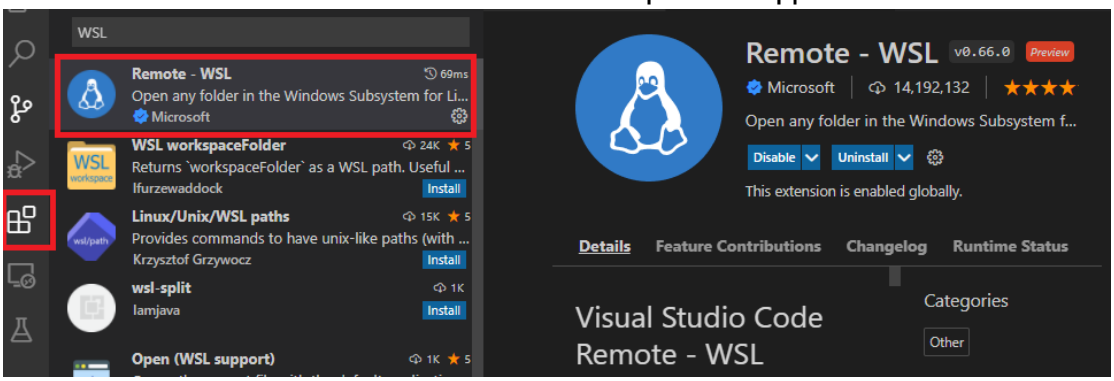
5.4.22

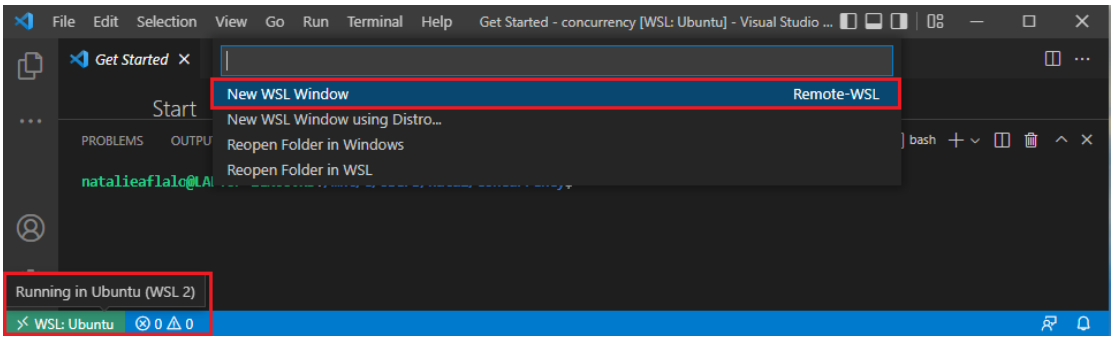
EASS שיעור 8- VSCode, המשך Async IO, Classes, Encoding binary data

עבודה עם VSCode בסביבת WSL:

בתוכנת VSCode יש להתקין את התוסף WSL-Remote.



לאחר ההתקנה יש לפתוח חלון של WSL בעזרת לחיצה בצד שמאל למטה ואז בחירה ב-new WSL window



ברגע שרשום למטה WSL:ubuntu ניתן לעבוד עם VSCode בסביבה הלינוקסית.

הרצת פעולות במקביל- 3 השיטות:

1. Multithreading
2. Processes
3. Async IO

5.4.22

הדגמת Multithreading:

ניצור תיקייה וקובץ שבהם נעבוד. נפתח את הקובץ ב-VSCode:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mkdir concurrency
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd concurrency
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$ touch concurrency.py
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$ ls
concurrency.py
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$ code .
```

נריץ פונקציה שמריצה דיפולטיבית 5 threads שכל אחד מהם מבצע print ונבדוק כמה זמן לקח לקוד לרוץ: (יש לוודא שמבצעים pip install לספריות שלא מותקנות כבר בסביבה הלוקלית)

Run and Debug

To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

concurrency.py > ...

