

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра конструювання електронно обчислювальної апаратури

**Лабораторна Робота 2**

з дисципліни

«Високорівнева розробка програмного забезпечення»

Виконала студентка:

Харчук Каріна

Навчальної групи: ФЕЛ, ДК-21

Перевірив: Мушта І. А.

                                                         Київ 2024

**Завдання**

*Завдання 1. Аналіз даних із Kaggle*

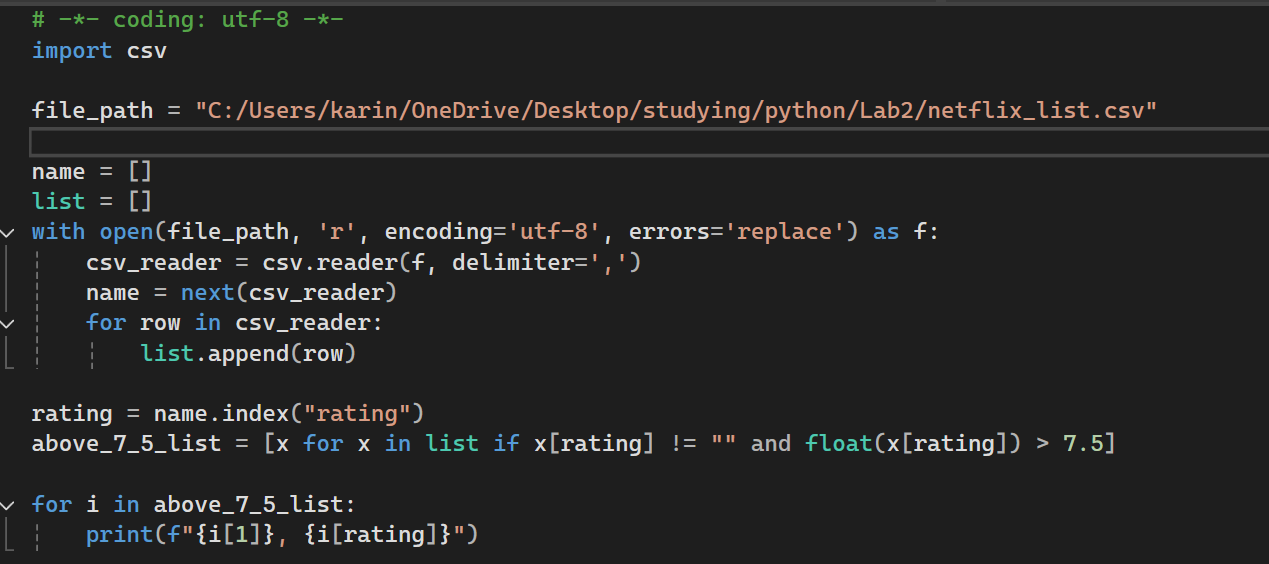
1. **Завантажте датасет із Kaggle**:

<https://www.kaggle.com/datasets/snehaanbhawal/netflix-tv-shows-and-movie-list/data>

1. **Виконайте наступні операції:**

**# -------------------------------------------- на 2 календарний контроль ----------------------------------**

* + Використайте вбудовані засоби Python для зчитування файлу netflix\_list.csv. Розділіть рядки за допомогою методу split(',') та збережіть у список списків.
  + Використайте **comprehensions** для виконання таких дій:   
    a) Створіть новий список, що містить **тільки шоу або фільми, де рейтинг більше 7.5**. (поле rating);



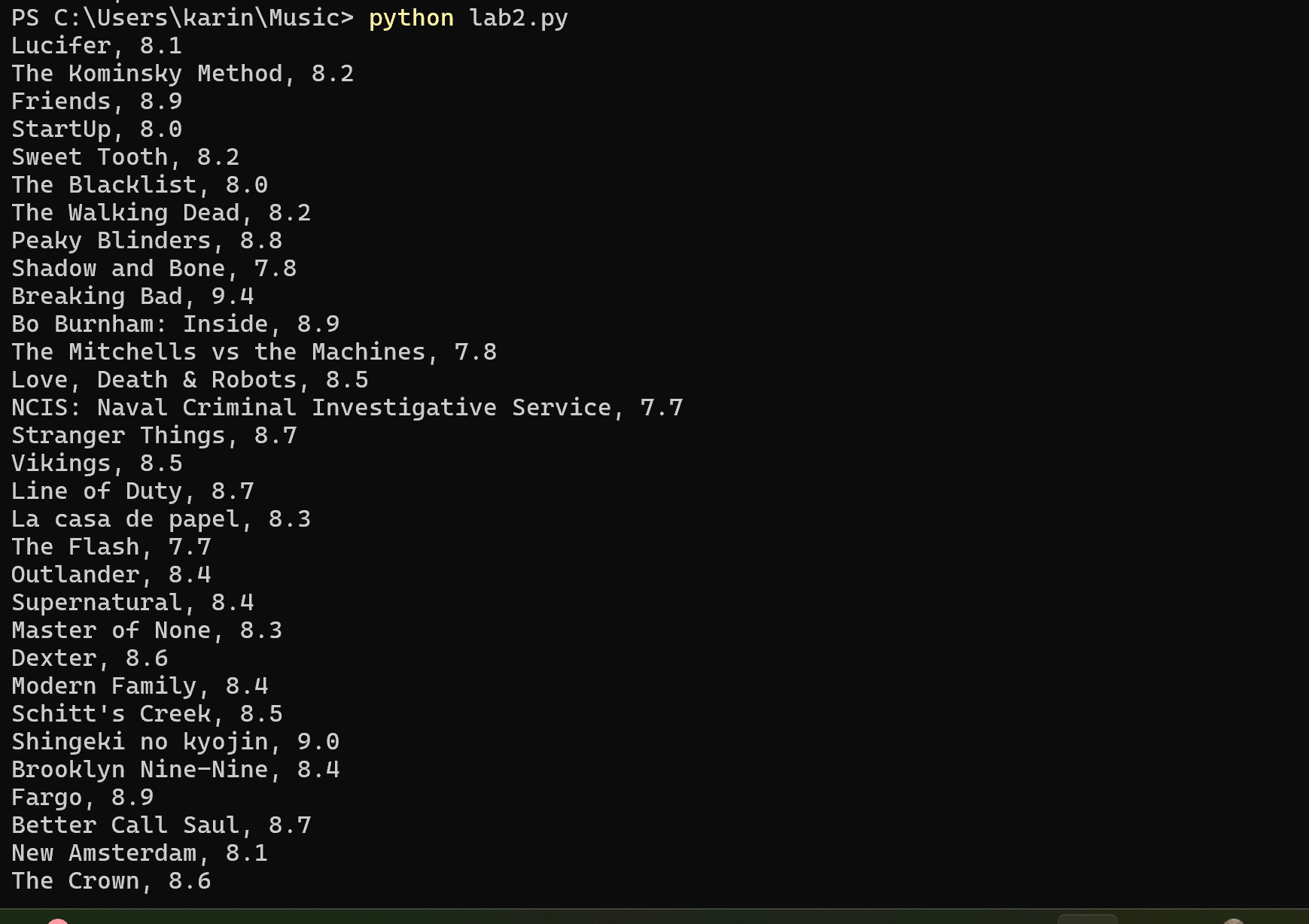
Перший рядок вказує, що файл збережений у кодуванні UTF-8. Це важливо для обробки тексту, особливо якщо у файлі є неанглійські символи.

