**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №1

з дисципліни «Системний аналіз» на тему:

«Застосування системного підходу під час написання програмного коду»

**Виконав:**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли:**

Викладач: Л. Ю. Дейнега

2024

Застосування системного підходу під час написання програмного коду

Мета роботи

Ознайомитись з основними можливостями, парадигмами, типами даних, синтаксичними особливостями та принципами мови програмування Python. Навчитися розробляти програми процедурного, об'єктно-орієнтованого програмування на основі системного підходу.

Результати виконання

Перше завдання

Умова

Файл містить перелік повних адрес файлів (ім'я диску, список каталогів, ім'я файлу та розширення). Виділити з кожної адреси ім'я файлу, розширення та адресу першого каталогу. Перевірити для кожного файлу чи існує він. Вивести у файл, ім'я якого формується з імені початкового файлу додаванням постфіксу "str", перелік файлів, які існують на диску, згрупувавши їх за форматами та відсортувавши в алфавітному порядку за іменем файлів. Імена файлів виводити у форматі "/першийКаталог/.../ім'яФайлу/", сортуючи за розширенням та шляхом.

Програмний код

currentDir = path.dirname(path.abspath(\_\_file\_\_))

inputFilePath = path.join(currentDir, "..", "data", "files.txt")

doUseReadyFile = inquirer.prompt(

    [

        inquirer.List(

            "choice",

            message="Would you like to use a predefined example or enter your own file path?",

            choices=["Predefined Example", "Own File Path"],

        )

    ]

)["choice"]

if doUseReadyFile == "Own File Path":

    inputFilePath = inquirer.prompt(

        [

            inquirer.Text(

                "file path",

                message="Enter your input file path",

                validate=lambda \_, x: "\\" in x or "/" in x and x != "",

            )

        ]

    )["file path"]

outputFilePath = path.join(

    currentDir, "..", "data", inputFilePath.split("\\")[-1][:-4] + "\_str.txt"

)

with open(inputFilePath, "r", encoding="utf-8") as f:

    fileNames = [line.strip() for line in f.readlines()]

filesData = [

    {

        "name": file.split("\\")[-1].split(".")[0],

        "extension": file.split("\\")[-1].split(".")[-1],

        "first\_dir": file.split("\\")[1],

        "does\_exist": path.exists(file),

        "full\_path": file,

    }

    for file in fileNames

]

outputTable = Table(box=box.ROUNDED, title="All Files")

outputTable.add\_column("Index", justify="right", style="cyan", no\_wrap=True)

outputTable.add\_column("File Name", style="green")

outputTable.add\_column("Extension", style="blue")

outputTable.add\_column("First Directory", style="magenta")

outputTable.add\_column("Does Exist", style="red", no\_wrap=True, justify="right")

for i, file in enumerate(filesData):

    outputTable.add\_row(

        f"{i}",

        file["name"],

        file["extension"],

        file["first\_dir"],

        "[green]Yes[/]" if file["does\_exist"] else "No",

    )

console.print("\n", outputTable, "\n")

with console.status("Checking for existing files...", spinner="point"):

    existingFiles = [file for file in filesData if file["does\_exist"]]

    existingFiles.sort(key=lambda x: (x["extension"], x["full\_path"], x["name"]))

if len(existingFiles) == 0:

    console.print(

        "[red]No existing files found. Please check your input file path or contents.[/]\n"

    )

    return

resultsTable = Table(box=box.ROUNDED, title="Existing Files")

resultsTable.add\_column("Index", justify="right", style="cyan", no\_wrap=True)

resultsTable.add\_column("File Name", style="green")

resultsTable.add\_column("Extension", style="blue")

resultsTable.add\_column("File Path", style="yellow")

for i, file in enumerate(existingFiles):

    resultsTable.add\_row(

        f"{i}",

        file["name"],

        file["extension"],

        f"/{file['first\_dir']}/.../{file['name']}.{file['extension']}",

    )

console.print(resultsTable, "\n")

with open(outputFilePath, "w", encoding="utf-8") as f:

    prevExtension = ""

    for file in existingFiles:

        curExtension = file["extension"]

        if curExtension != prevExtension:

            f.write(f"\n{curExtension.upper()}\n")

        f.write(f"{file['name']}.{file['extension']}\n")

        prevExtension = curExtension

Результати виконання



Рисунок 1.1 – Результати роботи програми для завдання один

При виконанні завдання як вхідні дані був використаний файл наступного змісту:

