **Zaglavlja** (*eng. headers*): odnose se na dodatne informacije o odgovoru (npr. `Content-Type: application/json`)
- **Tijelo** (*eng. body*): odnosi se na podatke koji se šalju s odgovorom (npr. `JSON`)
- **Verzija protokola** (*eng. protocol version*): odnosi se na verziju HTTP protokola (npr. `HTTP/1.1`)

> **Podsjetnik**: **Statusni kodovi** (*eng. HTTP status codes*) su brojevi koji se koriste u **HTTP odgovorima** kako bi klijentu dali informaciju u kojem je stanju zahtjev koji je poslao. Drugim riječima, ako klijent pošalje zahtjev koji rezultira greškom, poslužitelj uz odgovarajuću poruku vraća i statusni kod koji označava vrstu greške.

Statusne kodove možemo podijeliti u sljedeće skupine:

- `1xx` (100 - 199) - Informacijski odgovori (eng. *Informational responses*): Poslužitelj je primio zahtjev te ga i dalje obrađuje
- `2xx` (200 - 299) - Odgovori uspjeha (*eng. Successful responses*): Zahtjev klijenta uspješno primljen i obrađen
- `3xx` (300 - 399) - Odgovori preusmjeravanja (*eng. Redirection messages*): Ova skupina kodova govori klijentu da mora poduzeti dodatne radnje kako bi dovršio zahtjev
- `4xx` (400 - 499) - Greške na strani klijenta (*eng. Client error responses*): Sadrži statusne kodove koji se odnose na greške nastale na klijentskoj strani
- `5xx` (500 - 599) - Greške na strani poslužitelja (*eng. Server error responses*): Sadrži statusne kodove koji se odnose na greške nastale na poslužiteljskoj strani

Statusni kodovi neizbježan su dio HTTP komunikacije, a njihovom primjenom **standardiziramo komunikaciju između klijenta i poslužitelja**. Na taj način, klijent može interpretirati odgovor poslužitelja i ovisno o statusnom kodu poduzeti odgovarajuće radnje.

Statusnih kodova ima mnogo, a svaki od njih ima svoje značenje. Možete pronaći **popis** i **definicije** svih statusnih kodova na ovoj poveznici.

Međutim, u praksi se ne najčešće ne koriste svi statusni kodovi, već nekolicina njih. Evo nekoliko najčešće korištenih statusnih kodova:

- `200` - OK: Zahtjev je uspješno primljen i obrađen (npr. GET zahtjev za dohvat svih resursa)

- `201` - Created: Resurs je uspješno stvoren (npr. nakon slanja POST zahtjeva)
- `400` - Bad Request: Zahtjev nije moguće obraditi zbog neispravnih podataka (npr. korisnik je poslao neispravan ID resursa u zahtjevu)
- `404` - Not Found: Resurs nije pronađen (npr. korisnik je poslao ID resursa koja ne postoji na poslužitelju)
- `500` - Internal Server Error: Opća greška na poslužitelju (npr. greška prilikom obrade zahtjeva, najvjerojatnije zbog greške u kodu na poslužitelju)

Postoji puno varijacija 4xx, 5xx i 2xx statusnih kodova, pa tako imamo:

- `401` - Unauthorized: Korisnik nije autoriziran za pristup resursu (npr. korisnik nema prava pristupa resursu jer nije prijavljen)
- `204` - No Content: Zahtjev je uspješno primljen i obrađen, ali nema sadržaja za prikazati (npr. nakon brisanja resursa)
- `403` - Forbidden: Korisnik nema prava pristupa resursu (npr. korisnik nema prava pristupa resursu jer nije administrator)
- `301` - Moved Permanently: Resurs je trajno premješten na novu lokaciju (npr. kada se mijenja URL resursa)
- `503` - Service Unavailable: Poslužitelj nije dostupan (npr. poslužitelj je preopterećen)
- `409` - Conflict: Zahtjev nije moguće obraditi zbog konflikta (npr. korisnik pokušava ažurirati resurs koji je već ažuriran, npr. kod PUT/PATCH zahtjeva)

> **VAŽNO:** Za studente koji imaju poteškoća s razumijevanjem HTTP protokola iz sažetka danog u ovom poglavlju, preporuka je da prouče skriptu WA1 iz kolegija Web aplikacije s prijediplomskog studija Informatike u Puli. Potrebno se prijaviti s `AAI@EduHr računom na UNIPU` domeni.

2. Slanje konkurentnih HTTP zahtjeva pomoću `aiohttp` biblioteke

`aiohttp` (*Asynchronous HTTP Client/Server for asyncio and Python*) je biblioteka koja omogućuje **asinkrono programiranje HTTP klijenata i poslužitelja u Pythonu**. Ova datoteka izgrađena je na temelju `asyncio` biblioteke s kojom smo se upoznali u skripti `RS3`.

`aiohttp` biblioteka omogućuje nam da jednostavno implementiramo asinkrone HTTP klijente i poslužitelje u Pythonu, što je korisno u kontekstu razvoja i testiranja malih web servisa koji zahtijevaju visoku propusnost. Dodatno, datoteka pruža podršku za WebSocket protokol, što je korisno za razvoj aplikacija u stvarnom vremenu (eng. *real-time applications*).