import csv  
Імпортується модуль csv, який спрощує роботу з файлами у форматі CSV (Comma-Separated Values). Пишемо в file\_path  шлях до CSV-файлу, з яким працюватиме програма.  
Далі і ніціалізується порожній список name, у який будуть зчитані заголовки стовпців файлу та порожній список list, у який буде записано вміст рядків CSV-файлу.   
Відкривається файл file\_path для читання ('r') з використанням кодування utf-8. Параметр errors='replace' замінює некоректні символи спеціальними маркерами, щоб уникнути помилок під час читання.   
Далф використовується csv.reader для читання рядків файлу, при цьому дані розділяються комами.  
Зчитується перший рядок файлу (заголовки стовпців) і зберігається у список name.  
Фором починається цикл для обробки кожного наступного рядка файлу.   
Додає поточний рядок із файлу до списку list.   
rating = name.index("rating") знаходить індекс стовпця з назвою "rating" у заголовках (список name). Це дозволяє звертатися до стовпця рейтингу у кожному рядку.   
Створюється новий список above\_7\_5\_list, який включає лише ті рядки (x), де значення у стовпці rating:

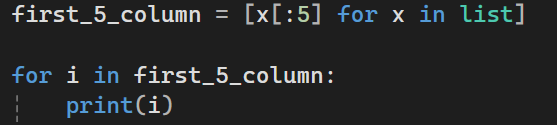
Не порожнє (x[rating] != "").

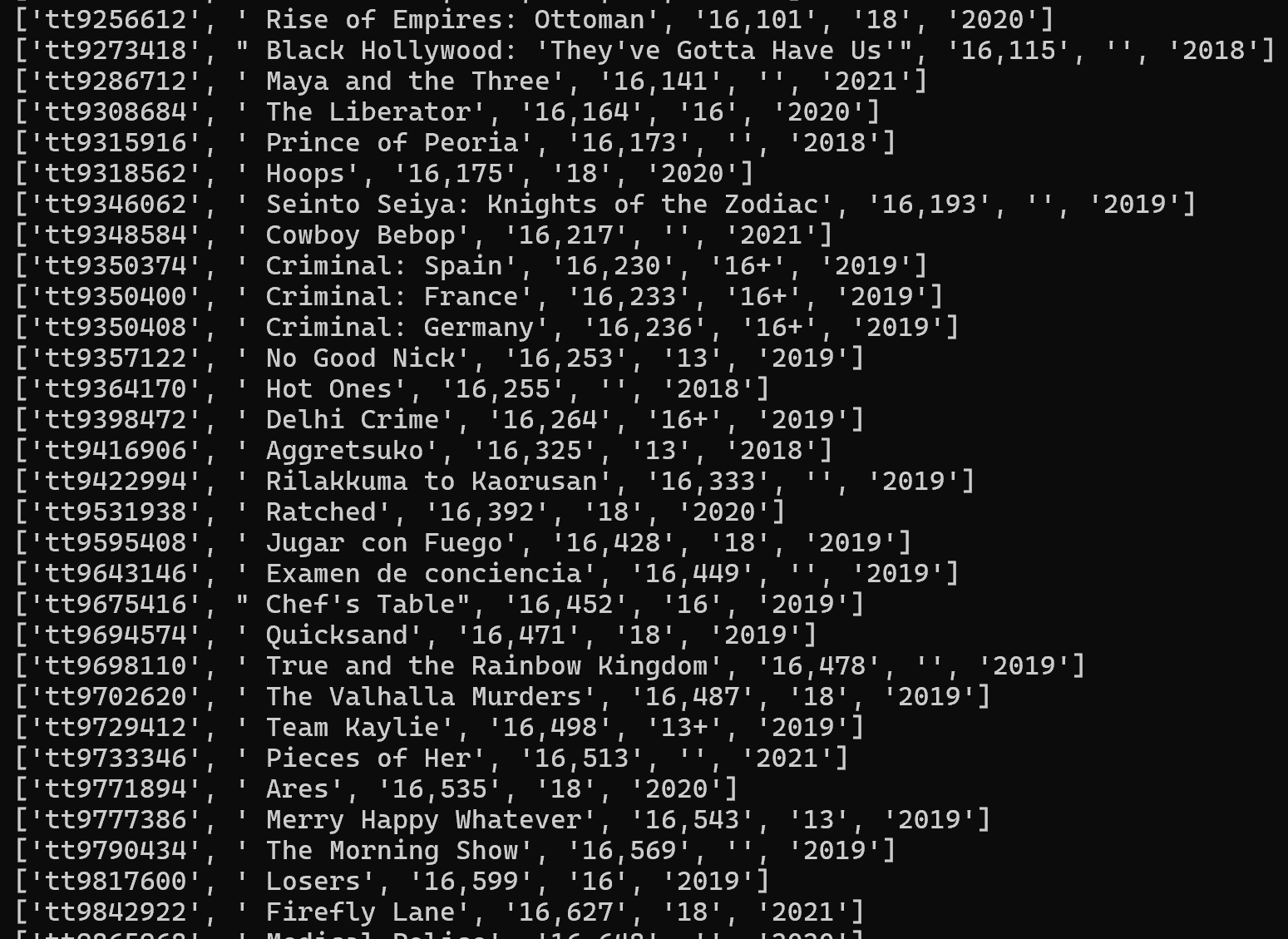
Більше 7.5 (float(x[rating]) > 7.5).

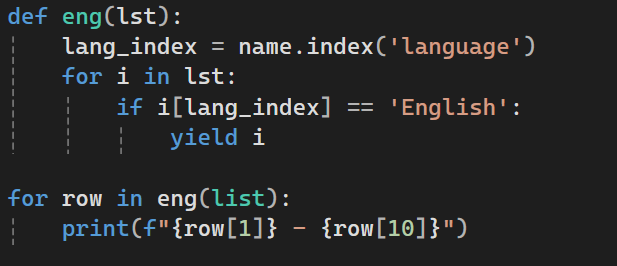
Потім починається цикл для виведення даних із відфільтрованого списку above\_7\_5\_list.   
Для кожного рядка у списку above\_7\_5\_list виводиться другий стовпець і рейтинг використовуючи форматування рядка.



b) Зберігайте тільки перші 5 колонок кожного рядка.