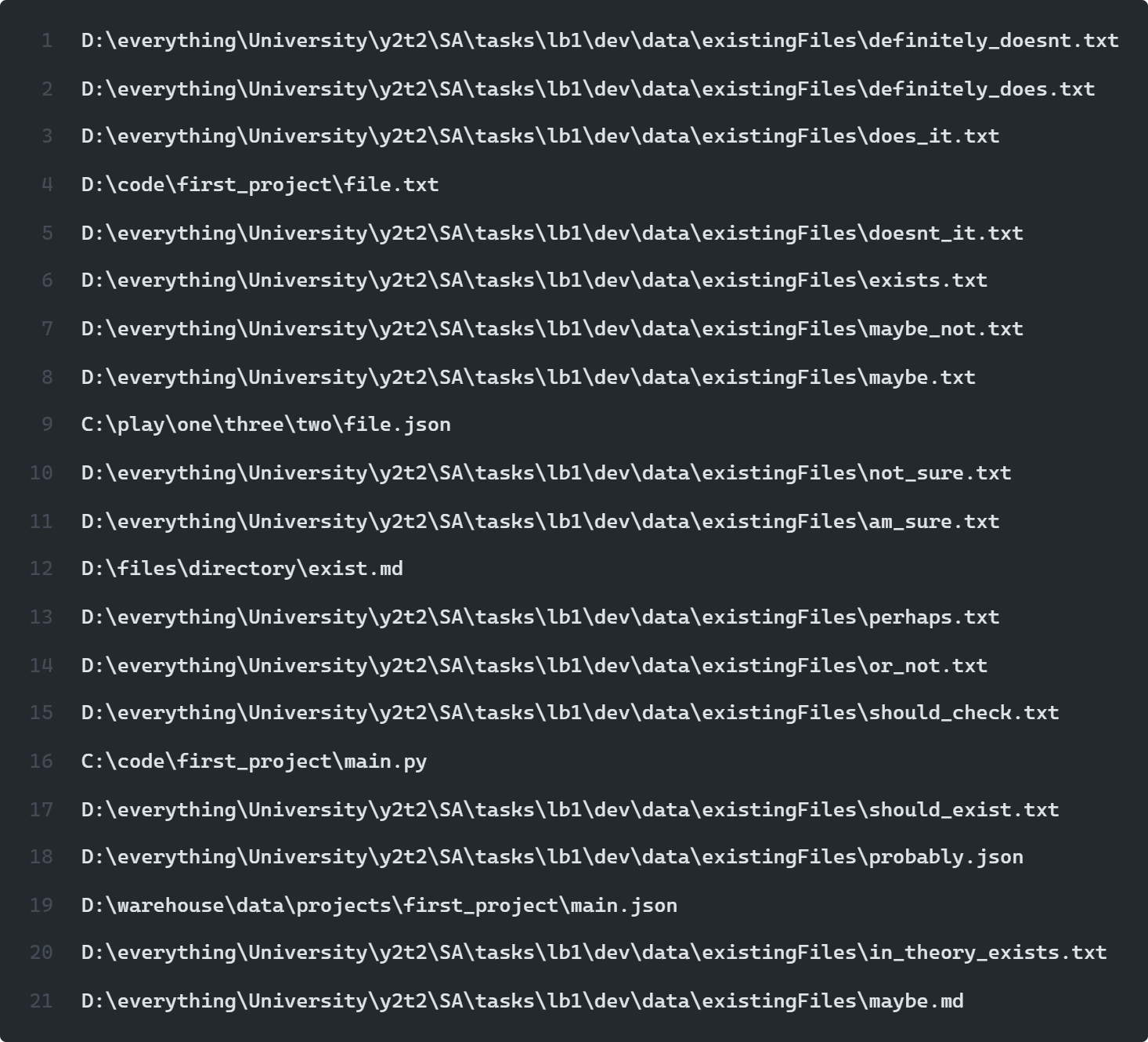


Рисунок 1.2 – Зміст файлу з вхідними даними

На виході в результаті роботи програми було отримано файл наступного змісту:

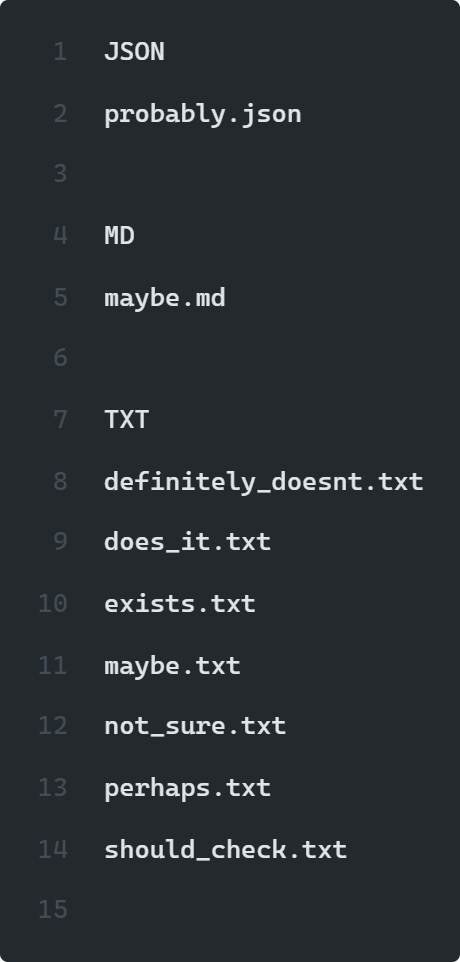


Рисунок 1.3 – Вихідний файл з результатами роботи програми

Друге завдання

Умова

Продаж квитків у кінотеатр з можливістю переглядати сеанси, переглядати доступні та зайняті місця для перегляду заданого сеансу у відповідній залі, бронювання та звільнення місць. Інформація про нові сеанси може додаватися.

Програмний код

class Room:

    def \_\_init\_\_(self, number: int = 0, seats: list[list[int]] = []) -> None:

        self.seats: list[list[int]] = seats

        self.number: int = number

class Movie:

    def \_\_init\_\_(self, title: str = "Movie", rooms: list[Room] = []) -> None:

        self.title: str = title

        self.rooms: list[Room] = rooms

class Ticket:

    def \_\_init\_\_(self, movie: Movie, room: Room, row: int, seat: int) -> None:

        self.movie: Movie = movie

        self.room: Room = room

        self.row: int = row

        self.seat: int = seat

class Cinema:

    def \_\_init\_\_(self, name: str = "Cinema", movies: list[Movie] = []) -> None:

        self.movies: list[Movie] = movies

        self.name: str = name

        self.tickets: list[Ticket] = []

        self.watched: dict = defaultdict(int)

    def buyTicket(self, movie: Movie, room: Room, row: int, seat: int) -> None:

        room.seats[row][seat] = 0

        self.tickets.append(Ticket(movie, room, row, seat))

    def sellTicket(self, ticket: Ticket) -> None:

        ticket.room.seats[ticket.row][ticket.seat] = 1

        self.tickets.remove(ticket)

    def readMoviesFromJson(self, path: str = "") -> None:

        with open(path, "r", encoding="utf-8") as f:

            data = json.load(f)

        self.name = data["name"]

        for movieData in data["movies"]:

            rooms = []

            for room in movieData["rooms"]:

                rooms.append(Room(number=room["number"], seats=room["seats"]))

            self.movies.append(Movie(title=movieData["title"], rooms=rooms))

def drawSeatsGrid(room: Room, movie: Movie) -> None:

    seats = [["🔴" if not seat else "🟢" for seat in row] for row in room.seats]

    table: Table = Table.grid(padding=(1, 1))

    for i, row in enumerate(seats):

        table.add\_row(\*[f"[bold]{i+1}[/]"] + row)

    console.print(

        f'[bold]All seats for "{movie.title}" in room #{room.number}[/bold]\n',

        table,

        "\n",

    )

def drawTicketsTable(tickets: list[Ticket]) -> None:

    table: Table = Table(box=box.ROUNDED, title="Bought Tickets")

    table.add\_column("Index", justify="right", style="cyan", no\_wrap=True)

    table.add\_column("Movie", style="bold yellow")

    table.add\_column("Year", style="violet", no\_wrap=True)

    table.add\_column("Room", style="magenta", no\_wrap=True)

    table.add\_column("Row", style="green", no\_wrap=True)

    table.add\_column("Seat", style="blue", no\_wrap=True)

    for i, ticket in enumerate(tickets):

        name, year = ticket.movie.title.split(" - ")

        table.add\_row(

            f"{i+1}",

            f"{name}",

            f"{year}",

            f"{ticket.room.number}",

            f"{ticket.row+1}",

            f"{ticket.seat+1}",

        )

    console.print(table, "\n")

def drawMoviesTable() -> None:

    table: Table = Table(box=box.ROUNDED, title="Watched Movies")

    table.add\_column("Index", style="cyan", no\_wrap=True, justify="right")