Za razliku od `asyncio` biblioteke koja je ugrađena u Python 3.7+, `aiohttp` biblioteku potrebno je instalirati ručno:

```
pip install aiohttp
```

Napomena, kod instalacije vanjskih paketa **preporučuje se korištenje virtualnog okruženja** kako bi se izbjegli konflikti između paketa.

Ako ste se odlučili koristiti `conda` alat za upravljanje virtualnim okruženjima, napraviti novo okruženje naziva `rs4` prije nego instalirate `aiohttp` biblioteku:

```
conda create --name rs4 python=3.13
```

Aktivirajte novo okruženje:

```
conda activate rs4
```

Unutar VS Codea promijenite interpreter na novo kreirano okruženje `rs4` kako biste izbjegli linting greške.

> Sada možete instalirati biblioteke 📚📚📚

2.1 Kako šaljemo HTTP zahtjeve sinkrono?

Međutim, prije nego se upoznamo s asinkronim načinom definiranja HTTP klijenata, vrijedno je prisjetiti se kako to radimo sinkrono, koristeći biblioteku `requests`.

`requests` je popularna biblioteka za rad s HTTP zahtjevima u Pythonu koja omogućuje jednostavno slanje zahtjeva na poslužitelj i primanje odgovora. Međutim, `requests` je **sinkrona biblioteka**, što znači da će svaki zahtjev blokirati izvođenje programa dok se ne primi odgovor.

Kako bismo poslali HTTP zahtjev koristeći `requests` biblioteku, prvo je potrebno instalirati biblioteku:

```
pip install requests
```

Uključimo `requests` biblioteku:

```
import requests
```

Jednostavni primjer slanja GET zahtjeva na poslužitelj. Zahtjev ćemo poslati na Cat Facts API koji vraća nasumične činjenice o mačkama:

```
import requests

response = requests.get("https://catfact.ninja/fact")
print(response.text)
```

Ako pokrenemo ovaj kod, dobit ćemo nasumični odgovor u obliku rječnika s ključevima `fact` i `length`:

```
{
  "fact": "The life expectancy of cats has nearly doubled over the last fifty years.",
  "length": 73
}
```

Možemo provjeriti statusni kod odgovora:

```
print(response.status_code) # 200
```

Rekli smo da u sinkronim programima, svaki zahtjev koji pošaljemo **čeka na odgovor prethodnog** prije nego pošaljemo novi. Ako neki zahtjevi traju dugo, to može značajno usporiti izvođenje programa.

Primjer: Poslat ćemo 5 zahtjeva na *Cat Facts API*, kod za slanje možemo spakirati u jednostavnu funkciju koja šalje GET zahtjev i ispisuje rezultat. Zahtjev možemo dohvatiti pod ključem `fact`, ali prije tog moramo napraviti deserijalizaciju JSON odgovora koristeći metodu `json()`:

```
import requests

def send_request():
  response = requests.get("https://catfact.ninja/fact")
  fact = response.json()["fact"]
  print(fact)

print("Šaljemo 1. zahtjev...")
send_request()

print("Šaljemo 2. zahtjev...")
send_request()

print("Šaljemo 3. zahtjev...")
send_request()

print("Šaljemo 4. zahtjev...")
send_request()

print("Šaljemo 5. zahtjev...")
send_request()
```

Vidimo da je za izvršavanje svakog zahtjeva potrebno pričekati odgovor prethodnog; na taj način smo napisali kod i to je OK. Ukupno vrijeme trajanja ovog programa je prosječno 1-2 sekunde, ovisno o brzini interneta.

Možemo koristiti biblioteku `time` kako bismo preciznije izmjerili vrijeme izvršavanja programa:

```
import requests
import time

def send_request():
  response = requests.get("https://catfact.ninja/fact")
  fact = response.json()["fact"]
  print(fact)

start = time.time()

print("Šaljemo 1. zahtjev...")
send_request()

print("Šaljemo 2. zahtjev...")
send_request()
```

```
print("Šaljemo 3. zahtjev...")
send_request()

print("Šaljemo 4. zahtjev...")
send_request()

print("Šaljemo 5. zahtjev...")
send_request()

end = time.time()
print(f"Izvršavanje programa traje {end - start:.2f} sekundi.")
```

Poslat ćemo 15 zahtjeva, kod možemo i strukturirati u petlju:

```
import requests
import time

def send_request():
  response = requests.get("https://catfact.ninja/fact")
  fact = response.json()["fact"]
  print(fact)

start = time.time()

for i in range(15):
  print(f"Šaljemo {i + 1}. zahtjev...")
  send_request()

end = time.time()
print(f"Izvršavanje programa traje {end - start:.2f} sekundi.")
```

Prosječno vrijeme trajanja programa iznad je 3-4 sekunde. Ako povećamo broj zahtjeva, **vrijeme izvršavanja će se povećati proporcionalno broju zahtjeva**. Obzirom da vrijeme izvođenja programa direktno ovisi o broju iteracija `i`, možemo reći da je vremenska složenost `O(n)`.