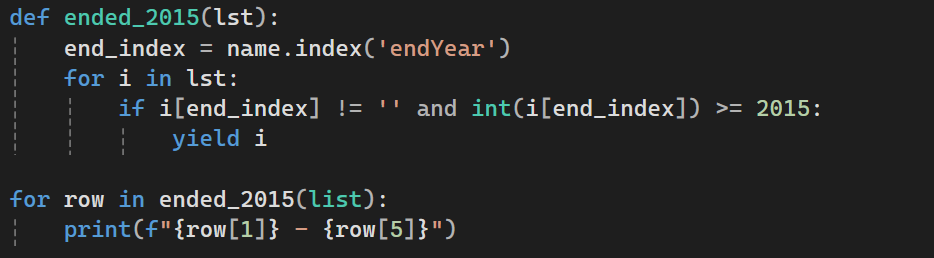


* + Напишіть **генераторну** функцію, яка буде повертати рядки, у яких:   
    a) Мова шоу або фільму —англійська (поле language); 

Оголошується функція eng, яка приймає на вхід список lst.   
Використовуючи заголовки, знаходимо індекс стовпця з назвою "language". Це дозволяє динамічно працювати зі стовпцями, не прив'язуючись до їхньої позиції.   
Починається цикл, який перебирає всі рядки списку lst і для кожного рядка перевіряється, чи значення в стовпці "language" дорівнює "English". Якщо умова виконується: генератор yield i  
Рядок повертається через yield. Замість того, щоб одразу повертати всі результати, yield повертає їх по одному.

Далі запускається цикл, який використовує eng. Для кожного рядка, який повертає eng виконується обробка. Потім виводиться значення другого стовпця та десятого стовпця. Значення об'єднуються у строку через дефіс (-).

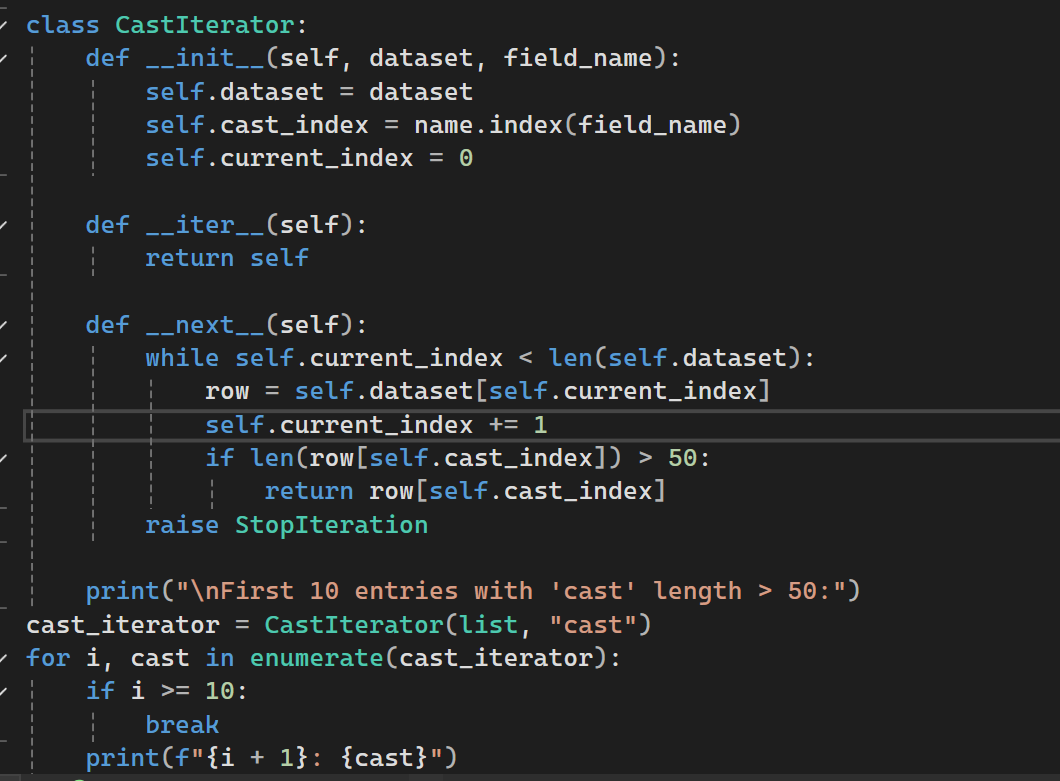
* +   
    b) Це або серіал, або фільм (поле type), що закінчився після 2015 року (поле endYear).   
    Використайте ключове слово **yield** для поступового повернення даних.

