FIPU: rs-mid - Službeni šalabahter

Funkcije višeg reda

```
map(fn, iterables) # primjenjuje 'fn' na svaki element iz 'iterables'.
filter(fn, iterables) # filtrira elemente 'iterables' prema predikatu 'fn'.
any(iterables) # vraća True ako je bilo koji element 'iterables' True
all(iterables) # vraća True ako su svi elementi 'iterables' True
```

Comprehension sintaksa

```
[expression for item in iterable] # list comp
[expression for item in iterable if condition] # list comp s if
{key_expression: value_expression for item in iterable} # dict comp
{key_expression: value_expression for item in iterable if condition} # dict comp s if
[expression1 if condition else expression2 for element in iterable] # list comp s if-else
{key_expression: value_expression1 if condition else value_expression2 for item in
iterable} # dict comp s if-else
[expression for item_1, item_2 in zip(list1, list2)] # list comp s više iterables
```

Klase i objekti

```
class SomeClass:
  def __init__(self, argument_1_value, argument_2_value):
    self.argument_1 = argument_1_value
    self.argument_2 = argument_2_value
  def some_method(self):
    return self.argument_1 + self.argument_2
instance = SomeClass(argument_1_value, argument_2_value)
instance.some_method()
```

asyncio

```
async def main(): # main korutina (asinkrona funkcija)
  pass
asyncio.run(main()) # pokretanje korutine
asyncio.sleep(sec) # čekanje u korutini
await coroutine() # sekvencijalno izvršavanje korutine
asyncio.gather(coroutine1, coroutine2, coroutine3...) # konkurentno izvršavanje
asyncio.gather(*lista_korutina) # konkurentno izvršavanje s *args
asyncio.create_task(coroutine) # izrada Taska
await asyncio.create_task(coroutine) # konkurentno izvršavanje s Task
```

aiohttp.ClientSession

```
async with aiohttp.ClientSession() as session: # izrada klijentske sesije kroz context manager
  response = await session.get(url) # slanje GET zahtjeva
  response_data = await response.json() # deserijalizacija JSON odgovora
  response = await session.post(url, json=data) # slanje POST zahtjeva s JSON podacima u tijelu zahtjeva
  response_data = await response.json() # deserijalizacija JSON odgovora
  response.status # statusni kod odgovora
# mjerenje vremena izvršavanja programa
start = time.time()
# kod koji se mjeri
end = time.time()
print(f"Izvršavanje je trajalo {end - start:.2f} sekundi.")
```

aiohttp.web

```
from aiohttp import web
app = web.Application()
app.router.add_get(path, handler_function) # GET zahtjev, path= '/resurs/{parameter}' za URL parametar
app.router.add_post(path, handler_function) # POST zahtjev (analogno za ostale HTTP metode)
web.run_app(app, port=PORT) # blocking pokretanje poslužitelja

async def handler_function(request) # korutina koja obrađuje zahtjev
  vrijednost_route_parametra = request.match_info['parameter'] # dohvaćanje URL param; /resurs/{parameter}
  vrijednost_query_parametra = request.query['parameter'] # dohvaćanje query param; /resurs?parameter=vrijednost
  request_data = await request.json() # dohvaćanje JSON podataka iz zahtjeva
  return web.json_response(response_data, status = 200) # slanje JSON odgovora s status kodom
```

aiohttp.AppRunner

```
async def start_server():
  runner = AppRunner(app) # instanciranje AppRunner instance
  await runner.setup() # pokretanje AppRunner instance
  site = web.TCPSite(runner, host, port) # registracija poslužitelja
  await site.start() # pokretanje poslužitelja
async def main():
  await start_server() # non-blocking pokretanje poslužitelja
```