Zahtjeve smo slali sinkrono (sekvencijalno), i to je vrlo vidljivo na ovom primjeru. Zamislimo da šaljemo 1000 zahtjeva, ili 10 000 zahtjeva - program bi trajao jako dugo, a poslužitelj bi morao čekati na svaki zahtjev. Na ovaj način, ne iskorištavamo puni potencijal poslužitelja, a aplikacije koje razvijamo nisu skalabilne.

Idemo vidjeti kako bismo to mogli riješiti asinkronim programiranjem odnosno **konkurentnim slanjem zahtjeva** na poslužitelj pomoću `aiohttp` biblioteke.

2.2 Asinkrono slanje HTTP zahtjeva

Cilj nam je poslati više zahtjeva na *Cat Facts API* i postići brže vrijeme izvršavanja programa (ne želimo da slanje i ispis rezultata 15 zahtjeva traje gotovo 4 sekunde).

Ako već niste, instalirajte `aiohttp` biblioteku.

Kako zahtjeve šaljemo konkurentno, najpraktičnije je kod spakirati u korutine. Međutim, kako bi radili s korutinama i asinkronim programiranjem općenito, svakako je potrebno uključiti i `asyncio` biblioteku.

```
import aiohttp
import asyncio
import time
```

Nećemo više koristiti `requests` biblioteku, zamijenili smo je s `aiohttp`.

Biblioteka `requests` u pozadini je definirala korisničku sesiju (eng. *session*) koja je omogućavala **ponovnu upotrebu veze s poslužiteljem**, odnosno pohranu HTTP zaglavlja, autentifikacije, kolačića i drugih objekata koji se ponavljaju u svakom zahtjevu. Na taj način, umjesto da se radi nova sesija za svako slanje zahtjeva, moguće je **ponovno koristiti već postojeću sesiju**.

U `aiohttp` biblioteci, potrebno je naglasiti definiranje sesije - ona nam omogućuje iste funkcionalnosti koje su prethodno navedene.

2.2.1 Context Manager `with`

Koncept **kontekstnog menadžera** (*eng. Context Manager*) u Pythonu omogućavaju nam alokaciju i dealokaciju resursa, odnosno upravljanje resursima koji se koriste u bloku koda.
Najčešće korišteni primjer *context managera* u Pythonu je naredba `with` koju koristimo kako bismo definirali **blok koda za rad s resursima** koje treba eksplicitno **(1) otvoriti**, **(2) koristiti** i **(3) zatvoriti**.

Primjerice:

- **datoteke** (otvaranje →čitanje/pisanje → zatvaranje)
- **mrežne veze** (otvaranje → slanje zahtjeva → zatvaranje)
- **baze podataka** (otvaranje → izvršavanje upita → zatvaranje)

Naredba `with` omogućava automatsko upravljanje resursima, osiguravajući da će se resursi pravilno osloboditi i zatvoriti čak i ako dođe do greške u bloku koda. Na taj način, kod postaje čišći i sigurniji.

Sintaksa naredbe `with`:

```
with neki_resurs as alias:
    # rad s resursom koristeći "alias"
```

Tipičan primjer korištenje naredbe `with` je rad s datotekama:

```
with open("datoteka.txt", "r") as file: # otvaramo datoteku za čitanje i koristimo alias
"file"
    sadržaj = file.read() # čitamo sadržaj datoteke
    print(sadržaj)
```

Bez korištenja naredbe `with`, morali bismo eksplicitno zatvoriti datoteku nakon što smo pročitali sadržaj:

```
file = open("datoteka.txt", "r")
sadržaj = file.read()
print(sadržaj)
file.close() # zatvaramo datoteku
```

Međutim kod iznad ne obuhvaća slučaj greške prilikom čitanja ili pisanja u datoteku ako postoji. U tom slučaju, trebali bismo koristiti `try-except-finally` blokove kako bismo osigurali da će se datoteka zatvoriti čak i ako dođe do greške.

```
try:
  file = open("datoteka.txt", "r")
  sadržaj = file.read()
  print(sadržaj)
except Exception as e:
  print(f"Greška: {e}")
finally:
  file.close()
```

> Osim spomenutih primjera resursa, možemo definirati i vlastite kontekstne menadžere, međutim to nije predmet ove skripte.

Naredba `with` **automatski zatvara resurs čak i ako dođe do greške u bloku koda**, što je jedan od razloga zašto se preporučuje njeno korištenje.

Dodatno, vidimo da je kod s naredbom `with` **kraći** i **lakši za čitanje**.

2.2.2 `ClientSession` klasa

Vratimo se na naš primjer slanja 15 zahtjeva na *Cat Facts API*. Što je ovdje resurs koji trebamo otvoriti i zatvoriti, u kontekstu `with` naredbe?

▶ Spoiler alert! Odgovor na pitanje

> U `aiohttp` biblioteci, za rad s HTTP sesijom koristimo klasu `ClientSession`.

Klasa `ClientSession` predstavlja asinkroni HTTP klijent koji omogućuje **konkurentno slanje HTTP zahtjeva unutar Python programa**. Ovaj klijent implementiran je kao kontekstni menadžer, što znači da ga možemo koristiti unutar `with` bloka.