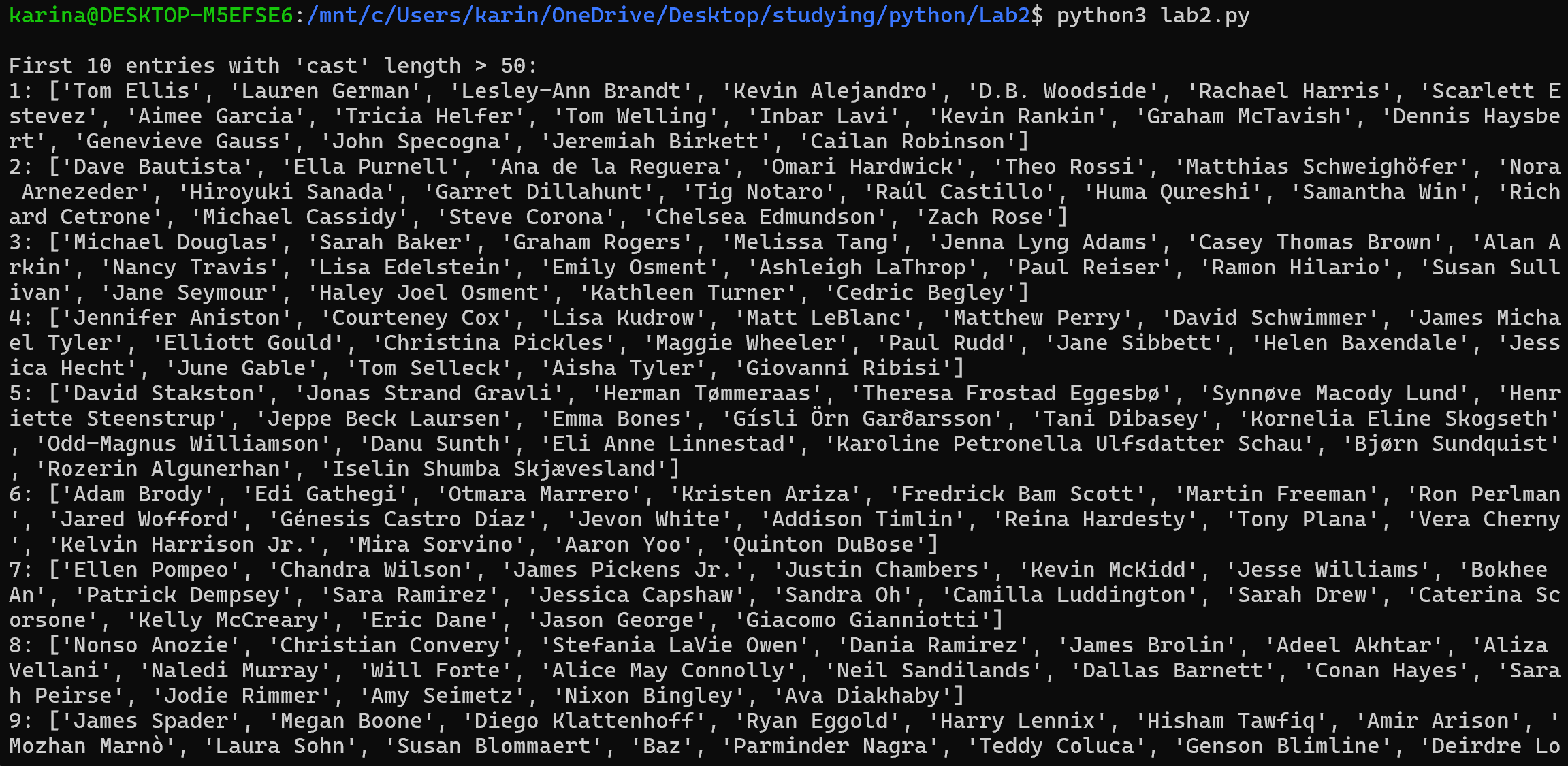




# ----------------------------------------------------------------------------------------------------------------------

* + Напишіть **ітератор**, який проходить через датасет та повертає лише **основний акторський склад** (поле cast), якщо його довжина перевищує 50 символів. Використайте цей ітератор для друку перших 10 таких записів.

**

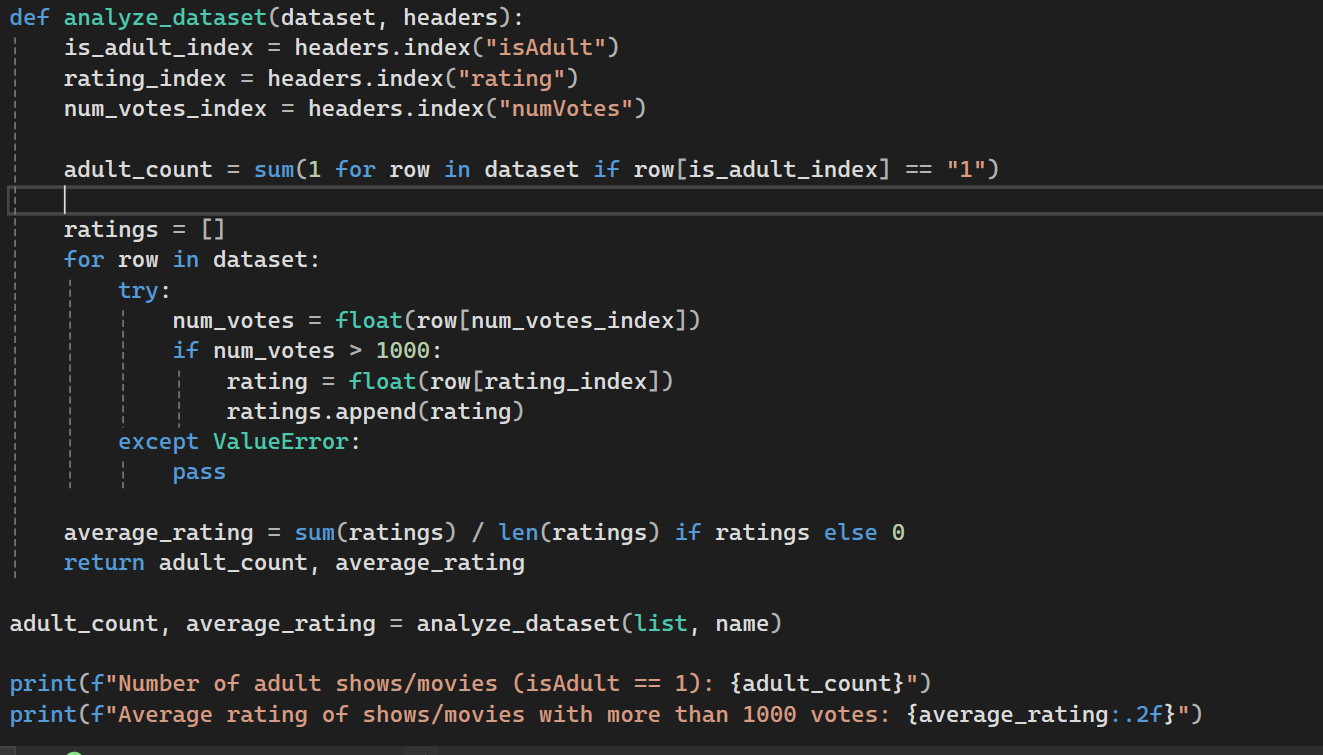


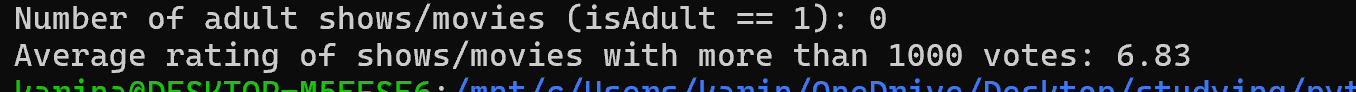
Клас CastIterator ініціалізується з датасетом (dataset) і назвою поля (field\_name). У методі \_\_init\_\_ визначається індекс стовпця cast через name.index(field\_name).

Наступний метод овертає самого себе як ітератор. Метод next спочатку проходить по датасету і якщо довжина cast у поточному рядку перевищує 50 символів, повертає це значення, якщо всі рядки оброблено, викликається StopIteration.

Як використовується ітератор:   
він проходить по всьому датасету, повертаючи лише ті записи, де довжина cast > 50 і логіка обмежує вивід до перших 10 записів.

* + Напишіть функцію, яка підраховує:   
    a) Скільки **шоу/фільмів для дорослих** (isAdult == 1) є у датасеті;   
    b) Середній рейтинг шоу та фільмів з більш ніж 1000 голосів (поле numVotes).