    table.add\_column("Movie", style="bold yellow", no\_wrap=True)

    table.add\_column("Year", style="violet", no\_wrap=True)

    table.add\_column("Times Watched", style="magenta", no\_wrap=True, justify="left")

    for i, (movie, count) in enumerate(cinema.watched.items()):

        name, year = movie.split(" - ")

        table.add\_row(

            f"{i+1}",

            f"{name}",

            f"{year}",

            f"{count}",

        )

    console.print(table, "\n")

currentDir: str = path.dirname(path.abspath(\_\_file\_\_))

moviesDataPath: str = path.join(currentDir, "..", "data", "movies.json")

cinema: Cinema = Cinema()

cinema.readMoviesFromJson(moviesDataPath)

while True:

    actions: list[str] = [

        "Browse Tickets",

        "View Bought Tickets",

        "View Watched Movies",

        "Add New Movie",

        "Exit",

    ]

    action = inquirer.prompt(

        [

            inquirer.List(

                "action",

                message="What would you like to do?",

                choices=actions,

            )

        ]

    )["action"]

    if action == "Browse Tickets":

        if not cinema.movies:

            console.print("[bold]No movies available[/bold]\n")

        movie = inquirer.prompt(

            [

                inquirer.List(

                    "movie",

                    message="Which movie would you like to watch?",

                    choices=[movie.title for movie in cinema.movies],

                )

            ]

        )["movie"]

        movie = [movie.title for movie in cinema.movies].index(movie)

        movie = cinema.movies[movie]

        room = inquirer.prompt(

            [

                inquirer.List(

                    "room",

                    message="Choose a room (hall) where you would like to watch the movie",

                    choices=range(1, len(movie.rooms) + 1),

                )

            ]

        )["room"]

        room = movie.rooms[room - 1]

        drawSeatsGrid(room, movie)

        availableRows = [i + 1 for i, row in enumerate(room.seats) if True in row]

        row = inquirer.prompt(

            [

                inquirer.List(

                    "row",

                    message="Choose a seat row where you would like to watch the movie",

                    choices=availableRows,

                )

            ]

        )["row"]

        row = row - 1

        seats = [i + 1 for i, seat in enumerate(room.seats[row]) if seat]

        seat = inquirer.prompt(

            [

                inquirer.List(

                    "seat",

                    message="Choose a seat where you would like to watch the movie",

                    choices=seats,

                )

            ]

        )["seat"]

        seat = seat - 1

        confirmation = inquirer.prompt(

            [

                inquirer.Confirm(

                    "confirm",

                    message=f'Are you sure you want to buy a ticket for "{movie.title}" in room {room.number} row {row+1} seat {seat+1}?',

                    default=True,

                )

            ]

        )

        if confirmation["confirm"]:

            cinema.buyTicket(movie, room, row, seat)

            console.print("\n[bold]Ticket bought![/bold]\n")

        drawSeatsGrid(room, movie)

    elif action == "View Bought Tickets":

        if not cinema.tickets:

            console.print("[bold]No bought tickets found[/bold]\n")

            continue

        drawTicketsTable(cinema.tickets)

        indeces = range(1, len(cinema.tickets) + 1)

        ticketIndex = inquirer.prompt(

            [

                inquirer.List(

                    "ticket",

                    message="Choose a ticket number",

                    choices=indeces,

                )

            ]

        )["ticket"]

        ticketIndex = ticketIndex - 1

        action = inquirer.prompt(

            [

                inquirer.List(

                    "action",

                    message="What would you like to do?",

                    choices=["Watch movie", "Return ticket", "Exit"],

                )

            ]

        )["action"]

        ticketData = cinema.tickets[ticketIndex]

        movieName = ticketData.movie.title

        if action == "Return ticket":

            cinema.sellTicket(ticketData)

            console.print(f'[bold]"{movieName}"[/bold] was returned!\n')

        elif action == "Watch movie":

            cinema.watched[movieName] += 1

            console.print(f'[bold]"{movieName}"[/bold] was watched!\n')

            cinema.tickets.remove(ticketData)

    elif action == "View Watched Movies":

        if not cinema.watched:

            console.print("[bold]No watched movies found[/bold]\n")

            continue

        drawMoviesTable()

    elif action == "Add New Movie":

        title: str = inquirer.prompt(

            [

                inquirer.Text(

                    "title",

                    message="Enter movie title. Make sure to separate the title from the year with a dash (-)",

                    validate=lambda \_, x: x != "" and "-" in x,

                )

            ]

        )["title"]

        title, year = title.split("-")

        title, year = title.strip(), year.strip()

        numOfRooms: int = inquirer.prompt(

            [

                inquirer.Text(

                    "rooms",

                    message="Enter number of available rooms",

                    validate=lambda \_, x: x != "" and x.isdigit(),

                )

            ]

        )["rooms"]

        numOfRooms = int(numOfRooms)

        seats = [

            [[random.choice([0, 1]) for \_ in range(10)] for \_ in range(5)]

            for \_ in range(numOfRooms)

        ]

        rooms: list[Room] = []

        for i in range(1, numOfRooms + 1):

            rooms.append(Room(i, seats[i - 1]))

        console.print(f'\n[bold]"{title} - {year}"[/bold] has been added!\n')

        movie = Movie(f"{title} - {year}", rooms)