- Kako bismo stvorili novu instancu `ClientSession` klase, kao i klase općenito, jednostavno pozivamo njen konstruktor:

U varijablu `session` spremamo instancu klase `ClientSession`:

```
session = aiohttp.ClientSession()
```

Nakon što smo stvorili instancu klase, možemo koristiti `with` blok kako bismo definirali blok koda za asinkroni rad s HTTP sesijom. Jedina razlika je što sad stvari radimo asinkrono pa moramo koristiti `async` ispred kontekstnog menadžera.

- Obzirom da koristimo `with`, možemo definirati alias `session` za **instancu unutar same naredbe**:

```
async with aiohttp.ClientSession() as session:
    # rad s HTTP sesijom
```

- Nad našom sesijom `session` sad možemo koristiti metodu `get` za slanje GET zahtjeva na isti način kao što smo to radili s `requests` bibliotekom:

```
async with aiohttp.ClientSession() as session:
  response = await session.get("https://catfact.ninja/fact")
  print(response)
```

Kako ovo sad pozvati? ***Context manager* sam po sebi nije funkcija, niti korutina**. Zato ga moramo pozvati unutar `async` funkcije ili korutine.

- jednostavno ga dodajemo unutar `main` korutine

```
async def main(): # definiramo main korutinu
  async with aiohttp.ClientSession() as session: # otvaramo HTTP sesiju koristeći context manager "with"
    response = await session.get("https://catfact.ninja/fact")
    print(response)

# pokrećemo main korutinu koristeći asyncio.run()
asyncio.run(main())
```

- Ako pokrenete kod vidjet ćete ogroman ispis, to je zato što smo ispisali cijeli HTTP odgovor, uključujući zaglavlja, statusnu liniju, tijelo itd...

Kako bismo dobili samo tijelo odgovora, možemo na isti način kao i kod `requests` biblioteke koristiti metodu `json()` za deserijalizaciju, ali s jednom razlikom - moramo koristiti `await` ključnu riječ jer je metoda sada asinkrona:

```
async def main():
  async with aiohttp.ClientSession() as session:
    response = await session.get("https://catfact.ninja/fact")
    fact_dict = await response.json() # dodajemo await
    print(fact_dict) # ispisuje nasumičnu činjenicu
```

OK, **sada znamo kako poslati jedan zahtjev asinkrono**. Vidimo da se trajanje nije promijenilo, ali to je zato što smo poslali samo jedan zahtjev.

Idemo poslati 5 zahtjeva na ovaj način, jednostavno ćemo kod iterirati 5 puta.

```
async def main():
  async with aiohttp.ClientSession() as session:
    for i in range(5):
      response = await session.get("https://catfact.ninja/fact")
      fact_dict = await response.json()
      print(fact_dict)
```

Trebali biste uočiti da stvari rade nešto brže nego prije. Jedino što može biti zbunjujuće je `print` naredba koja ispisuje činjenice sekvencijalno, ali.. **ispisuju li se one uopće sekvencijalno? Kako to možemo znati ako su činjenice nasumične?**

U ispis ćemo dodati vrijednost lokalne varijable `i` kako bismo vidjeli redoslijed ispisivanja činjenica:

```
async def main():
  async with aiohttp.ClientSession() as session:
    for i in range(5):
      response = await session.get("https://catfact.ninja/fact")
      fact_dict = await response.json()
      print(f"{i + 1}: {fact_dict["fact"]}")

asyncio.run(main())
```

Primjer ispisa:

```
1: Cats' hearing is much more sensitive than humans and dogs.
2: A cat named Dusty, aged 1 7, living in Bonham, Texas, USA, gave birth to her 420th
kitten on June 23, 1952.
3: British cat owners spend roughly 550 million pounds yearly on cat food.
4: There are more than 500 million domestic cats in the world, with approximately 40
recognized breeds.
5: A cat's appetite is the barometer of its health. Any cat that does not eat or drink for
more than two days should be taken to a vet.
```

> **VAŽNO!** Ako pokrenete kod više puta, uočit ćete da se činjenice uvijek ispisuju sekvencijalno, odnosno vrijednost `i` je uvijek: `1 2 3 4 5`.
>
> - Iako je kod iznad tehnički napisan "asinkrono", ova petlja se ustvari **ne izvršava konkurentno** budući da koristimo `await` ključnu riječ ispred svakog slanja zahtjeva `session.get`
> - Zbog toga ova petlja sama po sebi **neće slati zahtjeve konkurentno**, već će svaki zahtjev čekati na odgovor prethodnog, što znači da se svaka iteracija petlje izvršava sekvencijalno (jedna za drugom).

Vidjeli smo sličan problem u prošloj skripti gdje smo simulirali slanje zahtjeva na poslužitelj koristeći `asyncio.sleep` funkciju te smo rekli da ga možemo riješiti na 3 načina:

- koristeći `asyncio.gather` funkciju
- koristeći `asyncio.Task` objekte
- kombiniranjem ove dvije metode

2.2.3 Konkurentno slanje kroz `asyncio.gather`

U prošloj skripti ste naučili da možemo koristiti `asyncio.gather` funkciju kako bismo **pozvali više korutina konkurentno** i **zatim pohraniti sve rezultate u jednu listu**.

Sintaksa `asyncio.gather`:

```
asyncio.gather(*korutine) # ako su unutar liste

asyncio.gather(korutina1, korutina2, korutina3) # ako su pojedinačno
```

Kako ćemo ovdje definirati korutinu za slanje?