**

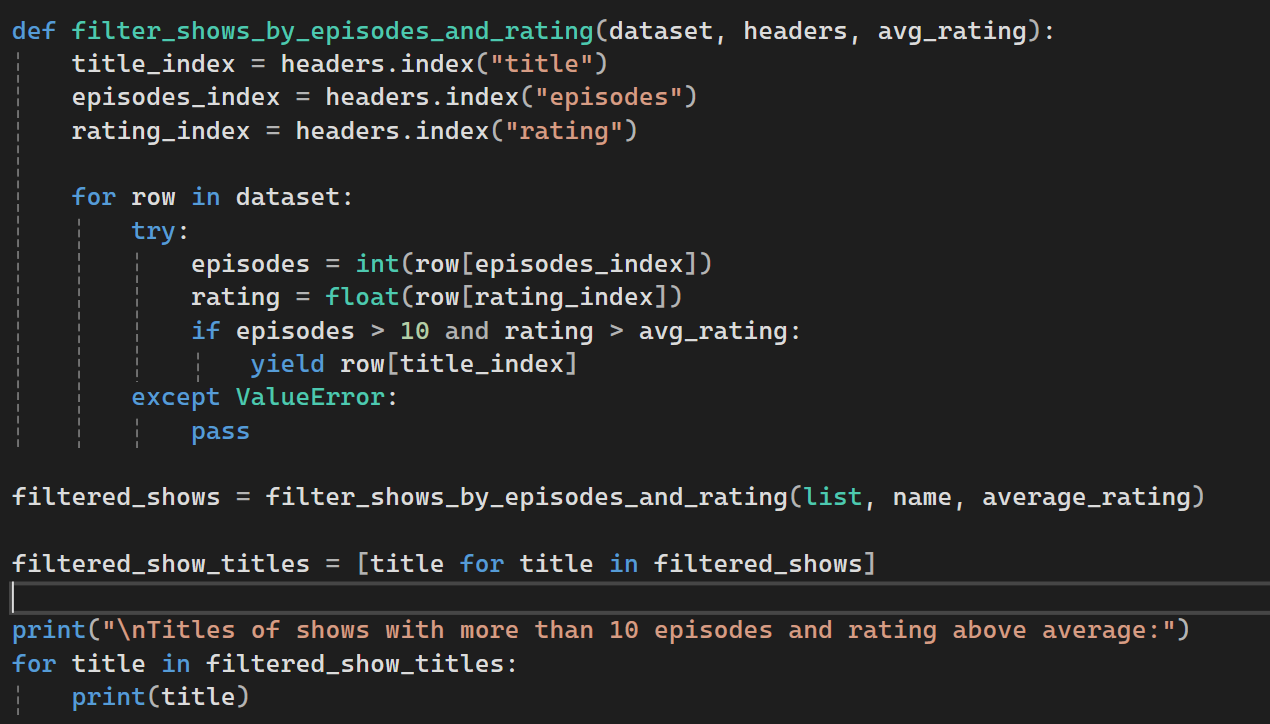
**

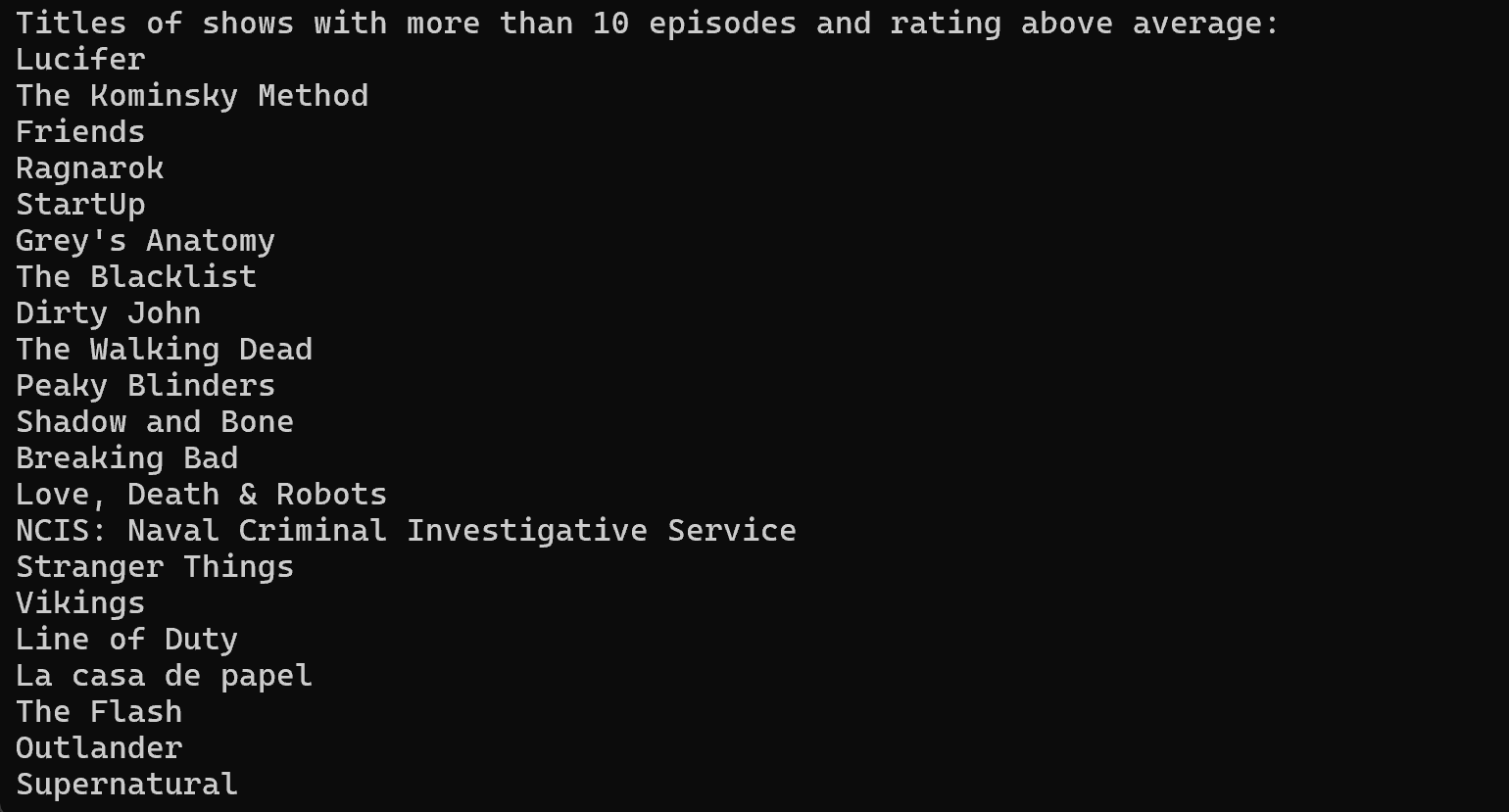
Функція analyze\_dataset обробляє датасет (список рядків) і заголовки колонок.  Знаходимо індекси колонок isAdult, rating і numVotes у списку заголовків headers. Це дозволяє звертатися до значень у відповідних колонках за індексом. Проходить по кожному рядку в dataset. Якщо значення в колонці isAdult дорівнює "1", додає 1 до загальної суми (adult\_count).