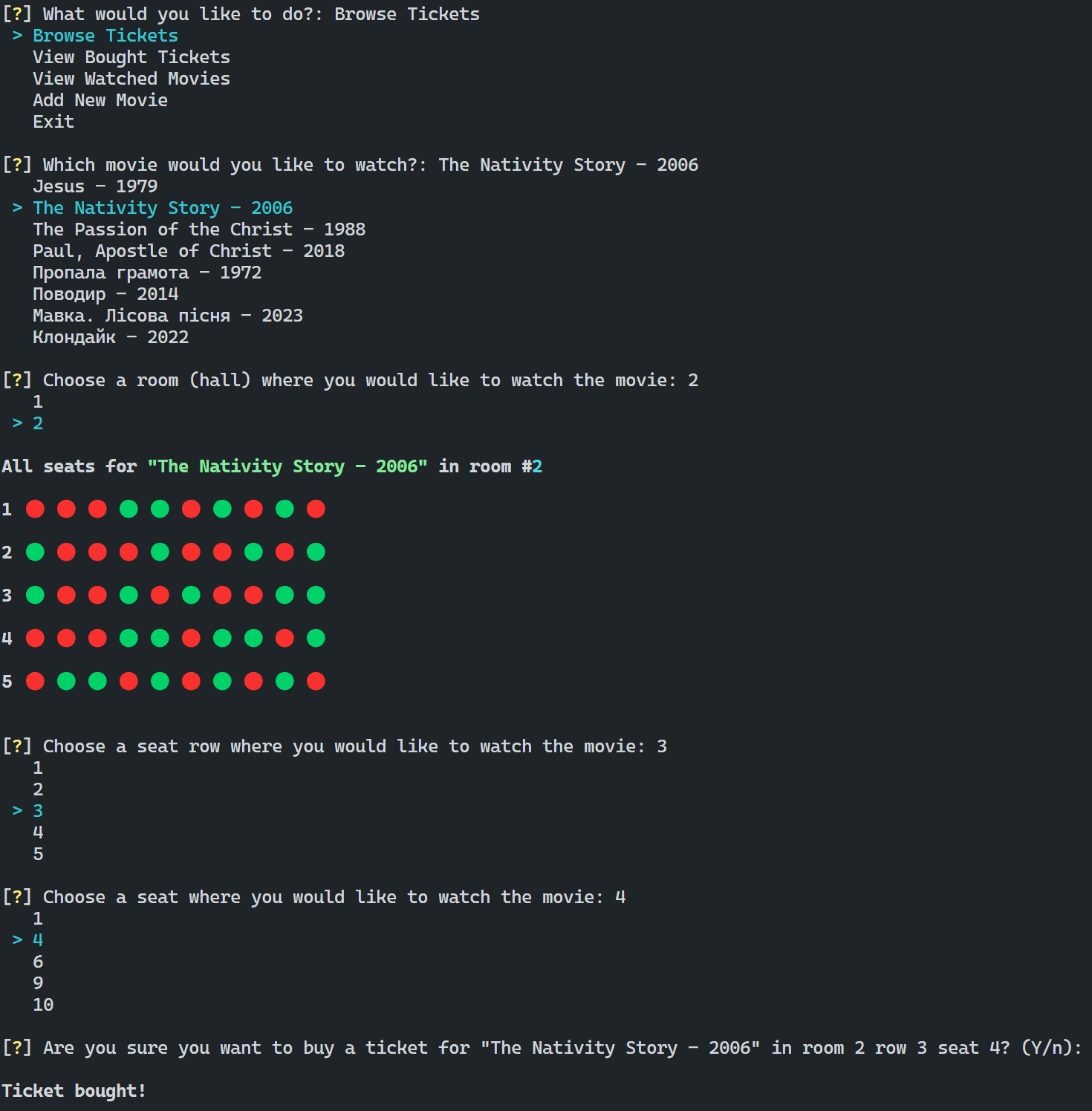
        cinema.movies.append(movie)

    else:

        break

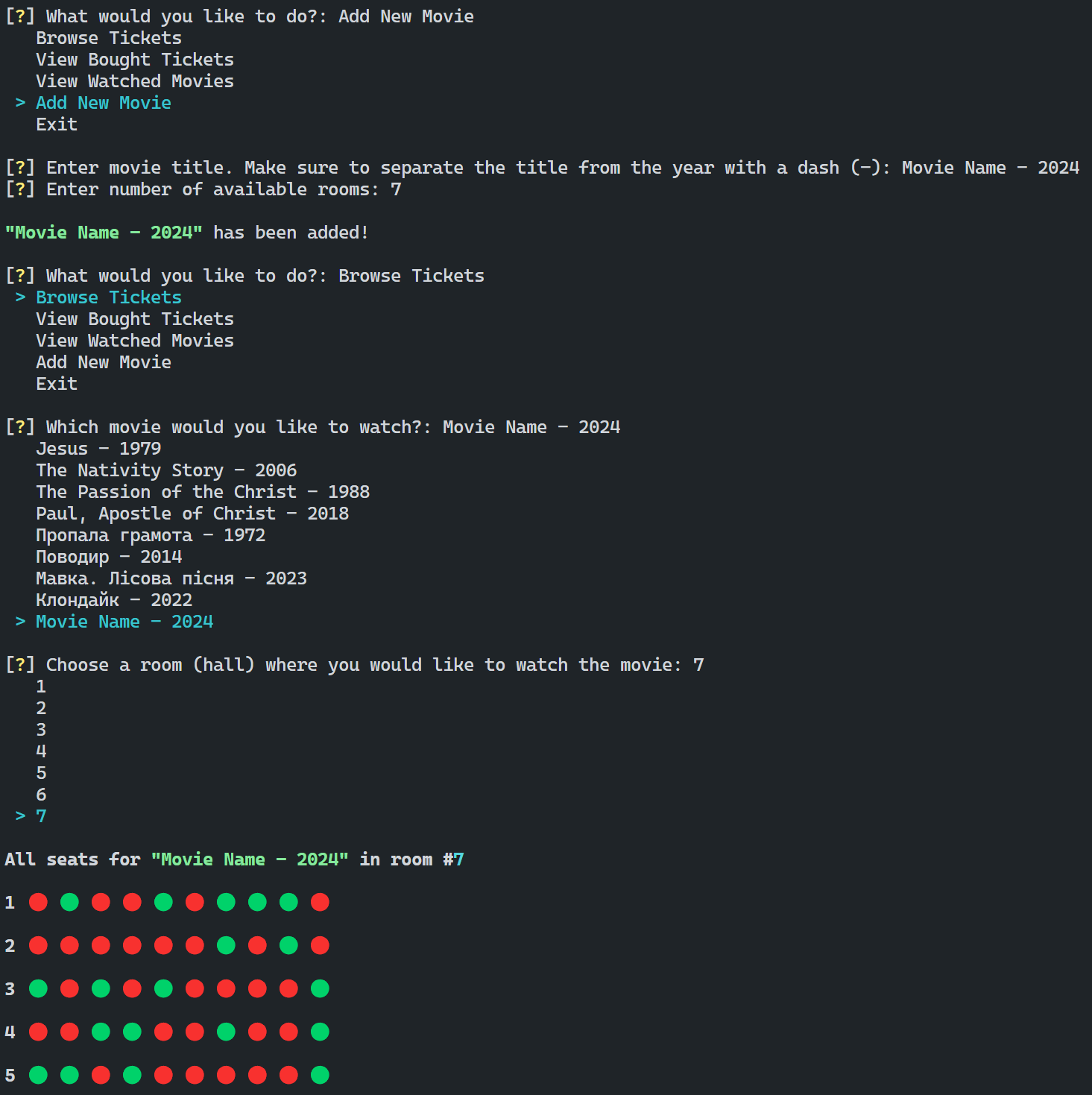
Результати виконання

Результати виконання програми наведені нижче у вигляді знімків з екрану:









Як вхідні дані використовувався файл JSON наступного змісту:

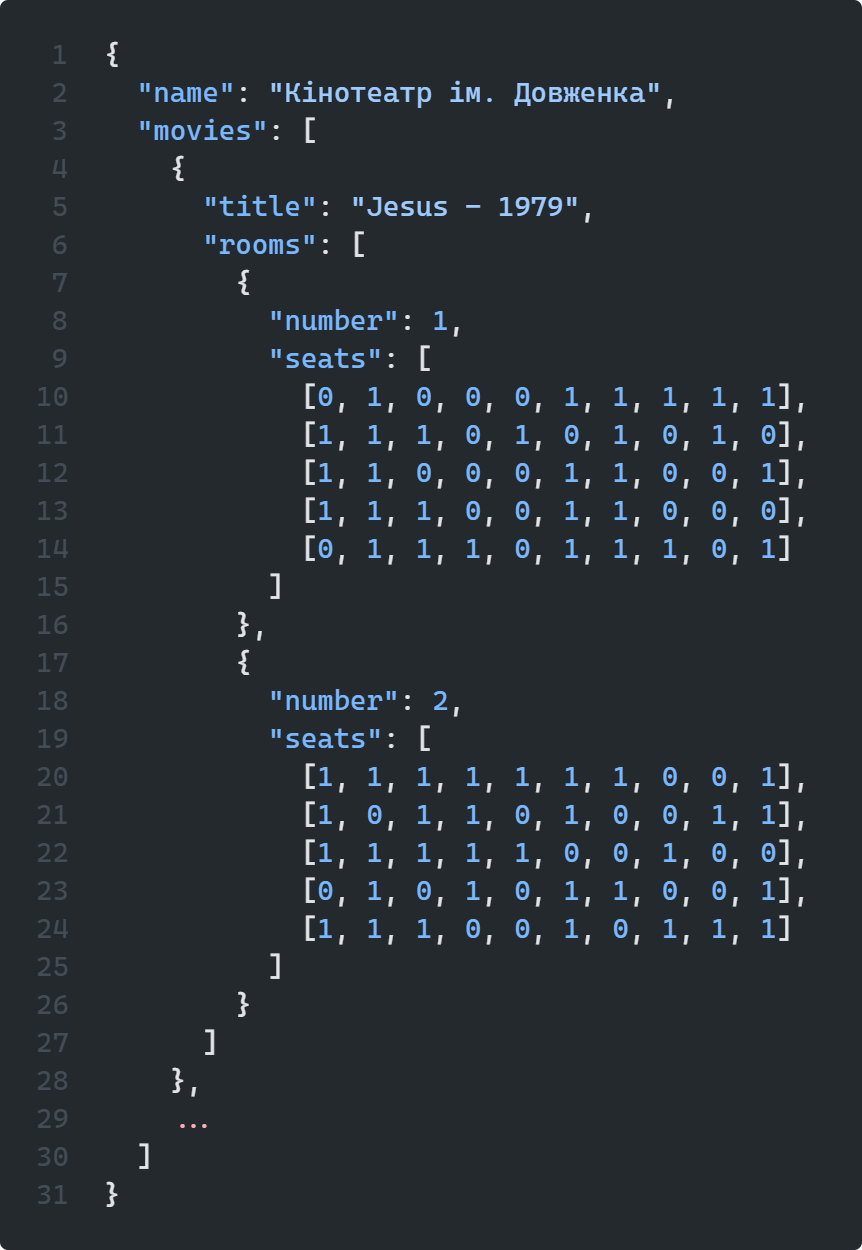


Рисунок 2.2 – Файл вхідних даних

Висновки

Таким чином, ми ознайомилися з основними можливостями, парадигмами, типами даних, синтаксичними особливостями та принципами мови програмування Python, а також навчилися розробляти програми процедурного, об'єктно-орієнтованого програмування на основі системного підходу.

Контрольні питання

Що таке системний аналіз?

Системний аналіз – це підхід до аналізу систем у різних предметних областях з метою визначення їхньої структури та шляхів покращення цих систем. Системи, своєю чергою, це набори взаємопов’язаних але відокремлених один від одного компонентів, які, працюючи разом, формують цілісний «організм», що виконує поставлену задачу.

Які принципи системного підходу?

Системний підхід передбачає принципи декомпозиції, цілісності, ітеративності, зворотного зв’язку та обмежень.

Принцип декомпозиції – це огляд кожної частини системи окремо задля розуміння її задачі та зв’язків з іншими компонентами.

Принцип цілісності передбачає врахування контексту цілісної системи задля кращого бачення цілісної картини роботи системи.

Принцип ітеративності передбачає ітеративний процес розробки та покращення системи задля запобігання помилок та неточностей в розробці.

Принцип зворотного зв’язку передбачає отримання зворотного зв’язку від замовника або керівника проєкту на кожній ітерації розробки та аналізу системи задля узгодження очікувань та забезпечення постійного покращення системи.

Принцип обмежень передбачає врахування системних або будь-яких інших наявних обмежень задля аби забезпечити очікувану продуктивність системи, а також задовольнити потребам та вимогам замовників/керівників.

Який інструментарій може використовуватися для розроблення програм мовою Python?

Інструментарій для розробки програм мовою Python це, перш за все, вбудовані інструменти мови, а також сторонні бібліотеки чи пакети. До вбудований засобів відносяться синтаксичні особливості мови, вбудовані функції та стандартні вбудовані бібліотеки мови Python. Розробники також часто використовують Інтегровані Середовища Розробки, які полегшують процес написання та відлагодження коду. Серед інших не менш корисних інструментів можна зазначити системи контролю версій, такі як Git чи SVN, та менеджери пакетів, такі як pip чи conda.

Які парадигми програмування підтримуються мовою Python?

Мова Python підтримує різноманітні парадигми програмування, але сама мова є перш за все об’єктно-орієнтованою. Інші парадигми програмування включать процедурне та функціональне, що можуть також забезпечуватися шляхом використання сторонніх бібліотек, як наприклад itertools або functools.

Які принципи лежать в основі програмування мовою Python?

В основі програмування мовою Python лежать принципи читабельності, простоти, можливості багаторазового використання, та гнучкості. Ці принципи також можуть бути покращені за допомогою рекомендацій, наданих PEP або Python Enhancement Proposals, в яких перераховано багато різних найкращих практик та інструкцій для написання якіснішого та зручнішого для супроводу коду мовою Python.

Для яких цілей може використовуватися мова Python?

Мова Python може бути використана для найбільш різноманітних цілей розробника, але передовими галузями її застосування на сьогодні є такі сфери як машинне навчання, штучний інтелект, аналіз даних, тощо. Мова Python є повільнішою в порівняння з іншими С-подібними аналогами, але за рахунок охайного синтаксису та безлічі різноманітних бібліотек вона залишається чудовим вибором для розробників у різних галузях програмних засобів.