```python
import threading
import concurrent.futures
import time
from tqdm.asyncio import trange, tqdm
import asyncio
import numpy as np


def run_threading(n_threads=5):
    threads = []
    print("Starting...")
    start = time.time()
    for i in range(n_threads):
        thread = threading.Thread(target=print, args=[f"I am thread {i}."])
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()
    end = time.time()
    print(f"Time to complete: {end - start}")

if __name__ == "__main__":
    run_threading()
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$  cd /mnt/c/Users/natal/concurrency
-python.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 42195 -- /mnt/c/Users/natal/conc
Starting...
I am thread 0.
I am thread 1.
I am thread 2.
I am thread 3.
I am thread 4.
Time to complete: 0.02778911590576172
```

- תחילה נמדוד ונתעד את זמן ההתחלה של הריצה.
- בלולאה הראשונה נגדיר כל thread, נפעיל אותו עם start ונוסיף אותו ל-pool שהיא רשימת threads.
- בלולאה השנייה נריץ את ה-threads בעזרת פונקציית join – הפונקציה דורשת שהמערכת תמתין עד סיום הריצה של ה-thread שזימן אותה (מבצעת BLOCK)
- לאחר שהיא מסתיימת נמדוד את זמן הסיום. כעת נחשב את ההפרש בין זמן ההתחלה לסיום.

נדגים מצב שבו לא משתמשים בפונקציית join:

```python
def run_threading(n_threads=5):
    threads = []
    print("Starting...")
    start = time.time()
    for i in range(n_threads):
        thread = threading.Thread(target=print, args=[f"I am thread {i}."])
        thread.start()
        threads.append(thread)

    #for thread in threads:
        #thread.join()
    end = time.time()
    print(f"Time to complete: {end - start}")

if __name__ == "__main__":
    run_threading()
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$  cd /mnt/c/Users/natal/concur
-python.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 37177 -- /mnt/c/Users/natal/
Starting...
I am thread 0.
I am thread 1.
I am thread 2.
I am thread 3.
Time to complete: 0.017986536026000977I am thread 4.
```

ניתן לראות שה-thread האחרון סיים לרוץ (הדפיס) אחרי שנמדד זמן הסיום והודפס הזמן הסופי.

5.4.22

הדגמת Processes:

```python
import threading
import concurrent.futures
import time
from tqdm.asyncio import trange, tqdm
import asyncio
import numpy as np


def do_concurrent(n=32):
    start = 10_000_000
    print_list = [i for i in range(start, start + n)]
    print("Starting...")
    start = time.time()
    with concurrent.futures.ProcessPoolExecutor(max_workers=2) as executor:
        futures = {executor.submit(print, i): i for i in print_list}
    for f in concurrent.futures.as_completed(futures):
        print(f"done {futures[f]} {f.result()}")
    end = time.time()
    print(f"Time to complete: {end - start}")

if __name__ == "__main__":
    do_concurrent(5)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$ /home/linuxbrew/.linuxbre
w/opt/python@3.10/bin/python3 /mnt/c/Users/natal/concurrency/concurrency.py
Starting...
10000000
10000002
10000001
10000003
10000004
done 10000003 None
done 10000000 None
done 10000002 None
done 10000004 None
done 10000001 None
Time to complete: 0.024893999099731445
```

פונקציית as_completed בשורה 16 מחכה עד להשלמת הפעולה של כל process, וכך תוצאת הזמן הסופית מתבצעת אחרי סיום כל התהליכים.

5.4.22

הדגמת Async IO:

```python
import threading
import concurrent.futures
import time
from tqdm.asyncio import trange, tqdm
import asyncio
import numpy as np

async def wait_and_print(t, n):
    await asyncio.sleep(t)
    print(f"coroutine {n} slept for {t} seconds")


async def do_tqdm_asyncio(n=10):
    coroutines = []
    async for i in trange(n):
        print(f"asyncio {i}")
        coroutines.append(wait_and_print(np.random.randint(1, 5), i))
    await asyncio.gather(*coroutines)


