

Department of Computer Engineering and Informatics

Faculty of Science and Technology, Middlesex University Dubai



**Middlesex
University
Dubai**

CST1510

**Programming For Data Communications and
Networks**

Portfolio-1

Student Name:Aahana Kembhavi

MISIS:M01085905

Module Lecturer:Miss Sumitra Kotipalli

Introduction

The purpose of this portfolio is to showcase my learning journey and the skills I developed throughout this module. It reflects my growth in understanding core programming concepts and applying them to practical tasks.

I built a strong foundation in Python programming, learning core concepts, advanced data handling, and practical coding skills. I also explored Pandas for data analysis, Git for version control, and applied everything in a hands-on project, strengthening my problem-solving and technical abilities.

In my learning progression, I explored different Python libraries, understood how to use in-built keywords, and learned to manage data efficiently. I also developed the ability to identify and fix errors, improving both my coding accuracy and analytical thinking. I developed skills in breaking complex problems into logical steps, thinking critically and creatively while interpreting data, and writing efficient, well-structured code.

My journey reflects strong growth in both understanding and applying Python concepts. I progressed from grasping basic syntax and logic to confidently using advanced features, libraries, and data structures.

Along the way, I learned to write cleaner code, manage data effectively, debug errors, and apply programming skills to real-world projects, showing clear improvement in both technical depth and problem-solving ability.

1. Introduction

2. Week 1- Getting Started with Python

2.1. QuestionExplanation

2.2. Code

2.3. Explanation of Code

2.4. Reflection

3. Week 2- Control structure and lists

2.1. QuestionExplanation

2.2. Code

2.3. Explanation of Code

2.4. Reflection

4. Week 3- Functions, Lambda Expression and Multidimensional Lists

2.1. QuestionExplanation

2.2. Code

2.3. Explanation of Code

2.4. Reflection

5. Week 4- String, Special Methods and Regular Expressions

2.1. QuestionExplanation

2.2. Code

2.3. Explanation of Code

2.4. Reflection

6. Week 5- File Handling, Exceptions and Data structures

2.1. QuestionExplanation

2.2. Code

2.3. Explanation of Code

2.4. Reflection

7. Conclusion:

2. Week 1-Getting Started With Python

2.1. Question Explanation

Excellent 3. IT: Path Decomposition Report

Ask for a file or directory path and print its absolute path, parent, name, stem, and suffix: This question asks us to enter a file that then converts it into a pure path and we need to display certain properties.

2.2. Code

```
# importing pathlib library to ask for directory path and print it's absolute path, parent, name, stem, and suffix.
from pathlib import Path

#Asking the user to enter a path
inp = input("Enter a path: ")

#expanduser converts string into path and resolve converts it into a pure path.
p = Path(inp).expanduser().resolve()

#these are properties from the imported library i.e, pathlib
print("Absolute path:", p)
print("Parent:", p.parent)
print("Name:", p.name)
print("Stem:", p.stem)
print("Suffix:", p.suffix)
```

Code

```
Enter a path: /home/user/project/service.log
Absolute path: C:\home\user\project\service.log
Parent: C:\home\user\project
Name: service.log
Stem: service
Suffix: .log
```

Output

2.3 Explanation of the Code

So First we start with importing the **library 'pathlib'** in order to print the path's absolute path

inp = input("Enter a path: "), So this is a prompt for the user to enter a path

p = Path(inp).expanduser().resolve(): Here We are adding a variable p that has expanduser which converts the string entered by the user into 'path' and then resolve converts the path into 'absolute path'.

from print("Absolute path:", p) to print("Suffix:", p.suffix): These are statements to print the properties of the path and display them.

2.4 Reflection

As a beginner with limited coding experience, Week 1 helped me build confidence by strengthening my understanding of the basics and guiding me to write cleaner, more efficient code. I also had difficulty because using this Library was unfamiliar but after learning it i could apply it in my code

3. Week 2- Control structure and lists

2.1. Question Explanation

The program creates a random two-digit lottery number, The user enters a two-digit number, There are three prize rules:

- 1) Exact match → £10,000
- 2) Both digits match but order different → £3,000
- 3) One digit matches → £1,000,

2.2. Code

```
: import random

# Generating a random two-digit lottery number
lottery = random.randint(10, 99)
lottery_str = str(lottery)

while True:

    user_input = input("Enter a two-digit number ('q' to quit): ")

    if user_input.lower() == 'q': # This is the Quit option that makes you forfeit trying again.
        print("Thanks for playing!")
        break

    print(f"Lottery number: {lottery}")

    if user_input == lottery_str:
        print("Exact match! You win £10,000")
        break
    elif sorted(user_input) == sorted(lottery_str):
        print("Both digits match! You win £3,000")
        break
    elif any(digit in lottery_str for digit in user_input):
        print("One digit matches! You win £1,000")
        break
    else:
        print("No match. Try again!\n")
```

Code



```
Enter a two-digit number ('q' to quit): 47
Lottery number: 54
One digit matches! You win £1,000
```

Output



2.3 Code Explanation:

while True: → Creates an infinite loop so the user can keep guessing, until they decide to forfeit.