Які основні типи даних мови програмування Python

Мова Python, хоч і динамічно типізована, але включає наступні типи даних: цілі числа, числа з плаваючою комою, рядки, булеві вирази, списки, кортежі, словники, множини.

Чим відрізняється динамічна типізація від статичної?

При динамічній типізації, на відміну від статичної, у розробника немає необхідності явно визначати тип даних, хоч така можливість і є у мові Python у вигляді типових анотацій. Натомість інтерпретатор Python автоматично визначає тип даних змінної і дозволяє зміну типу даних у процесі роботи програми.

Чим відрізняється імперативна парадигма програмування від декларативної?

Імперативна парадигма програмування передбачає зазначення конкретного набору дій для виконання програми шляхом безпосереднього написання коду, який виконуватиме задачу крок за кроком.

Декларативна парадигма програмування передбачає зазначення фінальної цілі програми, але без перерахування конкретних кроків її виконання.

Які типи даних у Python є незмінними та яким чином змінити їх значення?

До незмінних типів даних належать: int, float, complex (комплексне число), string (str), tuple (кортеж). Щоб змінити їх, необхідно або створити іншу змінну з потрібними значеннями, або переприсвоїти нові значення бажаній змінній.

Які типи даних у Python можуть змінюватися?

Змінними типами даних у Python є: list (масив), dict (словник), set (множина).

Яким чином виконати форматування рядка?

Для форматування рядка можна застосувати засіб мови Python f-рядки. Цей тип рядків дозволяє вбудування змінних у рядок а також зміну форматування, наприклад числа:

num: float = math.pi

print(f"{num:.2f}")

# Вивід - 3.14

Що таке перелік та чим він відрізняється від інших типів даних?

Перелік у Python - це впорядкована колекція елементів, які можуть змінюватися. Він відрізняється від інших типів тим, що дозволяє зберігати декілька елементів в одній змінній, і його можна змінювати після створення. Переліки позначаються квадратними дужками і можуть містити елементи різних типів даних. Ця гнучкість і можливість зміни відрізняє списки від інших типів даних, таких як кортежі, які є незмінними, і множини, які не допускають повторення елементів і є невпорядкованими.

Які дії можна виконувати над переліками?

Над переліками в Python можна виконувати такі дії, як додавання елементів, видалення елементів, доступ до елементів за індексом або зрізом, модифікація елементів, сортування та обернення.

Що таке множина та яким чином її визначити?

У Python множина - це невпорядкована колекція унікальних елементів. Її можна описати за допомогою фігурних дужок { } з елементами, розділеними комами.

Що таке словник та чим він відрізняється від інших типів даних?

У Python словник - це невпорядкована колекція пар ключ-значення. Він відрізняється від інших типів даних тим, що використовується для зберігання та пошуку даних за унікальним ключем, а не за індексом чи позицією. Це дозволяє здійснювати ефективний пошук і вилучення значень на основі пов'язаних з ними ключів.

Які дії можна виконувати над словниками?

У Python над словниками можна виконувати такі дії, як додавання та оновлення елементів, видалення елементів, доступ до елементів за ключем, а також такі операції, як перевірка існування ключа, копіювання та об'єднання словників.

Що таке кортеж та у яких випадках його необхідно використовувати?

У Python кортеж - це незмінна та впорядкована колекція елементів. Його слід використовувати, коли дані є фіксованими і не повинні змінюватися, наприклад, координати, записи в базі даних або будь-який інший набір даних, де важлива незмінність і впорядкованість. Кортежі позначаються круглими дужками ( ) і можуть містити елементи різних типів даних.

Яким чином визначається цикл з постумовою у мові Python?

У мові Python цикл з постумовою визначається за допомогою циклу while. Цикл продовжує виконуватися до тих пір, поки вказана умова є істинною після того, як тіло циклу було виконано хоча б один раз. Це гарантує, що умова циклу перевіряється після виконання тіла циклу, таким чином створюючи цикл з постумовою.

Яким чином визначити у Python аналог оператора switch у C/C++?

У Python аналог оператора switch з C/C++ може бути досягнутий за допомогою серії операторів if, elif та else для перевірки різних варіантів. Структуруючи серію умовних операторів, кожен з яких перевіряє певний варіант, можна емулювати поведінку, подібну до оператора switch, у Python.

Яким чином визначити рекурсію в Python?

У Python рекурсія визначається, коли функція викликає сама себе в межах власного визначення. Визначення функції повинно містити умову завершення, щоб запобігти нескінченній рекурсії.

Чим відрізняється синтаксис мови Python від C/C++?

Синтаксис Python та C/C++ дещо відрізняється. Python використовує відступи для позначення блоків коду, тоді як C/C++ використовує фігурні дужки. Python має динамічну типізацію, тобто типи змінних визначаються під час виконання, тоді як C/C++ має статичну типізацію, що вимагає явного оголошення типів. Крім того, Python має автоматичне керування пам'яттю, тоді як C/C++ вимагає ручного виділення та звільнення пам'яті.

Яким чином виконується робота з файлами в Python?

У мові Python робота з файлами здійснюється за допомогою вбудованих функцій, таких як open(), read(), write(), close() та інших пов'язаних з ними методів. Для роботи з файлами спочатку створюється файловий об'єкт за допомогою функції open(), де вказується ім'я файлу та режим (наприклад, read, write, append). Потім файловий об'єкт можна використовувати для читання з файлу або запису до нього, а після завершення роботи з файлом його слід закрити за допомогою методу close().

Які існують області видимості в Python?

У Python є чотири основні області видимості: вбудована область видимості, глобальна область видимості, охоплююча (нелокальна) область видимості та локальна область видимості. Вбудована область видимості містить вбудовані функції та виключення Python. Глобальна область видимості відноситься до змінних, оголошених поза будь-якою функцією або класом. Охоплююча (нелокальна) область видимості стосується вкладених функцій і відноситься до змінних в області видимості охоплюючої функції. Локальна область видимості відноситься до змінних, оголошених у функції або методі і доступних тільки в межах цієї функції або методу.

Яким чином згенерувати перелік або словник?

У Python можна створити список за допомогою зчислення списків, яке забезпечує лаконічний спосіб створення списків на основі існуючих послідовностей. Наприклад, можна використати зчислення списку для створення списку квадратів: [x\*\*2 for x in range(10)]. Аналогічно, за допомогою зчислення словника можна створити словник. Наприклад, за допомогою стислого синтаксису можна створити словник: {x: x\*\*2 for x in range(10)}.

Яким чином виконується обробка виключень?

