**SPAC uge 5 - Pensum**

**Ugens emme:**

* Multi Threading & PDF downloader

**Læringsmål:**

* Forstå begreberne multi-threading og dens anvendelse i Python
* Få viden om HTTP-anmodninger og deres implementering i Python
* Lær om SCRUM- og Kanban-metoder til projektledelse

**Aflevering/præsentation:**

* Du skal i denne uge aflevere dit projekt.
* Afleveringssted: Github og sende linket til din repository, i Teams (gem linket i en tekstfil og upload til Teams), så vi kan tilgå dit projekt.
* Afleveringstidspunkt: d. 08/11/2024/, kl. 15.00

**Ugeopgave:**

Denne uge fokuserer på vigtige teoretiske koncepter, der er nødvendige for at skabe software, der arbejder effektivt med flere samtidige opgaver, kommunikerer med webservere, og kan struktureres effektivt ved hjælp af agile projektledelsesmetoder som SCRUM.

Ugens læringsmål omfatter:

1. **Multi-threading**: Forstå forskellige metoder til at håndtere samtidighed i Python, herunder brug af traditionelle threads, thread pools, og asynkrone løsninger med asyncio.
2. **HTTP**: Forstå grundlaget for HTTP-anmodninger, såsom GET- og POST-metoder, samt hvordan man håndterer disse i Python ved hjælp af requests-modulet.
3. **SCRUM og Kanban**: Lær at planlægge og udføre sprints ved brug af SCRUM-metodologi, og hvordan man bruger et Kanban-board til at følge opgavernes fremskridt.

# **Multi-threading**

## **Hvad er Multi-threading?**

Multi-threading i Python er en teknik, der muliggør udførelse af flere opgaver parallelt ved at oprette flere tråde, der kører samtidig inden for et enkelt program. Dette er særligt nyttigt, når der arbejdes med I/O-tunge opgaver som netværksanmodninger eller filhåndtering, da det reducerer ventetid og forbedrer ydeevnen ved at udføre flere opgaver parallelt.

I et multi-threading miljø kan hver tråd køre uafhængigt af hinanden, men de deler samme hukommelse, hvilket kan føre til problemer såsom "race conditions", hvor flere tråde forsøger at skrive til den samme variabel på samme tid. Python’s Global Interpreter Lock (GIL) forhindrer dog visse race conditions, men begrænser også ægte parallel udførelse af CPU-bundne opgaver.

## **Direkte brug af Multi-threading**

For at oprette og arbejde med tråde i Python kan du bruge threading-modulet, der gør det muligt at oprette nye tråde, hvor hver tråd udfører en specifik opgave parallelt. Et simpelt eksempel ser sådan ud:

**import** threading

**import** time

**import** random

**def** task**(**arg**):**

**print(**f"Starting task with arg: {arg}"**)**

# Simulerer en arbejdsopgave

time**.**sleep**(**random**.**randint**(**1**,**5**))**

**print(**f"Completed task"**)**

# Opret flere tråde

threads **=** **[]**

args **=** **[**"argument 1"**,** "argument 2"**]**

**for** args **in** args**:**

thread **=** threading**.**Thread**(**target**=**task**,** args**=(**args**,))**

threads**.**append**(**thread**)**

thread**.**start**()**

# Vent på at alle tråde er færdige

**for** thread **in** threads**:**

thread**.**join**()**

Her oprettes en tråd for hver arbejdsopgave der skal udføres, og vi bruger thread.join() til at sikre, at hovedtråden venter, indtil alle opgaver er færdige.

## **Thread Pools**

Thread pools er en effektiv måde at administrere tråde på, da de genbruger eksisterende tråde fremfor at oprette nye tråde for hver opgave, hvilket kan reducere overheaden. Dette er ideelt, når du har mange små opgaver, der kan køre parallelt.

Python giver dig mulighed for at bruge concurrent.futures.ThreadPoolExecutor til at implementere en thread pool:

**from** concurrent**.**futures **import** ThreadPoolExecutor

**import** time

**import** random

**def** task**(**arg**):**

**print(**f"Starting task with arg: {arg}"**)**

# Simulerer en arbejdsopgave

time**.**sleep**(**random**.**randint**(**1**,** 5**))**

**print(**f"Completed task"**)**

args **=** **[**"argument 1"**,** "argument 2"**]**

**with** ThreadPoolExecutor**()** **as** executor**:**

executor**.map(**task**,** args**)**

I dette eksempel bliver ThreadPoolExecutor brugt til at oprette en pulje af tråde, der kan håndtere flere opgaver på én gang. Det reducerer belastningen ved at administrere trådenes livscyklus og er især effektivt, når du arbejder med mange samtidige opgaver.

