



Distance Learning System

# Async IO

Python Network Programming

# Šta je asyncio

---

- API/modul za konkurentno programiranje
- Omogućava jednonitni konkurentan rad
- Omogućava multipleksing ulaza/izlaza
- Zahteva manje resursa u odnosu na niti

```
import asyncio
```

# Korutine

---

- Korutina je posebna vrsta funkcije čije se izvršavanje može kontrolisati izvan nje
- Korutina se ne može startovati eksplicitno već metodom **run** modula **asyncio**
- Korutina se označava prefiksom **async**

```
async def f():  
    print("Hello")  
  
asyncio.run(f())
```

# await i awaitable

---

- Ključna reč **await** se nalazi unutar korutine, ispred poziva korutine (awaitable objekta)
- Await omogućava eksplicitan poziv **awaitable** objekta (**korutina**, **zadatak** ili **future**)

```
async def mycor():  
    print("Hello")  
    await asyncio.sleep(5)  
    print("World")  
  
asyncio.run(mycor())
```

# Zadaci (task-ovi)

- Zadaci su zapakovane korutine koje se mogu izvršavati paralelno

Specijalna (neblokirana)  
varijanta sleep metode

Startuje se u istom  
trenutku

Čeka se završetak  
taskova

```
async def f(t):  
    print(f"Hello {t}")  
    await asyncio.sleep(t)  
    print(f"World {t}")  
  
async def f1():  
    task1 = asyncio.create_task(f(3))  
    task2 = asyncio.create_task(f(5))  
    await task2  
    await task1  
  
asyncio.run(f1())
```

# Future

---

- Awaitable objekti koji mogu, ali ne moraju imati odgovor (vrednost)

```
import asyncio
import time

async def f(fut):
    fut.set_result("You don't have future my son!")

async def f1():
    loop = asyncio.get_event_loop()
    future = loop.create_future()
    asyncio.create_task(f(future))
    print(await future)

asyncio.run(f1())
```

# Grupisanje awaitable objekata

- Awaitable objekti se mogu grupisati i izvršiti asinhrono

```
import asyncio
import random

async def f(taskid):
    print(f"Task: {taskid}")
    sleeptime = random.randint(1,5)
    print(f"Task {taskid} will sleep {sleeptime} seconds")
    for i in range(sleeptime):
        print(f"Task {taskid}: {sleeptime-i} seconds left")
        await asyncio.sleep(1)
    print(f"Task {taskid} finished")

async def main():
    await asyncio.gather(
        f(1), f(2), f(3), f(4)
    )

asyncio.run(main())
```

# Preuzimanje rezultata

---

- Awaitable objekat je rezultat awaitable funkcije
- Da bi se od ovakve funkcije uzeo rezultat, placeholderi se moraju postaviti iza ključne reči await

```
async def f1():  
    return 10,20  
  
async def main():  
    x,y = await f1()  
    print(x,y)  
  
asyncio.run(main())
```



# Event loop

<https://docs.python.org/3/library/asyncio-eventloop.html>

- Osim automatskig upravljanja petljom događaja (aktivirane metodom `asyncio.run()`), moguće je i ručno upravljanje njome, pomoću `asyncio` metoda: **`get_running_loop`**, **`get_event_loop`**, **`run_until_complete`**, **`run_forever`**, **`stop`** i drugih.

```
async def main():  
    print("Hello")  
    await asyncio.sleep(5)  
    print("World")  
  
loop = asyncio.get_event_loop()  
loop.run_until_complete(main())  
loop.run_forever()
```