**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:

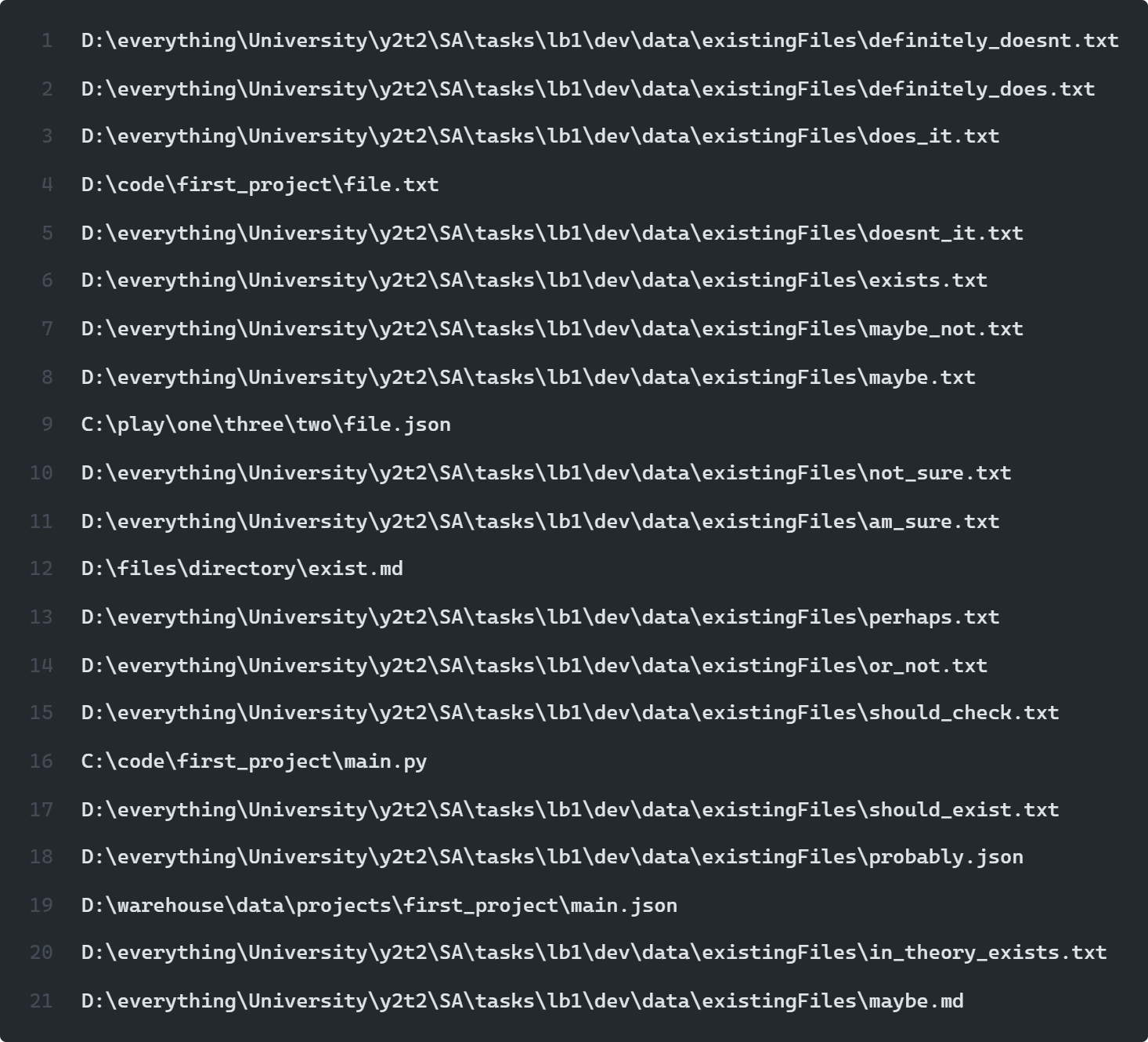


Рисунок 1.2 – Зміст файлу з вхідними даними

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