Ideja je da iz sljedeće `main` korutine izvučemo kod za slanje zahtjeva u zasebnu korutinu `get_cat_fact`:

```
async def main():
  async with aiohttp.ClientSession() as session:
    for i in range(5):
      response = await session.get("https://catfact.ninja/fact")
      fact_dict = await response.json()
      print(f"{i + 1}: {fact_dict["fact"]}")

asyncio.run(main())
```

Glavno pitanje je **gdje ćemo definirati *context manager***? U `main` korutini ili u `get_cat_fact` korutini?

▶ Spoiler alert! Odgovor na pitanje

U vanjsku korutinu prosljeđujemo alias `session`:

```
async def get_cat_fact(session):
  response = await session.get("https://catfact.ninja/fact")
  fact_dict = await response.json()
  return fact_dict
```

U `main` korutini tada moramo definirati otvaranje same sesije:

```
async def main():
  async with aiohttp.ClientSession() as session:
```

Napokon, možemo koristiti `asyncio.gather` funkciju kako bismo poslali 5 zahtjeva konkurentno. Kako već znamo dobro *comprehension* sintaksu, iskoristit ćemo *list comprehension* za izradu liste korutina:

```
async def get_cat_fact(session):
  response = await session.get("https://catfact.ninja/fact")
  fact_dict = await response.json()
  print(fact_dict['fact'])

async def main():
  async with aiohttp.ClientSession() as session:
    cat_fact_korutine = [get_cat_fact(session) for i in range(5)]
```

- **Pozivamo korutine konkurentno** koristeći `asyncio.gather` funkciju

```
async def main():
  async with aiohttp.ClientSession() as session:
    cat_fact_korutine = [get_cat_fact(session) for i in range(5)]
    await asyncio.gather(*cat_fact_korutine)
```

> Pokrenite kod - vidimo da se činjenice ispisuju dosta brzo.

```
A kitten will typically weigh about 3 ounces at birth.  The typical male housecat will
weigh between  7 and 9 pounds, slightly less for female housecats.
Cats see six times better in the dark and at night than humans.
There are approximately 60,000 hairs per square inch on the back of a cat and about
120,000 per square inch on its underside.
Cats bury their feces to cover their trails from predators.
The Egyptian Mau is probably the oldest breed of cat. In fact, the breed is so ancient
that its name is the Egyptian word for “cat.”
```

Ako se prisjetite, prosječno vrijeme trajanja programa s 5 činjenica je bilo 1-2 sekunde, ali tada smo imali i ispisivanje: `print("Šaljemo n. zahtjev...")` u svakoj iteraciji.

Dodat ćemo i ovdje `print` naredbu prije ispisa činjenice i izmjeriti vrijeme koristeći `time` modul:

```
async def get_cat_fact(session):
  print("Šaljemo zahtjev za mačji fact")
  response = await session.get("https://catfact.ninja/fact")
  fact_dict = await response.json()
  print(fact_dict['fact'])
```

I bez dodavanja `time` modula, odmah vidimo razliku u terminalu! Prije smo imali **sekvencijalno slanje zahtjeva po zahtjev**:

Sekvencijalno slanje (*requests*):

```
Šaljemo 1. zahtjev...
Cats often overract to unexpected stimuli because of their extremely sensitive nervous
system.
Šaljemo 2. zahtjev...
The normal body temperature of a cat is between 100.5 ° and 102.5 °F. A cat is sick if its
temperature goes below 100 ° or above 103 °F.
Šaljemo 3. zahtjev...
If they have ample water, cats can tolerate temperatures up to 133 °F.
Šaljemo 4. zahtjev...
Cats don’t have sweat glands over their bodies like humans do. Instead, they sweat only
through their paws.
Šaljemo 5. zahtjev...
The first commercially cloned pet was a cat named "Little Nicky." He cost his owner
$50,000, making him one of the most expensive cats ever.
Izvršavanje programa traje 1.26 sekundi.
```

Sada vidimo da se svi zahtjevi prvo pošalju **konkurentno**, a zatim ispisuju sve činjenice. **Ne čekamo više na odgovor kroz svaku iteraciju petlje**.

Konkurentno slanje (*aiohttp*):

```
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Lions are the only cats that live in groups, called prides. Every female within the pride
is usually related.
A happy cat holds her tail high and steady.
The average cat food meal is the equivalent to about five mice.
The Egyptian Mau is probably the oldest breed of cat. In fact, the breed is so ancient
that its name is the Egyptian word for “cat.”
A cat’s nose pad is ridged with a unique pattern, just like the fingerprint of a human.
```

Zanima nas još i vrijeme izvođenja programa.

