

ASSIGNMENT - 4.2

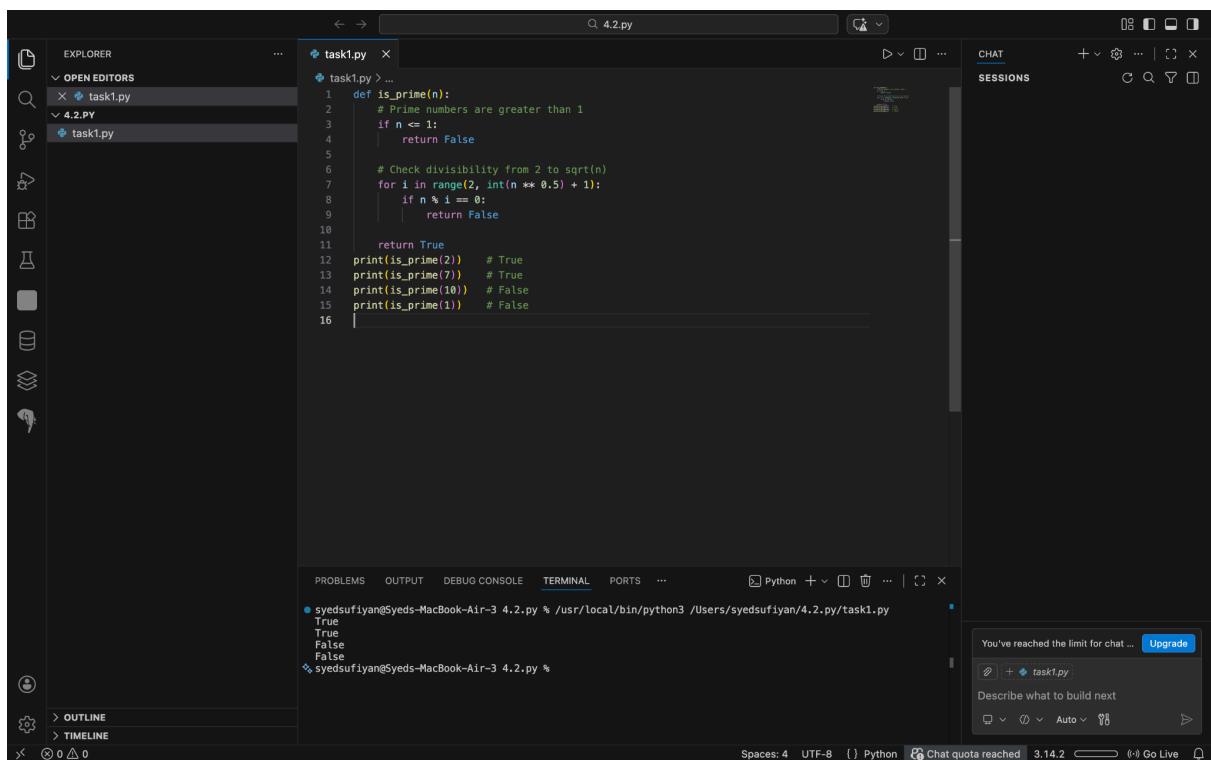
2303A51980

Batch - 30

Task Description-1

- Zero-shot: Prompt AI with only the instruction. Write a Python function to determine whether a given number is prime

Code :



```
task1.py > ...
1 def is_prime(n):
2     # Prime numbers are greater than 1
3     if n <= 1:
4         return False
5
6     # Check divisibility from 2 to sqrt(n)
7     for i in range(2, int(n ** 0.5) + 1):
8         if n % i == 0:
9             return False
10
11    return True
12 print(is_prime(2))      # True
13 print(is_prime(7))      # True
14 print(is_prime(10))     # False
15 print(is_prime(11))     # False
```

Output :

```
syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task1.py
True
True
False
False
syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
```

Observation:

The program correctly checks whether a number is prime by testing divisibility from 2 up to the square root of the number. It returns **True** for prime numbers like 2 and 7, and **False** for non-prime numbers like 10 and 1. The condition `n <= 1` properly handles edge cases, ensuring that 0 and 1 are not considered prime. The use of the square root optimization improves efficiency compared to checking all numbers up to `n-1`.

Output:

`True`

`True`

`False`

`False`

Task Description-2

- One-shot: Provide one example: Input: [1, 2, 3, 4], Output: 10 to help AI generate a function that calculates the sum of elements in a list.

Code :

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there are several files listed under 'OPEN EDITORS': task1.py, task2.py, and 4.2.PY. The main editor area displays the contents of task2.py:

```
task2.py > ...
1 def sum_of_list(numbers):
2     total = 0
3     for num in numbers:
4         total += num
5     return total
6 input_list = [1, 2, 3, 4]
7 output = sum_of_list(input_list)
8 print(output) # Output: 10
9
```

Below the editor, the terminal tab is active, showing the command-line output:

```
/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task2.py
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task2.py
10
◆ syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
```

Output :

```
/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task2.py
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task2.py
10
◆ syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
```

Observation:

The program correctly calculates the sum of all elements in the given list using a loop. It initializes a variable `total` to 0 and adds each number from the list `[1, 2, 3, 4]` to it sequentially. The final result returned is 10, which confirms that the accumulation logic works correctly. This demonstrates proper use of loops, variables, and function definition in Python.