Даліс ініціалізується порожній список ratings для збереження рейтингів. Цикл проходить по кожному рядку датасету. Там пробуємо перетворити значення numVotes на float. Якщо це вдається і значення більше 1000, виконується наступний крок. Пробуємо перетворити значення rating на float. Якщо це вдається, значення додається до списку ratings. Якщо значення numVotes або rating некоректне, наприклад, текст або порожнє, виконується except ValueError, і рядок пропускається. Обчислюємо середній рейтинг, якщо є хоча б одне валідне значення в списку ratings. Якщо список ratings порожній, середній рейтинг дорівнює 0. рядок return adult\_count, average\_rating означає, що повертаємо два значення: adult\_count — кількість дорослого контенту (isAdult == 1) та average\_rating — середній рейтинг для записів із більше ніж 1000 голосами.

Потім викликаємо функцію analyze\_dataset, передаючи їй датасет (list) і заголовки (name). Зберігаємо результати в змінних adult\_count і average\_rating.

* + Використайте **генератор** та **comprehensions** для обробки файлу, щоб створити список заголовків шоу, які:   
    a) Мають більше 10 епізодів;   
    b) Рейтинг вище середнього.

**

**

Функція filter\_shows\_by\_episodes\_and\_rating використовує генератор, щоб пройти через датасет і фільтрувати шоу. Перевіряє чи кількість епізодів (episodes) більше 10 і чи рейтинг (rating) вище середнього (передається через параметр avg\_rating). Вона повертає заголовок шоу (title), якщо умови виконуються.

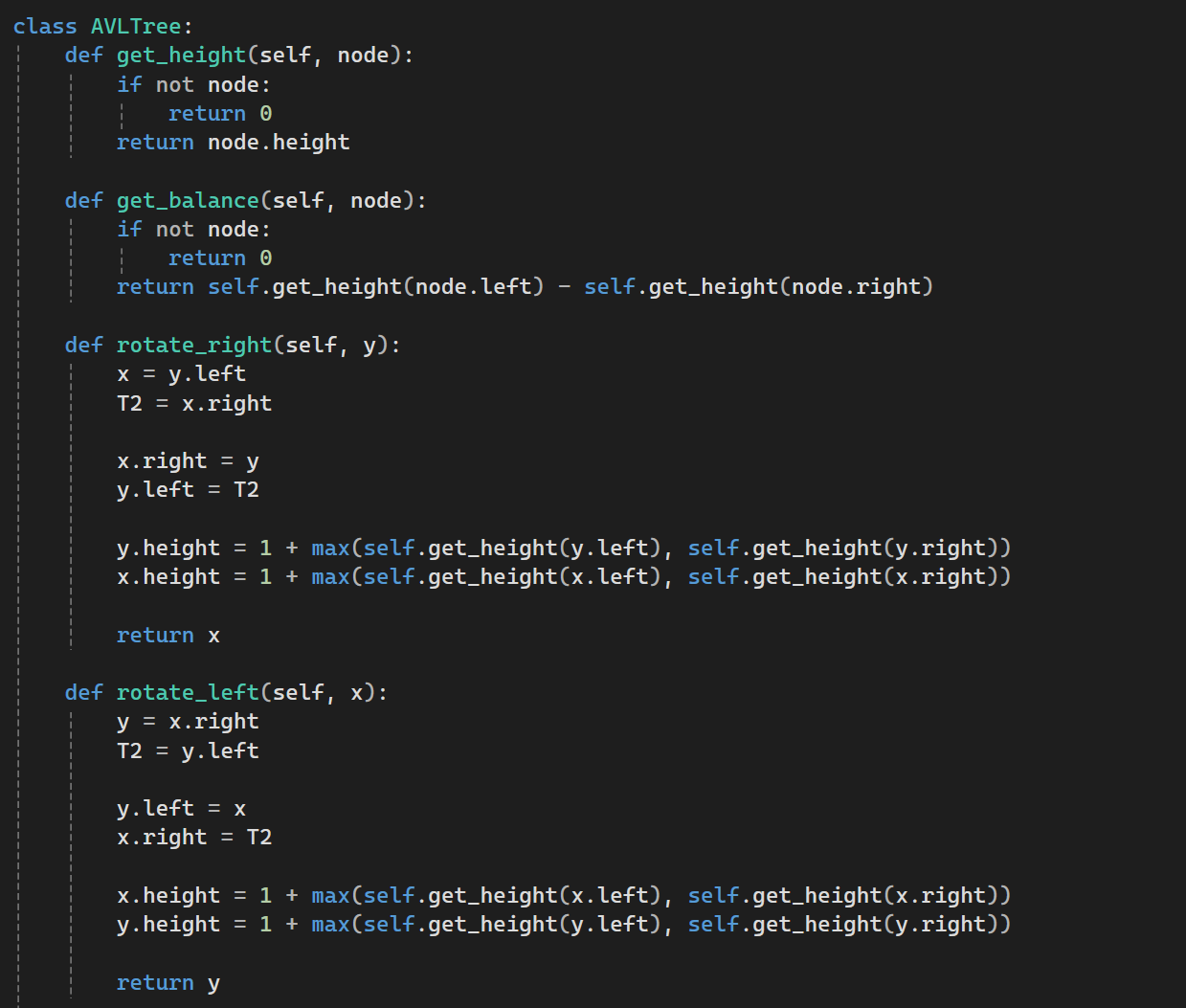
Список через comprehensions використовує результат генератора (filtered\_shows) для створення списку заголовків.

***Завдання 2. Реалізація самобалансуючого дерева***

Реалізуйте **самобалансуюче двійкове дерево**. Дерево повинно автоматично балансуватися під час додавання нових елементів за рахунок розбиття та обертання (приклад – алгоритм AVL-дерева).