Započeti ćemo mjeriti kad se pozove `main` korutina, a završiti na kraju `main` korutine.

```python
async def main():
  start = time.time()
  async with aiohttp.ClientSession() as session:
      cat_fact_korutine = [get_cat_fact(session) for i in range(5)]
      await asyncio.gather(*cat_fact_korutine)
  end = time.time()
  print(f"\nIzvršavanje programa traje {end - start:.2f} sekundi.")

asyncio.run(main())
```

Primjer ispisa:

```
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Cats have "nine lives" thanks to a flexible spine and powerful leg and back muscles
Cats' eyes shine in the dark because of the tapetum, a reflective layer in the eye, which
acts like a mirror.
The oldest cat on record was Crème Puff from Austin, Texas, who lived from 1967 to August
6, 2005, three days after her 38th birthday. A cat typically can live up to 20 years,
which is equivalent to about 96 human years.
When a cats rubs up against you, the cat is marking you with it's scent claiming
ownership.
Cats see six times better in the dark and at night than humans.

Izvršavanje programa traje 0.27 sekundi.
```

Vidimo da se vrijeme izvršavanja programa na ovom jednostavnom primjeru slanja 5 zahtjeva **smanjilo s ~1.26 sekundi na ~0.27 sekundi**.

Razliku možemo izraziti i u postocima:

$$\frac{sekvencijalnoVrijeme - konkurentnoVrijeme}{sekvencijalnoVrijeme} \times 100 \tag{1}$$

odnosno:

$$\frac{1.26 - 0.27}{1.26} \times 100 \approx 78.57\% \tag{2}$$

Dakle, **konkurentni kod se izvršio otprilike 78.57% brže od sekvencijalnog**!

Ako podijelimo staro vrijeme izvršavanja s novim, vidimo da je **konkurentni kod gotovo 5 puta brži od sekvencijalnog**.

$$\frac{sekvencijalnoVrijeme}{konkurentnoVrijeme} \tag{3}$$

$$\frac{1.26}{0.27} = 4.67 \tag{4}$$

Pokušajmo i s 15 zahtjeva:

```
async def main():
  start = time.time()
  async with aiohttp.ClientSession() as session:
      cat_fact_korutine = [get_cat_fact(session) for i in range(15)]
      await asyncio.gather(*cat_fact_korutine)
  end = time.time()
  print(f"\nIzvršavanje programa traje {end - start:.2f} sekundi.")
```

Primjer ispisa:

```
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Cat families usually play best in even numbers. Cats and kittens should be acquired in
pairs whenever possible.
Cats are subject to gum disease and to dental caries. They should have their teeth cleaned
by the vet or the cat dentist once a year.
The biggest wildcat today is the Siberian Tiger. It can be more than 12 feet (3.6 m) long
(about the size of a small car) and weigh up to 700 pounds (317 kg).
A group of cats is called a clowder.
The heaviest cat on record is Himmy, a Tabby from Queensland, Australia. He weighed nearly
47 pounds (21 kg). He died at the age of 10.
A cat can jump up to five times its own height in a single bound.
A commemorative tower was built in Scotland for a cat named Towser, who caught nearly
30,000 mice in her lifetime.
Purring does not always indicate that a cat is happy and healthy - some cats will purr
loudly when they are terrified or in pain.
Long, muscular hind legs enable snow leopards to leap seven times their own body length in
a single bound.
The most traveled cat is Hamlet, who escaped from his carrier while on a flight. He hid
for seven weeks behind a panel on the airplane. By the time he was discovered, he had
traveled nearly 373,000 miles (600,000 km).
Cats and kittens should be acquired in pairs whenever possible as cat families interact
best in pairs.
```

```
The earliest ancestor of the modern cat lived about 30 million years ago. Scientists
called it the Proailurus, which means “first cat” in Greek. The group of animals that pet
cats belong to emerged around 12 million years ago.
There are up to 60 million feral cats in the United States alone.
The strongest climber among the big cats, a leopard can carry prey twice its weight up a
tree.
The name "jaguar" comes from a Native American word meaning "he who kills with one leap".

Izvršavanje programa traje 0.61 sekundi.
```

> Vidimo da se vrijeme izvršavanja programa s 15 zahtjeva **smanjilo s 3-4 sekunde na 0.61 sekundi**. Ovdje nam je također program gotovo 5 puta brži, odnosno poboljšanje je ~80%.

2.2.4 Konkurentno slanje kroz `asyncio.Task`

Naučili smo kako koristiti `asyncio.gather` funkciju za konkurentno izvođenje korutina. Međutim, u prošloj skripti smo rekli da možemo definirati i tzv. **Taskove** koji predstavljaju izvršenje korutina unutar `asyncio` petlje.

Rekli smo da `Task` objekti, (možemo ih zvati i *Taskovi*) omogućuju bolju kontrolu nad izvršavanjem korutina jer možemo pratiti njihov status, upravljati njima pojedinačno, i eventualno čekati da završe pomoću `await` ključne riječi.

Taskove definiramo koristeći `asyncio.create_task` funkciju koja prima korutinu kao argument.

Sintaksa:

```
task = asyncio.create_task(korutina)
```

U našem primjeru dohvaćanja činjenica o mačkama, korutine su `get_cat_fact`. Možemo ih jednostavno pohraniti u listu i zatim izraditi `Task` objekte za svaku, koristeći *list comprehension*.