if __name__ == "__main__":
    asyncio.run(do_tqdm_asyncio(6))
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/concurrency$ /home/linuxbrew/.linuxbre
w/opt/python@3.10/bin/python3 /mnt/c/Users/natal/concurrency/concurrency.py
  0%|                                                          | 0/6 [00:00<?, ?it/s]
asyncio 0
asyncio 1
asyncio 2
asyncio 3
asyncio 4
asyncio 5
100%|                                                          | 6/6 [00:00<00:00, 15096.48it/s]
coroutine 5 slept for 1 seconds
coroutine 4 slept for 2 seconds
coroutine 0 slept for 3 seconds
coroutine 1 slept for 3 seconds
coroutine 2 slept for 3 seconds
coroutine 3 slept for 3 seconds
```

פונקציית gather בשורה 18 מחכה לכל ה-coroutines במערך שיסתיימו (בלעדיה לא היינו רואים את ההדפסות "coroutine {n} slept for {t} seconds").

השימוש בכוכבית בשורה 18 הוא כדי לבצע flatten. כלומר אם המערך הוא בצורה ([3,3,2,2,3,3],) אז הכוכבית הופכת אותו לצורה (3,3,2,2,3,3).

5.4.22

Class בפייתון:

- אינקפסולציה- מאפיין חשוב בתכנות מונחה עצמים המתייחס לאריזה של מידע (משתנים) ופעולות על המידע (מתודות) כיחידה אחת.
- אובייקטים- אינקפסולציה של משתנים ומתודות.
- Class מחלקה - תבנית ליצירת אובייקטים.

דוגמה:

Lst=list() #יצירת אובייקט רשימה ריקה
Lst=list([1,2,3]) #יצירת אובייקט רשימה עם ערכים התחלתיים

יצירת class –

- Self: כמו this בשפות אחרות, מתאר את האובייקט עצמו.
- __init__ : פונקציית בנאי
- Super : המחלקה ממנה אנו יורשים. בדוגמה- הsuper של dog הוא animal.

```
class Animal:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight
    def eat(self):
        print("%s is eating." % self.name)
    def make_sound(self):
        print("%s is making a sound." % self.name)
class Dog(Animal):
    def __init__(self, name, weight, size):
        self._size = size
        super().__init__(name, weight)
    def make_sound(self):
        print("Barking")
```

Encoding binary data:

לא כל מידע ניתן לתרגם ל-string, כמו תמונה. אך ניתן להמיר כל דבר לנתון בינארי, וכך ניתן להעביר אותו.

- Encode- המרת קובץ/תמונה/מחרוזת וכו' למידע בינארי
- Decode- המרת מידע בינארי לקובץ/תמונה/מחרוזת וכו'. יכול להיות שיהיה שוני בין הקובץ המקורי לקובץ שמוחזר מפעולת ה-decode.

```
>>> import base64
>>> encoded_string = base64.b64encode(b'binary\x00string')
'YmluYXJ5AHN0cmluZw=='
>>> decoded_bytes = base64.b64decode(encoded_string)
b'binary\x00string'
```

נדגים כיצד להשתמש ב-encoding על תמונה:

שלב ראשון- בצד שרוצה לשלוח את התמונה: (המרה מקובץ png לבינארי)

```
with open('image.png', 'rb') as f:
    data = f.read()
encoded_data = base64.b64encode(data)
```

שלב שני- בצד שרוצה לקבל את התמונה: (המרה חזרה של בינארי לקובץ png)

```
decoded_data = base64.b64decode(encoded_data)
with open('image_copy.png', 'wb') as f:  # note the 'wb' mode!
    f.write(decoded_data)
```

הדגמה של פקודת GET עם FASTAPI שמחזירה תמונה כמידע בינארי ב-JSON:

```
import base64
app=FastAPI()
img="./image.jpg"
@app.get("/v1/get-image")
async def main():
       return {"image": base64.b64encode(open(img, "rb").read())}
```

אם נרצה להחזיר מערך של תמונות כמידע בינארי ב-JSON:

```
img_arr=["a.jpg","b.jpg","c.jpg"]
@app.get("/v1/get-image-arr")
async def main():
       return {"image": [base64.b64encode(open(single_img, "rb").read())
                         for single_img in img_arr]}
```

הערה- ספרייה נוספת שניתן להשתמש בה- imageio.