У Python винятки обробляються за допомогою блоків try-except. Код, який може згенерувати виключення, розміщується у блоці try, а потенційні виключення перехоплюються та обробляються у блоці except. Якщо виняток виникає у блоці try, потік програми переноситься до відповідного блоку except, що дозволяє виконати певні дії з обробки помилок або відновлення. Крім того, необов'язковий блок finally можна використовувати для визначення коду, який завжди буде виконуватися, незалежно від того, виникне виняток чи ні.

Яким чином застосовується системний підхід під час створення коду?

Системний підхід застосовується при створенні коду шляхом дотримання найкращих практик, таких як розбиття проблеми на менші, керовані завдання, планування загальної структури та дизайну коду, а також використання таких інструментів, як псевдокод, блок-схеми або UML-діаграми для відображення логіки та функціональності. Крім того, використання модульних і багаторазових компонентів коду, написання чітких і описових коментарів, а також проведення ретельного тестування і валідації сприяють систематичному і організованому підходу до створення коду.

Додаток А – Повний програмний код

import json

import random

import inquirer

from os import path

from rich import box

from rich.table import Table

from rich.console import Console

from rich.traceback import install

from collections import defaultdict

install()

console = Console()

def taskOne() -> None:

    currentDir = path.dirname(path.abspath(\_\_file\_\_))

    inputFilePath = path.join(currentDir, "..", "data", "files.txt")

    doUseReadyFile = inquirer.prompt(

        [

            inquirer.List(

                "choice",

                message="Would you like to use a predefined example or enter your own file path?",

                choices=["Predefined Example", "Own File Path"],

            )

        ]

    )["choice"]

    if doUseReadyFile == "Own File Path":

        inputFilePath = inquirer.prompt(

            [

                inquirer.Text(

                    "file path",

                    message="Enter your input file path",

                    validate=lambda \_, x: "\\" in x or "/" in x and x != "",

                )

            ]

        )["file path"]

    outputFilePath = path.join(

        currentDir, "..", "data", inputFilePath.split("\\")[-1][:-4] + "\_str.txt"

    )

    with open(inputFilePath, "r", encoding="utf-8") as f:

        fileNames = [line.strip() for line in f.readlines()]

    filesData = [

        {

            "name": file.split("\\")[-1].split(".")[0],

            "extension": file.split("\\")[-1].split(".")[-1],

            "first\_dir": file.split("\\")[1],

            "does\_exist": path.exists(file),

            "full\_path": file,

        }

        for file in fileNames

    ]

    outputTable = Table(box=box.ROUNDED, title="All Files")

    outputTable.add\_column("Index", justify="right", style="cyan", no\_wrap=True)

    outputTable.add\_column("File Name", style="green")

    outputTable.add\_column("Extension", style="blue")

    outputTable.add\_column("First Directory", style="magenta")

    outputTable.add\_column("Does Exist", style="red", no\_wrap=True, justify="right")

    for i, file in enumerate(filesData):

        outputTable.add\_row(

            f"{i}",

            file["name"],

            file["extension"],

            file["first\_dir"],

            "[green]Yes[/]" if file["does\_exist"] else "No",

        )

    console.print("\n", outputTable, "\n")

    with console.status("Checking for existing files...", spinner="point"):

        existingFiles = [file for file in filesData if file["does\_exist"]]

        existingFiles.sort(key=lambda x: (x["extension"], x["full\_path"], x["name"]))

    if len(existingFiles) == 0:

        console.print(

            "[red]No existing files found. Please check your input file path or contents.[/]\n"

        )

        return

    resultsTable = Table(box=box.ROUNDED, title="Existing Files")

    resultsTable.add\_column("Index", justify="right", style="cyan", no\_wrap=True)

    resultsTable.add\_column("File Name", style="green")

    resultsTable.add\_column("Extension", style="blue")

    resultsTable.add\_column("File Path", style="yellow")

    for i, file in enumerate(existingFiles):

        resultsTable.add\_row(

            f"{i}",

            file["name"],

            file["extension"],

            f"/{file['first\_dir']}/.../{file['name']}.{file['extension']}",

        )

    console.print(resultsTable, "\n")

    with open(outputFilePath, "w", encoding="utf-8") as f:

        prevExtension = ""

        for file in existingFiles:

            curExtension = file["extension"]

            if curExtension != prevExtension:

                f.write(f"\n{curExtension.upper()}\n")

            f.write(f"{file['name']}.{file['extension']}\n")

            prevExtension = curExtension

def taskTwo() -> None:

    class Room:

        def \_\_init\_\_(self, number: int = 0, seats: list[list[int]] = []) -> None:

            self.seats: list[list[int]] = seats

            self.number: int = number

    class Movie:

        def \_\_init\_\_(self, title: str = "Movie", rooms: list[Room] = []) -> None:

            self.title: str = title

            self.rooms: list[Room] = rooms

    class Ticket:

        def \_\_init\_\_(self, movie: Movie, room: Room, row: int, seat: int) -> None:

            self.movie: Movie = movie

            self.room: Room = room

            self.row: int = row

            self.seat: int = seat

    class Cinema:

        def \_\_init\_\_(self, name: str = "Cinema", movies: list[Movie] = []) -> None:

            self.movies: list[Movie] = movies

            self.name: str = name

            self.tickets: list[Ticket] = []

            self.watched: dict = defaultdict(int)

        def buyTicket(self, movie: Movie, room: Room, row: int, seat: int) -> None:

            room.seats[row][seat] = 0

            self.tickets.append(Ticket(movie, room, row, seat))

        def sellTicket(self, ticket: Ticket) -> None:

            ticket.room.seats[ticket.row][ticket.seat] = 1

            self.tickets.remove(ticket)

        def readMoviesFromJson(self, path: str = "") -> None:

            with open(path, "r", encoding="utf-8") as f:

                data = json.load(f)

            self.name = data["name"]

            for movieData in data["movies"]:

                rooms = []

                for room in movieData["rooms"]:

                    rooms.append(Room(number=room["number"], seats=room["seats"]))

                self.movies.append(Movie(title=movieData["title"], rooms=rooms))

    def drawSeatsGrid(room: Room, movie: Movie) -> None:

        seats = [["🔴" if not seat else "🟢" for seat in row] for row in room.seats]

        table: Table = Table.grid(padding=(1, 1))

        for i, row in enumerate(seats):

            table.add\_row(\*[f"[bold]{i+1}[/]"] + row)

        console.print(

            f'[bold]All seats for "{movie.title}" in room #{room.number}[/bold]\n',

            table,

            "\n",

        )

    def drawTicketsTable(tickets: list[Ticket]) -> None:

        table: Table = Table(box=box.ROUNDED, title="Bought Tickets")

        table.add\_column("Index", justify="right", style="cyan", no\_wrap=True)