Nakon toga ćemo ih pozvati koristeći `await` ključnu riječ jednostavnim iteriranjem kroz listu `Task` objekata.

```
async def main():
  start = time.time()
  async with aiohttp.ClientSession() as session:
      cat_fact_tasks = [asyncio.create_task(get_cat_fact(session)) for _ in range(5)] #
pohranjujemo Taskove u listu
      for task in cat_fact_tasks: # ovaj kod izvršava se konkurentno jer smo koristili
Taskove
          await task # pozivamo svaki Task
  end = time.time()
  print(f"\nIzvršavanje programa traje {end - start:.2f} sekundi.")
```

Rezultat je identičan kao i kod `asyncio.gather` funkcije:

```
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
Šaljemo zahtjev za mačji fact
70% of your cat's life is spent asleep.
Cats eat grass to aid their digestion and to help them get rid of any fur in their
stomachs.
In 1987 cats overtook dogs as the number one pet in America.
In ancient Egypt, when a family cat died, all family members would shave their eyebrows as
a sign of mourning.
A cat can’t climb head first down a tree because every claw on a cat’s paw points the same
way. To get down from a tree, a cat must back down.

Izvršavanje programa traje 0.28 sekundi.
```

> Ako izvrtimo kod više puta, vidjet ćete da je rezultat izvođenja u pravilu identičan (~0,27 sekundi) kao što je to bio slučaj s `asyncio.gather` funkcijom.

Rekli smo da je moguće i kombinirati ova dva pristupa, odnosno koristiti `asyncio.gather` funkciju za konkurentno izvođenje *Taskova*:

```
async def main():
  start = time.time()
  async with aiohttp.ClientSession() as session:
    cat_fact_tasks = [asyncio.create_task(get_cat_fact(session)) for _ in range(5)] #
pohranjujemo Taskove u listu
    await asyncio.gather(*cat_fact_tasks) # pozivamo Taskove konkurentno
  end = time.time()
  print(f"\nIzvršavanje programa traje {end - start:.2f} sekundi.")
```

Možemo maknuti `print` naredbe unutar korutine `get_cat_fact` te vratiti samo činjenicu kao rezultat te korutine:

```
async def get_cat_fact(session):
  response = await session.get("https://catfact.ninja/fact")
  fact_dict = await response.json()
  return fact_dict['fact']

async def main():
  start = time.time()
  async with aiohttp.ClientSession() as session:
    cat_fact_tasks = [asyncio.create_task(get_cat_fact(session)) for _ in range(5)] #
pohranjujemo Taskove u listu
    actual_cat_facts = await asyncio.gather(*cat_fact_tasks) # pohranit ćemo rezultate u
listu
  end = time.time()
  print(actual_cat_facts)
  print(f"\nIzvršavanje programa traje {end - start:.2f} sekundi.")

asyncio.run(main())
```

Rezultat je lista činjenica:

```
['The Maine Coone is the only native American long haired breed.', 'The Amur leopard is
one of the most endangered animals in the world.', "A cat's normal pulse is 140-240 beats
per minute, with an average of 195.", 'The cat has 500 skeletal muscles (humans have
650).', 'A happy cat holds her tail high and steady.']

Izvršavanje programa traje 0.27 sekundi.
```

U slučaju da nam ispisi i vrijeme izvođenja nisu dovoljan dokaz da su zahtjevi uistinu poslani konkurentno, možemo još provjeriti **redoslijed ispisivanja činjenica** koji nam je, ako se prisjetite, kod sekvencijalnog slanja uvijek bio isti: `1 2 3 4 5`.

Ovdje to možemo testirati na način da ćemo jednostavno proslijediti `i` lokalnu varijablu iz petlje u korutinu `get_cat_fact`:

```
async def get_cat_fact(session, i):
  response = await session.get("https://catfact.ninja/fact")
  fact_dict = await response.json()
  print(f"{i + 1}: {fact_dict['fact']}") # dodajemo ispis u formatu: "redniBroj:
činjenica"
  return fact_dict['fact']

async def main():
  start = time.time()
  async with aiohttp.ClientSession() as session:
    cat_fact_tasks = [asyncio.create_task(get_cat_fact(session, i)) for i in range(5)] # u
korutinu prosljeđujemo "i"
    actual_cat_facts = await asyncio.gather(*cat_fact_tasks)
  end = time.time()
  print(f"\nIzvršavanje programa traje {end - start:.2f} sekundi.")
asyncio.run(main())
```

Primjer ispisa 1:

```
2: It is estimated that cats can make over 60 different sounds.
1: According to a Gallup poll, most American pet owners obtain their cats by adopting
strays.
5: Cats are the world's most popular pets, outnumbering dogs by as many as three to one
3: The oldest cat to give birth was Kitty who, at the age of 30, gave birth to two
kittens. During her life, she gave birth to 218 kittens.
4: Cats can jump up to 7 times their tail length.
```

Primjer ispisa 2:

```
1: In Japan, cats are thought to have the power to turn into super spirits when they die.
This may be because according to the Buddhist religion, the body of the cat is the
temporary resting place of very spiritual people.i
4: A cat sees about 6 times better than a human at night, and needs 1/6 the amount of of
light that a human does - it has a layer of extra reflecting cells which absorb light.
3: Cats lived with soldiers in trenches, where they killed mice during World War I.
5: In relation to their body size, cats have the largest eyes of any mammal.
2: Female felines are \superfecund
```

Primjer ispisa 3:

```
4: Mountain lions are strong jumpers, thanks to muscular hind legs that are longer than
their front legs.
2: Cats' hearing stops at 65 khz (kilohertz); humans' hearing stops at 20 khz.
3: Retractable claws are a physical phenomenon that sets cats apart from the rest of the
animal kingdom. I n the cat family, only cheetahs cannot retract their claws.
5: A cat uses its whiskers for measuring distances.  The whiskers of a cat are capable of
registering very small changes in air pressure.
1: Tylenol and chocolate are both poisionous to cats.
```

Ako se prisjetimo ilustracije konkurentnog izvođenja na samom početku skripte `RS3`, da se zaključiti zašto su rezultati ovakvi.