## **Asyncio**

Asyncio er et bibliotek i Python, der giver dig mulighed for at skrive asynkron ikke-blokerende kode, hvilket er nyttigt, når du arbejder med I/O-bundne opgaver. I stedet for at oprette flere tråde, arbejder asyncio ved at lade programmet håndtere flere opgaver ved hjælp af såkaldt "cooperative multitasking". Dette betyder at asyncio reducerer overhead for flere tråde og kontekst skift.

### Flow of Control med await

Når du bruger async- og await-syntaksen i asyncio, fortæller du Python, at programmet skal vente på en asynkron funktion, før det fortsætter med eksekveringen. Forskellen fra multi-threading er, at mens programmet venter, kan tråden arbejde på andre opgaver, hvilket gør det muligt at håndtere mange opgaver uden at blokere.

Et simpelt eksempel med asyncio kunne se sådan ud:

**import** asyncio

**async** **def** task**(**arg**):**

**print(**f" Starting task with arg: {arg}"**)**

# Simulerer asynkron arbejdsopgave

**await** asyncio**.**sleep**(**random**.**randint**(**1**,** 5**))**

**print(**f"Completed task"**)**

**async** **def** main**():**

args **=** **[**"argument 1"**,** "argument 2"**]**

tasks **=** **[**task**(**url**)** **for** arg **in** args**]**

**await** asyncio**.**gather**(\***tasks**)**

asyncio**.**run**(**main**())**

I dette eksempel bruges await til at indikere, at funktionen skal vente på, at asyncio.sleep() er fuldført, uden at blokere resten af programmet. Dette tillader, at flere opgaver kan startes og fuldføres parallelt.

# **HTTP**

## **Forståelse af HTTP-protokollen**

HTTP (Hypertext Transfer Protocol) er en kommunikationsprotokol, der styrer dataudveksling mellem klienter (som browsere eller scripts) og servere på internettet. Når man arbejder med software, der kommunikerer med hjemmesider eller interagerer med webapplikationer, er det vigtigt at forstå de forskellige HTTP-metoder og statuskoder for effektiv fejlfinding og integration.

## HTTP Metoder:

* **GET**: Bruges til at anmode om data fra serveren uden at ændre serverens tilstand. GET-anmodninger bruges typisk til at hente filer eller hjemmesider. Data sendes som parametre i URL'en, hvilket gør GET mindre egnet til sensitive oplysninger, da de er synlige i URL'en.
* **POST**: Bruges til at sende data til serveren, som typisk vil ændre serverens tilstand. POST er velegnet til at uploade filer, indsende formularer eller foretage handlinger, der kræver ændringer på serveren. Data sendes som en del af anmodningens brødtekst og er ikke synlig i URL'en.
* **PUT**: Anvendes til at uploade eller opdatere en eksisterende ressource på serveren. I modsætning til POST er PUT idempotent, hvilket betyder, at det samme kald kan gentages uden at ændre resultatet.
* **DELETE**: Bruges til at slette en ressource på serveren. Som PUT er DELETE også idempotent, hvilket betyder, at gentagne anmodninger om at slette en ressource ikke fører til fejl, hvis ressourcen allerede er slettet.
* **HEAD**: Minder om GET, men uden at returnere selve brødteksten i svaret. Bruges ofte til at kontrollere, om en ressource findes, eller til at få metadata som længde og type uden at hente hele indholdet.

## HTTP Statuskoder:

Når en HTTP-anmodning sendes, returnerer serveren en statuskode, der angiver, hvordan anmodningen blev behandlet. Her er nogle af de vigtigste kategorier af statuskoder:

* **2xx (Succesfulde svar)**: Disse koder indikerer, at anmodningen blev behandlet med succes.
  + **200 OK**: Anmodningen blev gennemført, og svaret indeholder de ønskede data.
  + **201 Created**: Bruges ved succesfuld oprettelse af en ressource, typisk ved POST eller PUT.
* **3xx (Omdirigeringer)**: Bruges til at fortælle klienten, at den skal foretage en anden anmodning til en ny URL.
  + **301 Moved Permanently**: Ressourcen er permanent flyttet til en ny adresse.
  + **302 Found**: Ressourcen er midlertidigt tilgængelig på en anden adresse.
* **4xx (Klientfejl)**: Indikerer, at noget gik galt med klientens anmodning.
  + **400 Bad Request**: Serveren forstod ikke anmodningen pga. syntaksfejl.
  + **401 Unauthorized**: Klienten mangler autorisation til at få adgang til ressourcen.
  + **404 Not Found**: Den ønskede ressource kunne ikke findes på serveren.
* **5xx (Serverfejl)**: Indikerer problemer på serverens side.
  + **500 Internal Server Error**: En generel fejl opstod på serveren.
  + **503 Service Unavailable**: Serveren er midlertidigt ikke tilgængelig, typisk på grund af overbelastning eller vedligeholdelse.