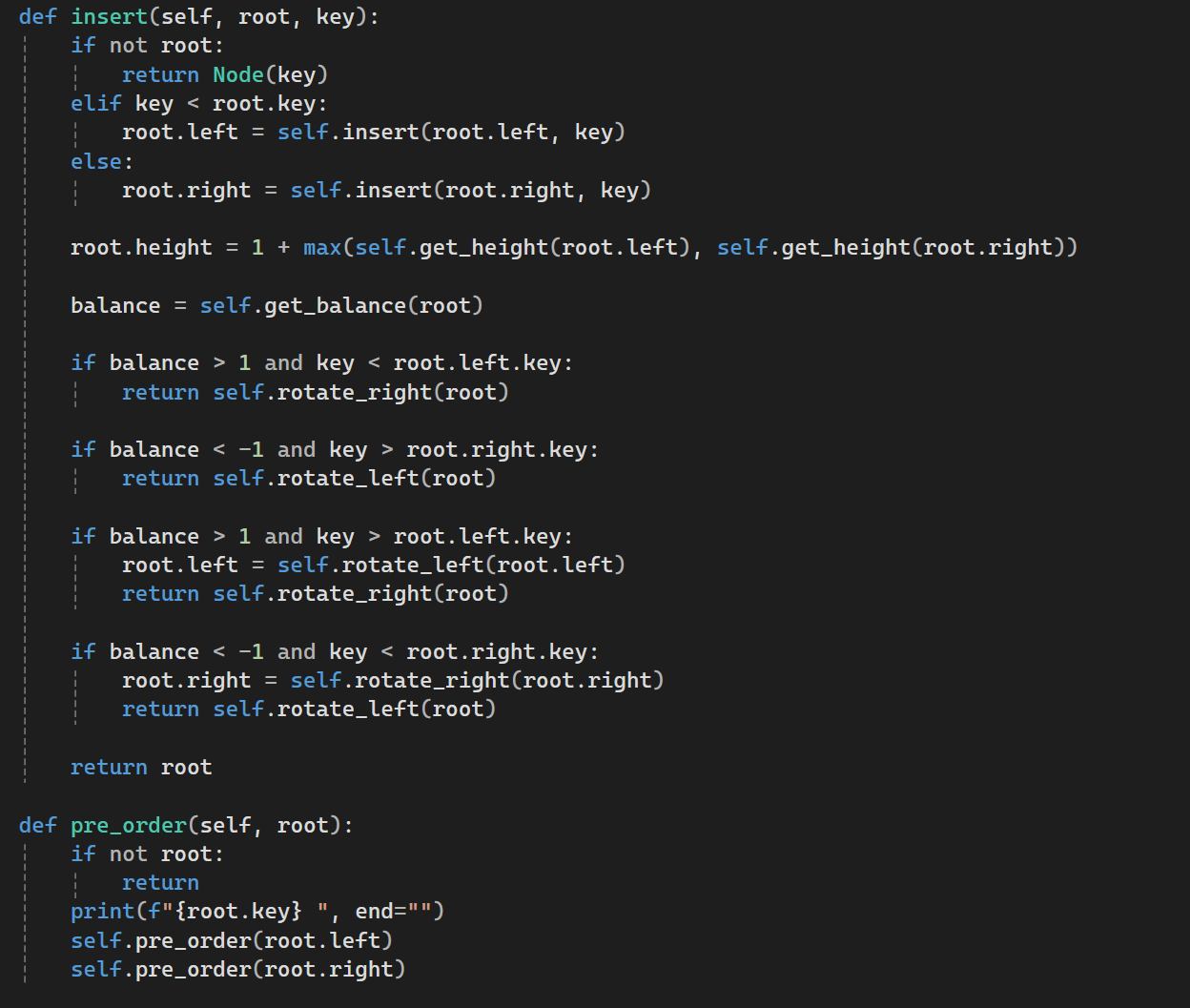
### ****Умови виконання:****

1. Використайте класи для реалізації дерева.
2. Використовуйте **рекурсивні функції** для вставки нових елементів та балансування дерева.
3. Створіть функцію для **обчислення висоти піддерева**.
4. Реалізуйте функцію для **обертання** дерева вліво та вправо для балансування.
5. Функція insert повинна вставляти нові елементи та підтримувати баланс дерева.

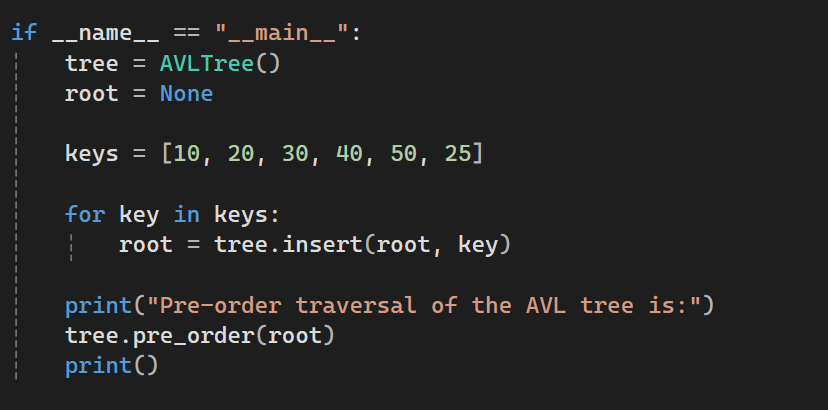


Клас Node визначає вузол дерева. У конструкторі ініціалізується значення ключа (key), що зберігається у вузлі, а також посилання на ліве і праве піддерева (left і right), які за замовчуванням рівні None. Поле height зберігає висоту вузла і на початку ініціалізується як 1 (вузол без дітей).

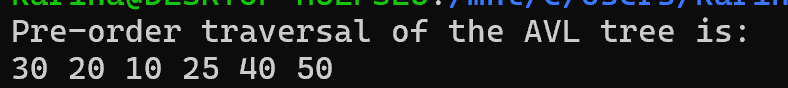
Клас AVLTree реалізує всі необхідні методи для роботи з AVL-деревом. Метод get\_height обчислює висоту вузла. Якщо вузол не існує (None), висота дорівнює 0. Метод get\_balance обчислює баланс вузла як різницю висот його лівого і правого піддерев. Якщо вузол не існує, повертається 0. Метод rotate\_right виконує праве обертання дерева навколо заданого вузла. Це потрібно для балансування дерева у випадках, коли ліве піддерево стає занадто високим. Спочатку зберігаються потрібні вузли, потім виконується переназначення посилань, а висоти вузлів оновлюються відповідно до нової структури. Повертається новий корінь піддерева після обертання. Метод rotate\_left працює аналогічно, але виконує обертання вліво, коли праве піддерево стає занадто високим.

Метод insert відповідає за вставку нового елемента в дерево. Спочатку виконується звичайна вставка, як у звичайному двійковому дереві пошуку: якщо ключ менший за поточний вузол, перехід до лівого піддерева, інакше до правого. Після вставки оновлюється висота поточного вузла. Далі обчислюється баланс вузла. Якщо баланс виходить за межі діапазону [-1, 1], виконується відповідне обертання: праве (rotate\_right) для лівого випадку, ліве (rotate\_left) для правого випадку, або комбінація обох для складніших сценаріїв (лівоправий чи праволівий).