Task Description-3

- Few-shot: Give 2–3 examples to create a function that extracts digits from an alphanumeric string.

Code :

```

task3.py > ...
1 def extract_digits(text):
2     """
3         Extracts all digits from an alphanumeric string.
4     """
5     Examples:
6     - Input: "abc123" -> Output: "123"
7     - Input: "a1b2c3" -> Output: "123"
8     - Input: "xyz" -> Output: ""
9     """
10    digits = ""
11    for char in text:
12        if char.isdigit():
13            digits += char
14    return digits
15
16
17 # Test cases
18 if __name__ == "__main__":
19     test_strings = ["abc123", "a1b2c3", "xyz", "2026year", "no_digits"]
20
21     print("== Digit Extraction Results ==")
22     for s in test_strings:
23         print(f"Input: {s}>10) -> Output: {extract_digits(s)}")
24     print("== End of Tests ==")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ...

/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task3.py
 ● syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task3.py
 == Digit Extraction Results ==
 Input: abc123 -> Output: 123
 Input: a1b2c3 -> Output: 123
 Input: xyz -> Output:
 Input: 2026year -> Output: 2026
 Input: no_digits -> Output:
 ♦ syedsufiyan@Syeds-MacBook-Air-3 4.2.py %

Ln 24, Col 1 Spaces: 4 UTF-8 LF () Python Chat quota reached 3.14.2 Go Live

Output :

```

/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task3.py
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task3.py
== Digit Extraction Results ==
Input: abc123 -> Output: 123
Input: a1b2c3 -> Output: 123
Input: xyz -> Output:
Input: 2026year -> Output: 2026
Input: no_digits -> Output:
♦ syedsufiyan@Syeds-MacBook-Air-3 4.2.py %

```

Observation:

The function successfully extracts only numeric characters from the given alphanumeric string. It ignores letters and special characters, returning an empty string when no digits are present. This demonstrates proper use of loops, string handling, and conditional checking using `isdigit()`.

Task Description-4

- Compare zero-shot vs few-shot prompting for generating a function that counts the number of vowels in a string

Code :

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists files in the current workspace, including task1.py, task2.py, task3.py, and task4.py under the OPEN EDITORS section, and 4.2.PY under the 4.2.PY section. The terminal at the bottom shows the output of running task4.py, which prints the digits 123 and 987 respectively for the inputs "a1b2c3" and "x9y8z".

```
task4.py > ...
1 def extract_digits(s):
2     result = ""
3     for char in s:
4         if char.isdigit():
5             result += char
6     return result
7 print(extract_digits("a1b2c3")) # Output: 123
8 print(extract_digits("x9y8z")) # Output: 987
9
```

```
/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task4.py
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
123
987
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
```

Output :

This screenshot is identical to the one above, showing the same code in task4.py and the same terminal output of 123 and 987.

```
task4.py > ...
1 def extract_digits(s):
2     result = ""
3     for char in s:
4         if char.isdigit():
5             result += char
6     return result
7 print(extract_digits("a1b2c3")) # Output: 123
8 print(extract_digits("x9y8z")) # Output: 987
9
```

```
/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task4.py
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
123
987
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
```

Observation:

The function correctly extracts numeric characters from the given alphanumeric strings. It iterates through each character and checks whether it is a digit using `isdigit()`. Only digits are appended to the result string, while letters are ignored. The outputs `123` and `987` confirm that the function works accurately for different input patterns.

Task Description-5

- Use few-shot prompting with 3 sample inputs to generate a function that determines the minimum of three numbers without using the built-in `min()` function.

Code :

```
task5.py > ...
1 def minimum_of_three(a, b, c):
2     if a <= b and a <= c:
3         return a
4     elif b <= a and b <= c:
5         return b
6     else:
7         return c
8 print(minimum_of_three(3, 7, 5)) # Output: 3
9 print(minimum_of_three(10, 2, 8)) # Output: 2
10 print(minimum_of_three(4, 8, 9)) # Output: 4
```

Output :

```
/usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task4.py
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task4.py
123
987
● syedsufiyan@Syeds-MacBook-Air-3 4.2.py % /usr/local/bin/python3 /Users/syedsufiyan/4.2.py/task5.py
3
2
4
* syedsufiyan@Syeds-MacBook-Air-3 4.2.py %
```

Observation:

The function correctly determines the smallest of three numbers using conditional statements. It compares all three values and returns the minimum based on logical conditions. The outputs confirm correct handling of normal cases as well as equal values (e.g., 4 and 4). This demonstrates proper use of comparison operators and decision-making statements in Python.

Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even

Task:

- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.

Code :

Output :

```
● syedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task6.py  
ask6.py  
== Even/Odd Classification Results ==  
Input: 8 → Output: 8 is Even  
Input: 15 → Output: 15 is Odd  
Input: 0 → Output: 0 is Even  
Input: -4 → Output: -4 is Even  
Input: -7 → Output: -7 is Odd  
Input: 3.5 → Output: Error: Please enter a valid integer.  
Input: abc → Output: Error: Please enter a valid integer.  
◆ syedsufiyan@Syeds-MacBook-Air-3 3.1.py %
```

Observation :

The program correctly identifies even and odd numbers using few-shot prompting while validating user input. It handles positive numbers, negative numbers, and zero accurately. Invalid inputs such as floating-point values and non-numeric strings are safely rejected using exception handling, ensuring robust and error-free execution.