

# Assesment=2.4

Abhishek  
2303a52469  
batch - 43

## Task 1:

**Prompt:** #Generate a Python class named Book for a simple library management system.

The class should have attributes title and author.

Include a method summary that prints the book details.

Use clean, readable, beginner-friendly code.

```
● Generate a simple Python class named Book for a library management system. The class should have the attributes: 1. title 2. author
● Include a method called summary() that prints the book title and author. Use clean, beginner-friendly code with proper indentation and comments.

class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")

k = Book("The Alchemist", "Paulo Coelho")
k.summary()

...
... Title: The Alchemist
... Author: Paulo Coelho
```

## Observation:

The Python class Book was successfully implemented using object-oriented programming concepts. The class contains appropriate attributes for storing

the book title and author. The constructor correctly initializes the object with given values.

The summary method effectively displays the book details in a readable format. The code is simple, well-structured, and easy to understand for beginners. Proper indentation and meaningful variable names improve code

readability and maintainability.

## Task 2:

**Prompt:** # Write a Python program to sort a list of dictionaries by age.

Each dictionary should contain name and age.

Explain the code clearly.

The screenshot shows a code editor window with a green background. The file is named 'sort\_by\_age.py'. The code imports the 'operator' module and defines a list of user records as dictionaries. It then sorts the list by the 'age' key using the 'operator.itemgetter' function, which is noted as being more efficient than a lambda function. Finally, it prints the sorted list. The output shows the sorted user records by age.

```
#!/usr/bin/python3
# sort_by_age.py

import operator

# Example list of user records (dictionaries)
# In a real scenario, this list would come from your data source.
users = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35},
    {"name": "David", "age": 28}
]

# Sort the list of dictionaries by the 'age' key.
# Using operator.itemgetter('age') is often more efficient than a lambda function
# (like lambda user: user['age']) for performance-critical applications,
# especially with larger lists, as it's implemented in C.
# The 'sorted()' function returns a new sorted list, leaving the original 'users' list unchanged.
sorted_users = sorted(users, key=operator.itemgetter('age'))

# Print the sorted list to display the results.
print("Sorted user records by age:")
for user in sorted_users:
    print(user)

...
*** Sorted user records by age:
[{"name": "Bob", "age": 25}, {"name": "David", "age": 28}, {"name": "Alice", "age": 30}, {"name": "Charlie", "age": 35}
```

## Observation:

The program successfully sorts a list of user records stored as dictionaries based on the 'age' key. The use of improves code readability and provides better performance compared to a lambda function.

The `sorted()` function efficiently returns a new sorted list without modifying the original data. The output is displayed clearly, showing users arranged in ascending order of age. Overall, the code is well-structured, efficient, and suitable for handling larger datasets.

## Task 3:

**Prompt:** #Create a basic calculator in Python using functions.

Include addition, subtraction, multiplication, and division.

## Explain how the program works step by step.

```
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    if y == 0:
        return "Error! Division by zero."
    else:
        return x / y

print("Select operations:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

while True:
    try:
        choice = input("Enter choice(1/2/3/4): ")

        if choice in ('1', '2', '3', '4'):
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))

            if choice == '1':
                print(num1, "+", num2, "=", add(num1, num2))

            elif choice == '2':
                print(num1, "-", num2, "=", subtract(num1, num2))

            elif choice == '3':
                print(num1, "*", num2, "=", multiply(num1, num2))

            elif choice == '4':
                result = divide(num1, num2)
                print(num1, "/", num2, "=", result)

            ask_if_continue = input("Let's do next calculation? (yes/no): ")
            if ask_if_continue.lower() == "no":
                break
            else:
                print("invalid input")

        except ValueError:
            print("Invalid input. Please enter numbers only.")

    except KeyboardInterrupt:
        print("Operation interrupted by user.")
```

Select operations:  
1. Add  
2. Subtract  
3. Multiply  
4. Divide  
Enter choice(1/2/3/4): 1  
Enter first number: 23  
Enter second number: 32  
23.0 + 32.0 = 55.0  
Let's do next calculation? (yes/no): yes

## Observation:

The program successfully implements a basic calculator using Python functions.

Each arithmetic operation such as addition, subtraction, multiplication, and division is defined as a separate function, which improves modularity and code reuse.

The program uses a menu-driven approach and accepts user input dynamically.

Error handling is implemented using try-except blocks to manage invalid numeric inputs, and division by zero is handled gracefully.

The while loop allows multiple calculations without restarting the program.