        table.add\_column("Movie", style="bold yellow")

        table.add\_column("Year", style="violet", no\_wrap=True)

        table.add\_column("Room", style="magenta", no\_wrap=True)

        table.add\_column("Row", style="green", no\_wrap=True)

        table.add\_column("Seat", style="blue", no\_wrap=True)

        for i, ticket in enumerate(tickets):

            name, year = ticket.movie.title.split(" - ")

            table.add\_row(

                f"{i+1}",

                f"{name}",

                f"{year}",

                f"{ticket.room.number}",

                f"{ticket.row+1}",

                f"{ticket.seat+1}",

            )

        console.print(table, "\n")

    def drawMoviesTable() -> None:

        table: Table = Table(box=box.ROUNDED, title="Watched Movies")

        table.add\_column("Index", style="cyan", no\_wrap=True, justify="right")

        table.add\_column("Movie", style="bold yellow", no\_wrap=True)

        table.add\_column("Year", style="violet", no\_wrap=True)

        table.add\_column("Times Watched", style="magenta", no\_wrap=True, justify="left")

        for i, (movie, count) in enumerate(cinema.watched.items()):

            name, year = movie.split(" - ")

            table.add\_row(

                f"{i+1}",

                f"{name}",

                f"{year}",

                f"{count}",

            )

        console.print(table, "\n")

    currentDir: str = path.dirname(path.abspath(\_\_file\_\_))

    moviesDataPath: str = path.join(currentDir, "..", "data", "movies.json")

    cinema: Cinema = Cinema()

    cinema.readMoviesFromJson(moviesDataPath)

    while True:

        actions: list[str] = [

            "Browse Tickets",

            "View Bought Tickets",

            "View Watched Movies",

            "Add New Movie",

            "Exit",

        ]

        action = inquirer.prompt(

            [

                inquirer.List(

                    "action",

                    message="What would you like to do?",

                    choices=actions,

                )

            ]

        )["action"]

        if action == "Browse Tickets":

            if not cinema.movies:

                console.print("[bold]No movies available[/bold]\n")

            movie = inquirer.prompt(

                [

                    inquirer.List(

                        "movie",

                        message="Which movie would you like to watch?",

                        choices=[movie.title for movie in cinema.movies],

                    )

                ]

            )["movie"]

            movie = [movie.title for movie in cinema.movies].index(movie)

            movie = cinema.movies[movie]

            room = inquirer.prompt(

                [

                    inquirer.List(

                        "room",

                        message="Choose a room (hall) where you would like to watch the movie",

                        choices=range(1, len(movie.rooms) + 1),

                    )

                ]

            )["room"]

            room = movie.rooms[room - 1]

            drawSeatsGrid(room, movie)

            availableRows = [i + 1 for i, row in enumerate(room.seats) if True in row]

            row = inquirer.prompt(

                [

                    inquirer.List(

                        "row",

                        message="Choose a seat row where you would like to watch the movie",

                        choices=availableRows,

                    )

                ]

            )["row"]

            row = row - 1

            seats = [i + 1 for i, seat in enumerate(room.seats[row]) if seat]

            seat = inquirer.prompt(

                [

                    inquirer.List(

                        "seat",

                        message="Choose a seat where you would like to watch the movie",

                        choices=seats,

                    )

                ]

            )["seat"]

            seat = seat - 1

            confirmation = inquirer.prompt(

                [

                    inquirer.Confirm(

                        "confirm",

                        message=f'Are you sure you want to buy a ticket for "{movie.title}" in room {room.number} row {row+1} seat {seat+1}?',

                        default=True,

                    )

                ]

            )

            if confirmation["confirm"]:

                cinema.buyTicket(movie, room, row, seat)

                console.print("\n[bold]Ticket bought![/bold]\n")

            drawSeatsGrid(room, movie)

        elif action == "View Bought Tickets":

            if not cinema.tickets:

                console.print("[bold]No bought tickets found[/bold]\n")

                continue

            drawTicketsTable(cinema.tickets)

            indeces = range(1, len(cinema.tickets) + 1)

            ticketIndex = inquirer.prompt(

                [

                    inquirer.List(

                        "ticket",

                        message="Choose a ticket number",

                        choices=indeces,

                    )

                ]

            )["ticket"]

            ticketIndex = ticketIndex - 1

            action = inquirer.prompt(

                [

                    inquirer.List(

                        "action",

                        message="What would you like to do?",

                        choices=["Watch movie", "Return ticket", "Exit"],

                    )

                ]

            )["action"]

            ticketData = cinema.tickets[ticketIndex]

            movieName = ticketData.movie.title

            if action == "Return ticket":

                cinema.sellTicket(ticketData)

                console.print(f'[bold]"{movieName}"[/bold] was returned!\n')

            elif action == "Watch movie":

                cinema.watched[movieName] += 1

                console.print(f'[bold]"{movieName}"[/bold] was watched!\n')

                cinema.tickets.remove(ticketData)

        elif action == "View Watched Movies":

            if not cinema.watched:

                console.print("[bold]No watched movies found[/bold]\n")

                continue

            drawMoviesTable()

        elif action == "Add New Movie":

            title: str = inquirer.prompt(

                [

                    inquirer.Text(

                        "title",

                        message="Enter movie title. Make sure to separate the title from the year with a dash (-)",

                        validate=lambda \_, x: x != "" and "-" in x,

                    )

                ]

            )["title"]

            title, year = title.split("-")

            title, year = title.strip(), year.strip()

            numOfRooms: int = inquirer.prompt(

                [

                    inquirer.Text(

                        "rooms",

                        message="Enter number of available rooms",

                        validate=lambda \_, x: x != "" and x.isdigit(),

                    )

                ]

            )["rooms"]

            numOfRooms = int(numOfRooms)

            seats = [

                [[random.choice([0, 1]) for \_ in range(10)] for \_ in range(5)]

                for \_ in range(numOfRooms)

            ]

            rooms: list[Room] = []

            for i in range(1, numOfRooms + 1):

                rooms.append(Room(i, seats[i - 1]))

            console.print(f'\n[bold]"{title} - {year}"[/bold] has been added!\n')

            movie = Movie(f"{title} - {year}", rooms)

            cinema.movies.append(movie)

        else:

            break

def main() -> None:

    availableTasks = [

        "First - Checking Files",

        "Second - Cinema Tickets",

    ]

    selectedTask = inquirer.prompt(

        [

            inquirer.List(

                "task",

                message="Which task would you like to look at?",

                choices=availableTasks,

            )

        ]

    )["task"]

    if selectedTask == availableTasks[0]:

        taskOne()

    elif selectedTask == availableTasks[1]:

        taskTwo()

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

    main()