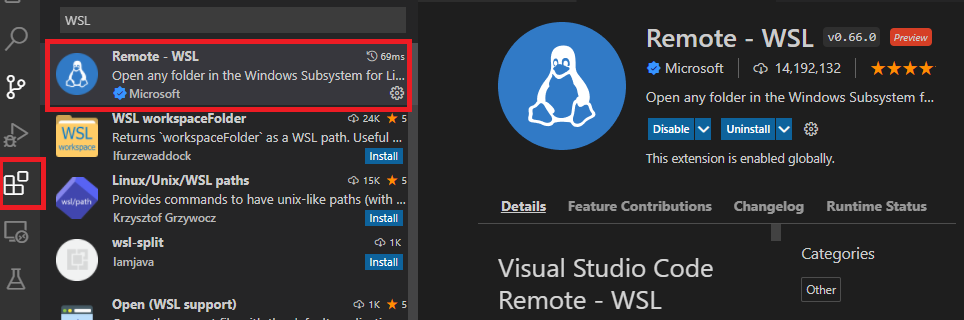
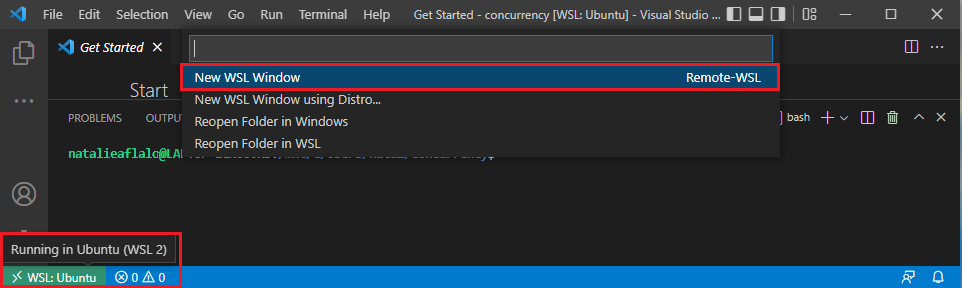
**EASS שיעור 8- VSCode, המשך Async IO, Classes, Encoding binary data**

**עבודה עם VSCode בסביבת WSL:**

בתוכנת VSCode יש להתקין את התוסף WSL-Remote.   


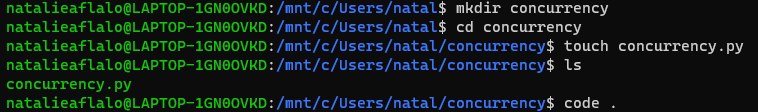
לאחר ההתקנה יש לפתוח חלון של WSL בעזרת לחיצה בצד שמאל למטה ואז בחירה ב-new WSL window  