## requests-modulet i Python

requests-modulet er et kraftfuldt og simpelt bibliotek til at sende HTTP-anmodninger i Python. Det understøtter alle standard HTTP-metoder og gør det nemt at håndtere svar fra serveren, inklusive håndtering af fejl.

Et eksempel på en simpel GET-anmodning for at printe en hjemmeside kunne se sådan ud:

**import** requests

url **=** "https://example.com"

response **=** requests**.**get**(**url**)**

**if** response**.**status\_code **==** 200**:**

**print(**"Succeed in getting resource:"**,** response**.**status\_code**)**

**print(**response**.**content**)**

**else:**

**print(**"Failed get resource:"**,** response**.**status\_code**)**

Her bruges requests.get() til at sende en GET-anmodning til en hjemmeside, og hvis anmodningen lykkes (statuskode 200), printes indholdet.

# SCRUM og Kanban

## SCRUM-metodologi

SCRUM er en agil projektledelsesmetode, der bygger på iterative processer og tæt samarbejde med interessenter. Det er vigtigt at arbejde tæt sammen med interessenter for at sikre, at kravene til programmet er klare, i overensstemmelse med forretningsmålene og at de opfyldes. Processen er opdelt i korte perioder kaldet **sprints**, som normalt varer 1-4 uger. Hvert sprint starter med planlægning og slutter med en evaluering af fremskridtene.

## Sprint Planning

Sprint planning er et centralt element i SCRUM, hvor teamet definerer, hvilke opgaver der skal udføres i det næste sprint. Formålet er at sikre, at alle teammedlemmer forstår deres ansvar, og at de prioriterede opgaver passer ind i den fastsatte sprintlængde. Sprint planning involverer:

1. **Backlog Refinement**: Gennemgang af opgaver i backloggen for at sikre, at de er prioriterede og klare til implementering.
2. **Sprintmål**: Definer klare og opnåelige mål for sprinten, som er tilpasset projektets overordnede mål.
3. **Opdeling af Opgaver**: Store opgaver opdeles i mindre, mere håndterbare opgaver, som kan fordeles på teamet.

## Daily Standup

Daily Standup er en kort, daglig mødeaktivitet, hvor teammedlemmer hurtigt deler, hvad de arbejdede på i går, hvad de vil arbejde på i dag, og om der er nogen blokeringer. Mødet hjælper med at holde teamet synkroniseret og fjerne eventuelle forhindringer hurtigt.

## Sprint Review

Sprint Review afholdes ved afslutningen af hvert sprint. Her præsenterer teamet det arbejde, de har færdiggjort, for interessenterne. Målet er at demonstrere produktet eller funktionerne og få feedback, som kan påvirke kommende opgaver eller sprintmål.

## Sprint Retrospective

Sprint Retrospective er et møde, hvor teamet reflekterer over sprintens forløb og identificerer områder for forbedring. Fokus er på, hvad der fungerede godt, hvad der kunne gøres bedre, og hvordan teamet kan optimere deres processer i næste sprint.

## Sprintforløb Opsummering

Et typisk sprint starter med **Sprint Planning**, hvor målene og opgaverne for sprinten fastlægges. Undervejs mødes teamet dagligt til **Daily Standup** for at sikre fremdrift og synkronisering. Ved sprintens afslutning holdes et **Sprint Review** for at demonstrere de leverede funktioner og få feedback. Til sidst reflekterer teamet over deres præstationer i en **Sprint Retrospective**, hvor de identificerer forbedringer til det næste sprint.

## Kanban Board

Et **Kanban board** er et visuelt værktøj, der bruges i SCRUM til at følge fremskridtene i sprinten. Opgaver er organiseret i kolonner, typisk: "To Do", "In Progress", og "Done". Det giver et klart overblik over, hvilke opgaver der er i gang, og hvilke der er afsluttet, hvilket fremmer teamets effektivitet og kommunikation.

Anvend enten [Github](https://github.com/), eller [Trello](https://trello.com/)

# Kilder og Yderligere Information

For en information kan følgende ressourcer benyttes:

1. **Thread**  
   <https://docs.python.org/3.12/library/threading.html#threading.Thread>
2. **ThreadPool**  
   <https://docs.python.org/3.12/library/concurrent.futures.html#processpoolexecutor>
3. **Asyncio**  
   <https://docs.python.org/3.12/library/asyncio.html>
4. **Requests Bibliotek**  
   <https://requests.readthedocs.io/en/latest/>
5. **SCRUM Guide**  
   <https://scrumguides.org/scrum-guide.html>
6. **HTTP Statuskoder**  
   <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#>