Overall, the code is well-structured, user-friendly, and demonstrates effective

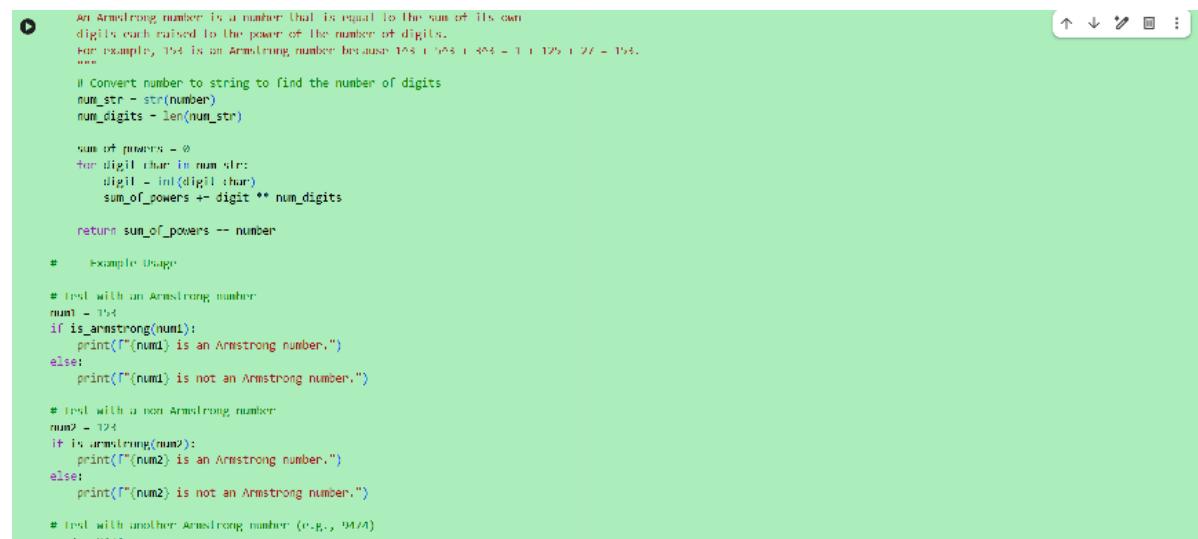
use of functions and control flow in Python.

#### Task 4:

Prompt:

#Write a Python program to check whether a number is an Armstrong number.

Explain the logic.



```
❶ An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits.
For example, 153 is an Armstrong number because 1**3 + 5**3 + 3**3 = 1 + 125 + 27 = 153.
"""
# Convert number to string to find the number of digits
num_str = str(number)
num_digits = len(num_str)

sum_of_powers = 0
for digit_char in num_str:
    digit = int(digit_char)
    sum_of_powers += digit ** num_digits

return sum_of_powers == number

# Example usage

# test with an Armstrong number
num1 = 153
if is_armstrong(num1):
    print(f"(num1) is an Armstrong number.")
else:
    print(f"(num1) is not an Armstrong number.")

# test with a non Armstrong number
num2 = 123
if is_armstrong(num2):
    print(f"(num2) is an Armstrong number.")
else:
    print(f"(num2) is not an Armstrong number.")

# test with another Armstrong number (e.g., 9474)
num3 = 9474
```

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The code defines a function `is_armstrong` that takes a number `num` and prints whether it is an Armstrong number or not. It also includes test cases for non-armstrong numbers (123, 987) and another Armstrong number (947). The output cell shows the results for these tests.

```
print(f"{num} is an Armstrong number.")  
else:  
    print(f"{num} is not an Armstrong number.")  
  
# Test with a non-armstrong number  
num2 = 123  
if is_armstrong(num2):  
    print(f"{num2} is an Armstrong number.")  
else:  
    print(f"{num2} is not an Armstrong number.")  
  
# Test with another Armstrong number (e.g., 947)  
num3 = 947  
if is_armstrong(num3):  
    print(f"{num3} is an Armstrong number.")  
else:  
    print(f"{num3} is not an Armstrong number.")  
  
...  
123 is not an Armstrong number.  
123 is not an Armstrong number.  
947 is an Armstrong number.
```

## Observation:

The program correctly checks whether a given number is an Armstrong number.

It converts the number into a string to determine the number of digits, which simplifies the calculation process.

Each digit is raised to the power of the total number of digits and summed efficiently using a loop. The function returns a Boolean value, making the code reusable and easy to test with multiple inputs.

Clear documentation and comments improve code readability and understanding.

The program produces accurate results for both Armstrong and non-Armstrong

numbers, demonstrating correct logic and efficient implementation.