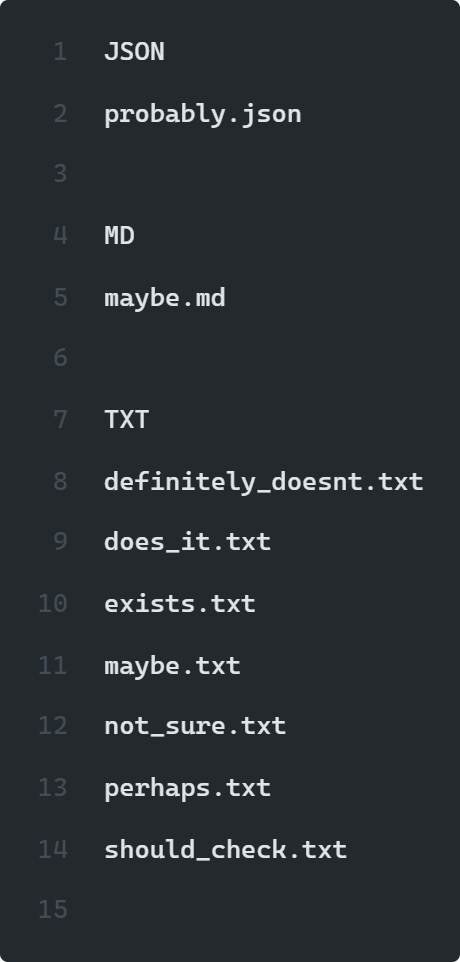


Рисунок 1.3 – Вихідний файл з результатами роботи програми

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)

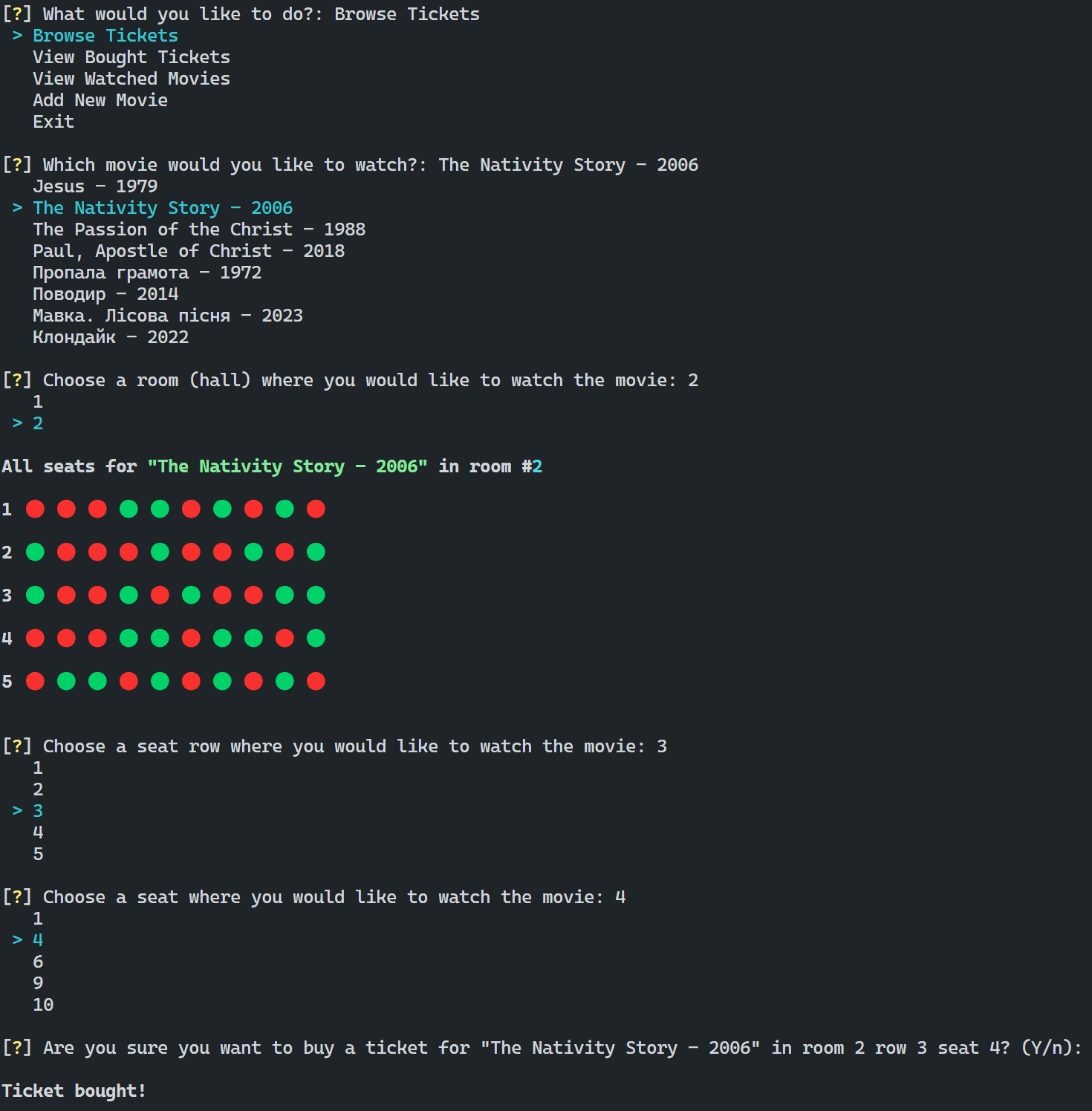
        cinema.movies.append(movie)