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

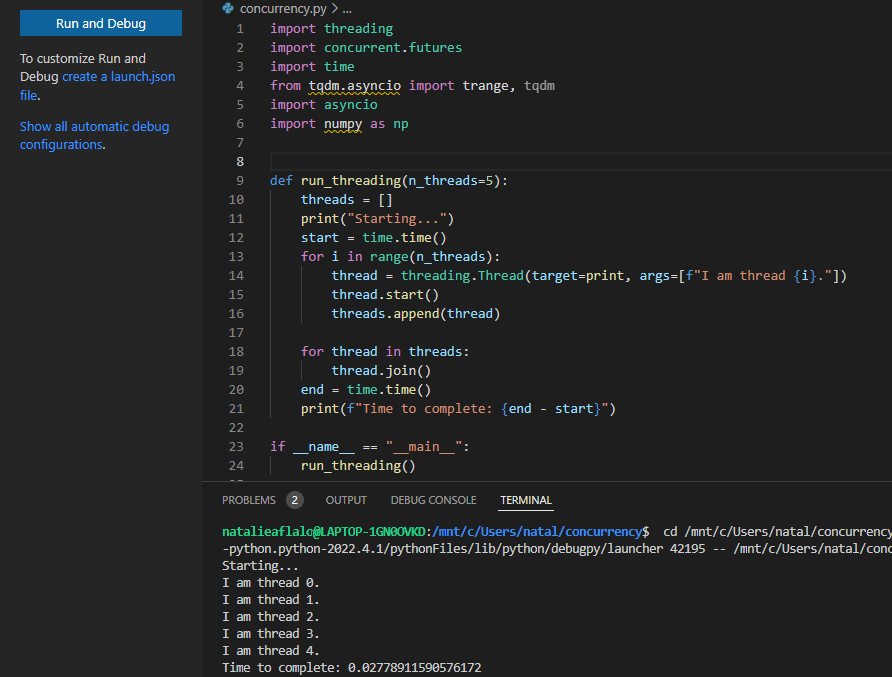
**הרצת פעולות במקביל- 3 השיטות:**

1. Multithreading
2. Processes
3. Async IO

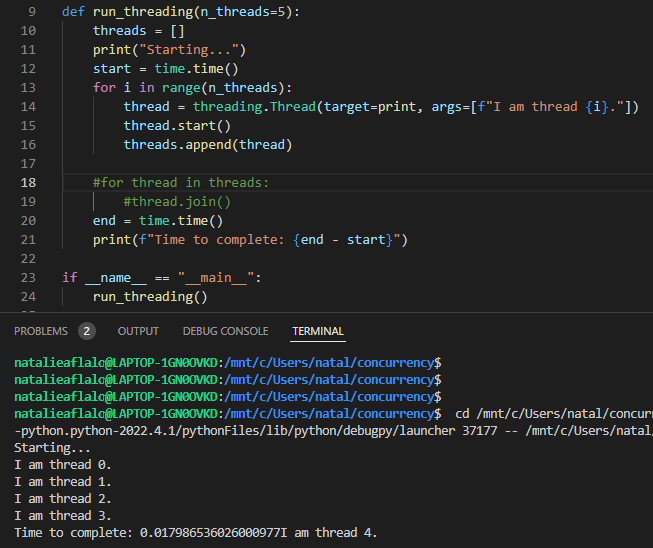
הדגמת Multithreading:

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


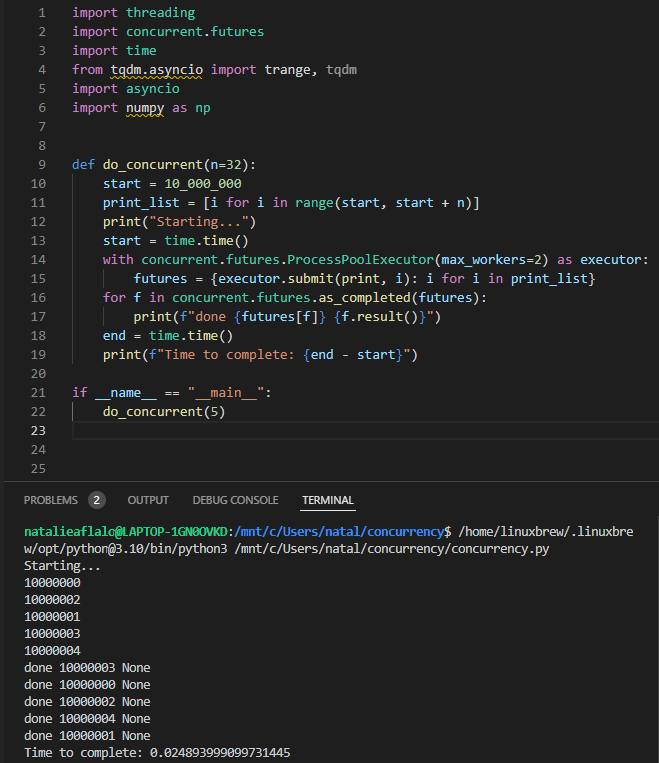
נריץ פונקציה שמריצה דיפולטיבית 5 threads שכל אחד מהם מבצע print ונבדוק כמה זמן לקח לקוד לרוץ: (יש לוודא שמבצעים pip install לספריות שלא מותקנות כבר בסביבה הלוקלית)



* 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.

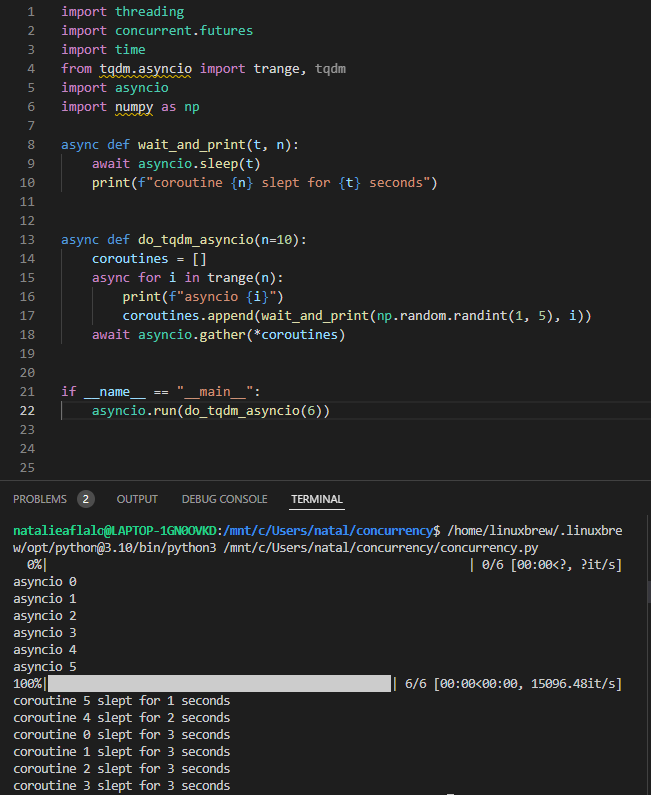
נדגים מצב שבו לא משתמשים בפונקציית join:  
  
ניתן לראות שה-thread האחרון סיים לרוץ (הדפיס) אחרי שנמדד זמן הסיום והודפס הזמן הסופי.

הדגמת Processes:



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

הדגמת Async IO:



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) .

**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.