Метод pre\_order виконує обхід дерева у порядку pre-order (обхід кореня, потім лівого і правого піддерев). Якщо вузол не існує, обхід завершено. Інакше ключ вузла виводиться, і метод рекурсивно викликається для лівого і правого піддерев.



Секція if \_\_name\_\_ == "\_\_main\_\_": запускає тестування дерева. Створюється екземпляр класу AVLTree, і нові ключі послідовно додаються до дерева за допомогою методу insert. Після кожної вставки дерево автоматично балансується. Тепер 0020 викликається метод pre\_order, щоб вивести ключі у прямому обході, демонструючи збалансовану структуру дерева.



# -\*- coding: utf-8 -\*-

import csv

#task 1

'''

file\_path = "/mnt/c/Users/karin/OneDrive/Desktop/studying/python/Lab2/netflix\_list.csv"

name = []

list = []

with open(file\_path, 'r', encoding='utf-8', errors='replace') as f:

csv\_reader = csv.reader(f, delimiter=',')

name = next(csv\_reader)

for row in csv\_reader:

list.append(row)

'''

'''

rating = name.index("rating")

above\_7\_5\_list = [x for x in list if x[rating] != "" and float(x[rating]) > 7.5]

for i in above\_7\_5\_list:

print(f"{i[1]}, {i[rating]}")

'''

'''

first\_5\_column = [x[:5] for x in list]

for i in first\_5\_column:

print(i)

'''

'''

def eng(lst):

lang\_index = name.index('language')

for i in lst:

if i[lang\_index] == 'English':

yield i

for row in eng(list):

print(f"{row[1]} - {row[10]}")

'''

'''

def ended\_2015(lst):

end\_index = name.index('endYear')

for i in lst:

if i[end\_index] != '' and int(i[end\_index]) >= 2015:

yield i

for row in ended\_2015(list):

print(f"{row[1]} - {row[5]}")

'''

'''

class CastIterator:

def \_\_init\_\_(self, dataset, field\_name):

self.dataset = dataset

self.cast\_index = name.index(field\_name)

self.current\_index = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

while self.current\_index < len(self.dataset):

row = self.dataset[self.current\_index]

self.current\_index += 1

if len(row[self.cast\_index]) > 50:

return row[self.cast\_index]

raise StopIteration

print("\nFirst 10 entries with 'cast' length > 50:")

cast\_iterator = CastIterator(list, "cast")

for i, cast in enumerate(cast\_iterator):

if i >= 10:

break

print(f"{i + 1}: {cast}")

'''

'''

def analyze\_dataset(dataset, headers):

is\_adult\_index = headers.index("isAdult")

rating\_index = headers.index("rating")

num\_votes\_index = headers.index("numVotes")

adult\_count = sum(1 for row in dataset if row[is\_adult\_index] == "1")

ratings = []

for row in dataset:

try:

num\_votes = float(row[num\_votes\_index])

if num\_votes > 1000:

rating = float(row[rating\_index])

ratings.append(rating)

except ValueError:

pass

average\_rating = sum(ratings) / len(ratings) if ratings else 0

return adult\_count, average\_rating

adult\_count, average\_rating = analyze\_dataset(list, name)

print(f"Number of adult shows/movies (isAdult == 1): {adult\_count}")

print(f"Average rating of shows/movies with more than 1000 votes: {average\_rating:.2f}")

def filter\_shows\_by\_episodes\_and\_rating(dataset, headers, avg\_rating):

title\_index = headers.index("title")

episodes\_index = headers.index("episodes")

rating\_index = headers.index("rating")

for row in dataset:

try:

episodes = int(row[episodes\_index])

rating = float(row[rating\_index])

if episodes > 10 and rating > avg\_rating:

yield row[title\_index]

except ValueError:

pass

filtered\_shows = filter\_shows\_by\_episodes\_and\_rating(list, name, average\_rating)

filtered\_show\_titles = [title for title in filtered\_shows]

print("\nTitles of shows with more than 10 episodes and rating above average:")

for title in filtered\_show\_titles:

print(title)

'''

#task 2

class Node:

def \_\_init\_\_(self, key):

self.key = key # Значення вузла

self.left = None # Ліве піддерево

self.right = None # Праве піддерево

self.height = 1 # Висота вузла

class AVLTree:

def get\_height(self, node):

if not node:

return 0

return node.height

def get\_balance(self, node):

if not node:

return 0

return self.get\_height(node.left) - self.get\_height(node.right)

def rotate\_right(self, y):

x = y.left

T2 = x.right

# Виконуємо обертання

x.right = y

y.left = T2

# Оновлюємо висоти

y.height = 1 + max(self.get\_height(y.left), self.get\_height(y.right))

x.height = 1 + max(self.get\_height(x.left), self.get\_height(x.right))

# Повертаємо новий корінь

return x

def rotate\_left(self, x):

y = x.right

T2 = y.left

# Виконуємо обертання

y.left = x

x.right = T2

# Оновлюємо висоти

x.height = 1 + max(self.get\_height(x.left), self.get\_height(x.right))

y.height = 1 + max(self.get\_height(y.left), self.get\_height(y.right))

# Повертаємо новий корінь

return y

def insert(self, root, key):

# Звичайна вставка у двійкове дерево

if not root:

return Node(key)

elif key < root.key:

root.left = self.insert(root.left, key)

else:

root.right = self.insert(root.right, key)

# Оновлюємо висоту вузла

root.height = 1 + max(self.get\_height(root.left), self.get\_height(root.right))

# Обчислюємо баланс вузла

balance = self.get\_balance(root)

# Балансування дерева

# Лівий випадок

if balance > 1 and key < root.left.key:

return self.rotate\_right(root)

# Правий випадок

if balance < -1 and key > root.right.key:

return self.rotate\_left(root)

# Ліво-правий випадок

if balance > 1 and key > root.left.key:

root.left = self.rotate\_left(root.left)

return self.rotate\_right(root)

# Право-лівий випадок

if balance < -1 and key < root.right.key:

root.right = self.rotate\_right(root.right)

return self.rotate\_left(root)

# Повертаємо корінь вузла

return root

def pre\_order(self, root):

if not root:

return

print(f"{root.key} ", end="")

self.pre\_order(root.left)

self.pre\_order(root.right)

# Тестування AVL-дерева

if \_\_name\_\_ == "\_\_main\_\_":

tree = AVLTree()

root = None

# Вставка елементів

keys = [10, 20, 30, 40, 50, 25]

for key in keys:

root = tree.insert(root, key)

# Вивід дерева у прямому обході

print("Pre-order traversal of the AVL tree is:")

tree.pre\_order(root)

print()