    else:

        break

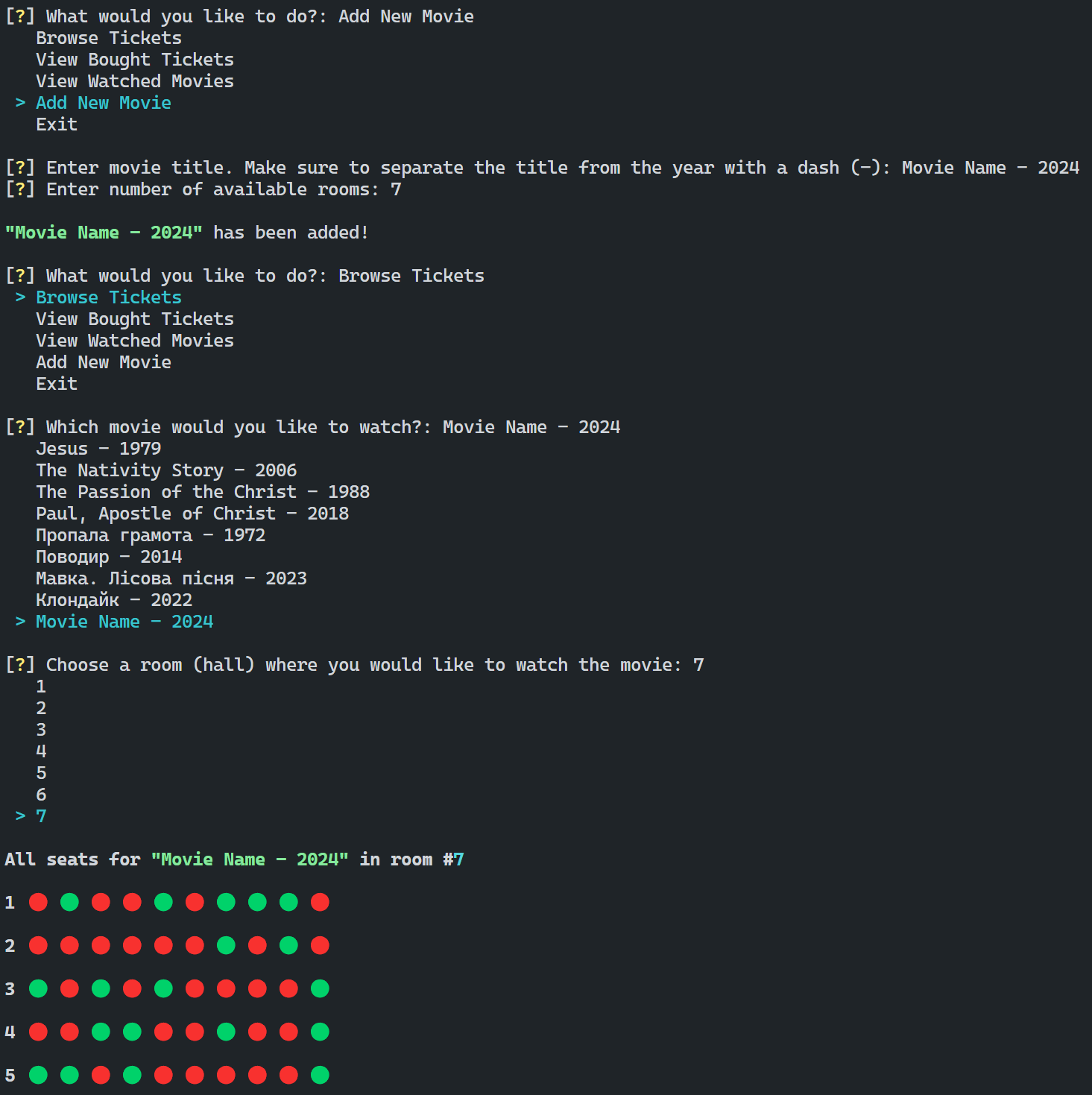
Результати виконання

Результати виконання програми наведені нижче у вигляді знімків з екрану:









Як вхідні дані використовувався файл JSON наступного змісту:

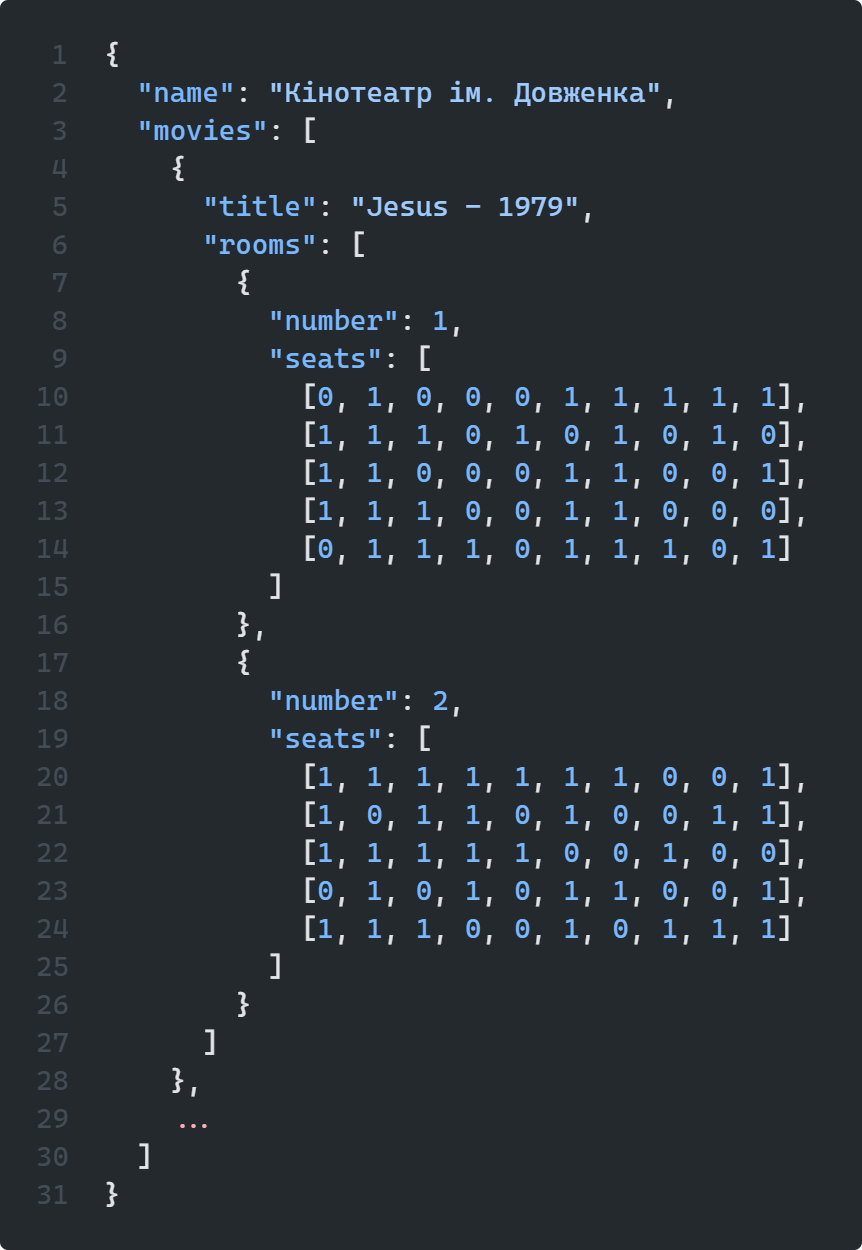


Рисунок 2.2 – Файл вхідних даних

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

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()