Naime, **svaki zahtjev se šalje u isto vrijeme raspodjelom računalnih resursa unutar event loopa**, a **odgovori se vraćaju u različito vrijeme**. Zbog toga je redoslijed ispisivanja činjenica **nasumičan**. Pitanje je, kako upravljati ovakvim nasumičnim ponašanjem? Za sada ćemo ostaviti ovu temu otvorenom.

> Sada definitivno možemo reći da je kod koji smo definirali **konkurentan** te možemo uočiti **jasna poboljšanja u brzini izvođenja programa** 🚀

3. Zadaci za vježbu - Slanje konkurentnih HTTP zahtjeva

1. **Definirajte korutinu** `fetch_users` koja će slati GET zahtjev na JSONPlaceholder API na URL-u: `https://jsonplaceholder.typicode.com/users`. Morate simulirate slanje 5 zahtjeva konkurentno unutar `main` korutine. Unutar `main` korutine izmjerite vrijeme izvođenja programa, a rezultate pohranite u listu odjedanput koristeći `asyncio.gather` funkciju. Nakon toga koristeći `map` funkcije ili *list comprehension* izdvojite u zasebne 3 liste: samo imena korisnika, samo email adrese korisnika i samo username korisnika. Na kraju `main` korutine ispišite sve 3 liste i vrijeme izvođenja programa.

2. **Definirajte dvije korutine**, od kojih će jedna služiti za dohvaćanje činjenica o mačkama koristeći `get_cat_fact` korutinu koja šalje GET zahtjev na URL: `https://catfact.ninja/fact`. Izradite 20 `Task` objekata za dohvaćanje činjenica o mačkama te ih pozovite unutar `main` korutine i rezultate pohranite odjednom koristeći `asyncio.gather` funkciju. Druga korutina `filter_cat_facts` ne šalje HTTP zahtjeve, već mora primiti gotovu listu činjenica o mačkama i vratiti novu listu koja sadrži samo one činjenice koje sadrže riječ "cat" ili "cats" (neovisno o velikim/malim slovima).

Primjer konačnog ispisa:

```
Filtrirane činjenice o mačkama:
- A 2007 Gallup poll revealed that both men and women were equally likely to own a cat.
- The first cat in space was a French cat named Felicette (a.k.a. “Astrocat”) In 1963, France blasted the cat into outer space. Electrodes implanted in her brains sent neurological signals back to Earth. She survived the trip.
- The lightest cat on record is a blue point Himalayan called Tinker Toy, who weighed 1 pound, 6 ounces (616 g). Tinker Toy was 2.75 inches (7 cm) tall and 7.5 inches (19 cm) long.
- The first commercially cloned pet was a cat named "Little Nicky." He cost his owner $50,000, making him one of the most expensive cats ever.
- In the 1750s, Europeans introduced cats into the Americas to control pests.
- A group of cats is called a clowder.
```

3. **Definirajte korutinu** `get_dog_fact` koja dohvaća činjenice o psima sa DOG API.

Korutina `get_dog_fact` neka dohvaća činjenicu o psima na URL-u: `https://dogapi.dog/api/v2/facts`. Nakon toga, **definirajte korutinu** `get_cat_fact` koja dohvaća činjenicu o mačkama slanjem zahtjeva na URL: `https://catfact.ninja/fact`.

Istovremeno pohranite rezultate izvršavanja ovih *Taskova* koristeći `asyncio.gather(*dog_facts_tasks, *cat_facts_tasks)` funkciju u listu `dog_cat_facts`, a zatim ih koristeći *list slicing* odvojite u dvije liste obzirom da znate da je prvih 5 činjenica o psima, a drugih 5 o mačkama.

Na kraju, definirajte i **treću korutinu** `mix_facts` koja prima liste `dog_facts` i `cat_facts` i vraća **novu listu** koja za vrijednost indeksa `i` sadrži činjenicu o psima ako je duljina činjenice o psima veća od duljine činjenice o mačkama na istom indeksu, inače vraća činjenicu o mački. Na kraju ispišite rezultate filtriranog niza činjenica. Liste možete paralelno iterirati koristeći `zip` funkciju, npr. `for dog_fact, cat_fact in zip(dog_facts, cat_facts)`.

Primjer konačnog ispisa:

```
Mixane činjenice o psima i mačkama:

If they have ample water, cats can tolerate temperatures up to 133 °F.
Dogs with little human contact in the first three months typically don’t make good pets.
The most popular dog breed in Canada, U.S., and Great Britain is the Labrador retriever.
An estimated 1,000,000 dogs in the U.S. have been named as the primary beneficiaries in
their owner’s will.
When a cats rubs up against you, the cat is marking you with it's scent claiming
ownership.
```