user_input = input("Enter a two-digit number (or 'q' to quit): ") → Takes user input each time the loop/code runs.

if user_input.lower() == 'q': → Allows the user to quit the game by typing the alphabet 'q'.

print(f"Lottery number: {lottery}") → Shows the generated lottery number.

if user_input == lottery_str: → Checks for exact match (£10,000).

elif sorted(user_input) == sorted(lottery_str): → Checks if both digits match in any order (£3,000).

elif any(digit in lottery_str for digit in user_input): → Checks if at least one digit matches (£1,000).

else: → If none match, tells the user to try again.

break → Stops the loop once the user wins or quit

2.4 Reflection

Learning conditional statements taught me how to make decisions with if, elif, and else. Loops like for and while help repeat tasks efficiently.

Using both together makes programs dynamic and responsive. I also learned to plan logic carefully to avoid errors.

These skills improved my problem-solving and coding confidence.

4. Week 3- Functions, Lambda Expression and Multidimensional Lists

3.1. Question Explanation:

The program converts temperatures between Celsius and Fahrenheit.

The user chooses which conversion to perform and enters a numeric temperature.

3.2. Code

10. Rewrite the quiz or the scientific converter including mainly lambda functions.

Code

```
]: # scientific converter using Lambda functions, (converting Celsius to Fahrenheit and vice versa)

c_to_f = lambda c: (c * 9/5) + 32
f_to_c = lambda f: (f - 32) * 5/9

choice = input("Convert to (C)elsius or (F)ahrenheit? ").upper()
value = float(input("Enter the temperature: "))

result = c_to_f(value) if choice == 'F' else f_to_c(value)
print("Converted temperature:", result)
```

Output

```
Convert to (C)elsius or (F)ahrenheit? celsius
Enter the temperature: 34
Converted temperature: 1.1111111111111112
```

3.3. Explanation of Code

This Python program converts temperatures between Celsius and Fahrenheit using lambda functions. It has **c_to_f = lambda c: (c * 9/5) + 32** to change Celsius to Fahrenheit and **f_to_c = lambda f: (f - 32) * 5/9** to change Fahrenheit to Celsius.

The user chooses the type of

conversion with choice = input("Convert to (C)elsius or (F)ahrenheit? ").upper() and enters a number with **value = float(input("Enter the temperature: "))**.

Then the program decides which lambda to use with **result = c_to_f(value)** if **choice == 'F'** else **f_to_c(value)** and prints it with **print("Converted temperature:", result)**. (Lambdas are just a short way to make small functions without writing def)

3.4. Reflection

One thing I learned is how to use lambda functions to make small, quick functions without writing def.

One difficulty I faced was understanding how to use them with user input and one-line if-else statements, but I also realized that lambdas can make code cleaner and easier to read once you get the hang of them.

5. Week 4- Strings, Special Methods, Modules, Try-Except statement and (optional) Regular Expressions

4.1. Question Explanation

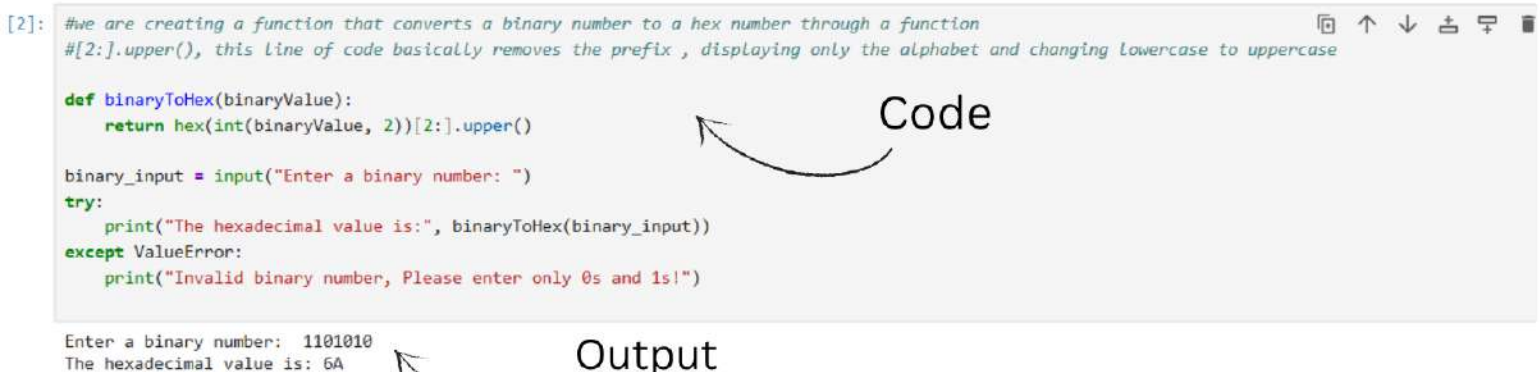
This Question is about writing a function that converts a binary number entered by the user into its hexadecimal equivalent.

The program should prompt the user for a binary number, convert it to hex, and display the result. It should also handle invalid input gracefully.

Using `int(binaryValue, 2)` converts the binary string to an integer, and `hex()` converts it to hexadecimal. The `[2:].upper()` removes the 0x prefix from the hex string and converts any letters to uppercase.

4.2. Code

16. (Binary to hex) Write a function that parses a binary number and converts it into a hex number. The function header is: `def binaryToHex(binaryValue):` Write a test program that prompts the user to enter a binary number and displays the corresponding hexadecimal value.



```
[2]: #we are creating a function that converts a binary number to a hex number through a function  
#[2:].upper(), this line of code basically removes the prefix , displaying only the alphabet and changing lowercase to uppercase  
  
def binaryToHex(binaryValue):  
    return hex(int(binaryValue, 2))[2:].upper()  
  
binary_input = input("Enter a binary number: ")  
try:  
    print("The hexadecimal value is:", binaryToHex(binary_input))  
except ValueError:  
    print("Invalid binary number, Please enter only 0s and 1s!")  
  
Enter a binary number: 1101010  
The hexadecimal value is: 6A
```

4.3. Explanation of Code

`def binaryToHex(binaryValue): return hex(int(binaryValue, 2))[2:].upper()` converts a binary string to hexadecimal

`binary_input = input("Enter a binary number: ")` gets input from the user.

`try-except ValueError:` ensures the program handles invalid binary input.

`print("The hexadecimal value is:", binaryToHex(binary_input))` prints the converted value

4.4. Reflection

I learned how Python makes number base conversions simple using `int()` and `hex()`, and I realized it can be tricky at first to handle invalid input, but using `try-except` makes the program more reliable and user-friendly.

6. Week 5- Files and Exceptions

5.1. Question Explanation

This Question is about creating a system to manage tickets for events. Each event has a limited number of tickets, and we need to track both the availability of tickets and the users who purchased them.

Users should be able to buy tickets if they are still available, check how many tickets are left, and see who has already purchased tickets.

To handle this efficiently, dictionaries are used: one (available_tickets) to store the number of tickets for each event, and another (purchasers) to store lists of users who bought tickets for each event.

3. Implement functions to check availability and view who has purchased tickets.

Code

```
available_tickets = {"Concert": 100, "Football Game": 50, "Theater Show": 75}
purchasers = {"Concert": ["John", "Emily"], "Football Game": ["Alex"], "Theater Show": ["Sam", "John"]}

def buy_ticket(event, user):
    if available_tickets.get(event, 0) > 0:
        if event not in purchasers:
            purchasers[event] = []
        purchasers[event].append(user)
        available_tickets[event] -= 1
        return True
    return False

def check_availability(event):
    return available_tickets.get(event, 0)

def view_purchasers(event):
    return purchasers.get(event, [])

event_name = input("Enter the event name: ")
user_name = input("Enter your name: ")

if buy_ticket(event_name, user_name):
    print("Ticket purchased successfully!")
    print("Remaining tickets:", check_availability(event_name))
    print("Purchasers:", view_purchasers(event_name))
else:
    print("Sorry, tickets are sold out or event not found.")
```

Output

```
Enter the event name: Concert
Enter your name: Aahana
Ticket purchased successfully!
Remaining tickets: 99
Purchasers: ['John', 'Emily', 'Aahana']
```

5.3. Explanation of Code

available_tickets keeps the count of remaining tickets for each event.

purchasers keeps track of who bought tickets.

buy_ticket(event, user) checks if tickets are available, adds the user to the purchaser list, and decreases the ticket count.

check_availability(event) returns the number of remaining tickets.

view_purchasers(event) returns the list of purchasers for the event.

5.4. Reflection

I learned how dictionaries can help organize related data like ticket counts and purchaser lists, and I realized it can be tricky at first to update both counts and lists correctly, but practicing this helped me understand simple data management for events.

7. Conclusion

Over the past five weeks, I explored the fundamentals and intermediate concepts of Python programming. I started with understanding basic syntax and operations, progressed through control structures, lists, and functions, and finally delved into advanced topics like string manipulation, regular expressions, file handling, exceptions, and data structures. Each week built upon the previous, giving me a comprehensive understanding of Python programming.

I developed technical and analytical skills, including using loops, conditional statements, writing functions, employing lambda expressions, handling multidimensional lists, working with strings and special methods, applying regular expressions, managing files, handling exceptions, and organizing data using various data structures. I also enhanced my problem-solving and logical reasoning abilities. I also learnt how different libraries have different purposes, functionalities, and implementations and how I can link them to our other module

Data science and Ai

The biggest challenge was understanding and applying regular expressions and handling exceptions in complex scenarios. Overcoming this challenge improved my confidence in debugging and working with real-world data efficiently.

I look forward to learning more advanced Python topics in the upcoming weeks, including object-oriented programming, libraries like NumPy and Pandas, and applying my skills to projects involving data analysis and automation. I also plan to use these foundational skills in my further studies in Data Science and AI.

Overall, this journey has tranquilised my Python programming skills and problem-solving capabilities solving the typical and excellent questions. I am excited to continue learning, applying these skills, and tackling more complex programming challenges in the future.