

Assignment 2.4 - Exploring Additional AI Coding Tools (Gemini & Cursor AI)

Name: D. Prasanna Latha

Roll Number: 2503A51L37

Course Code: 24CS002PC215

Course Title: AI Assisted Coding

Assignment Number: 2.4

Academic Year: 2025-2026

Observation: The bubble sort implementation worked correctly but was slower compared to the built-in sort() function. This shows how AI tools can generate both traditional algorithms and efficient built-in alternatives, helping us compare performance and readability.

Task 1: Sorting with Bubble Sort vs Built-in sort()

Prompt/Comment given to AI Tool: Generate Python code for sorting a list using bubble sort and also using Python's built-in sort()

Generated Code:

```
# Bubble Sort Implementation
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
# Built-in sort
arr = [64, 25, 12, 22, 11]
print("Bubble Sort:", bubble_sort(arr.copy()))
print("Built-in sort:", sorted(arr))
```

Explanation:

The bubble sort function repeatedly swaps adjacent elements if they are in the wrong order. The built-in sort() uses Timsort which is more efficient.

```

1 # Bubble Sort implementation
2 def bubble_sort(arr):
3     n = len(arr)
4     for i in range(n):
5         # Last i elements are already in place
6         for j in range(0, n - i - 1):
7             if arr[j] > arr[j + 1]:
8                 # Swap if the element found is greater
9                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
10
11 # Example list
12 my_list = [64, 34, 25, 12, 22, 11, 90]
13
14 # Using bubble sort
15 bubble_sorted_list = my_list.copy()
16 bubble_sort(bubble_sorted_list)
17 print("Sorted with bubble sort:", bubble_sorted_list)
18
19 # Using built-in sort()
20 builtin_sorted_list = my_list.copy()
21 builtin_sorted_list.sort()
22 print("Sorted with built-in sort():", builtin_sorted_list)

```

PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe "c:/Users/nishc/OneDrive/Documents/Desktop/ASSIGNMENT2.4 AI/TASK1"
Sorted with bubble sort: [11, 12, 22, 25, 34, 64, 90]
Sorted with built-in sort(): [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI>

****Output:****

Input: [64, 25, 12, 22, 11] Output: Bubble Sort: [11, 12, 22, 25, 64] Built-in sort: [11, 12, 22, 25, 64]

Observation: The code successfully counted vowels, consonants, and digits. The AI-generated solution was simple, accurate, and reusable for different input strings. It highlighted how AI can assist in writing clean and logical character analysis functions.

Task 2: Count Vowels, Consonants, and Digits

****Prompt/Comment given to AI Tool:**** Write a function that counts vowels, consonants, and digits in a string

****Generated Code:****

```

def analyze_string(s: str):    vowels = "aeiouAEIOU"    v = c = d = 0    for ch in s:    if ch.isdigit():        d += 1
elif ch.isalpha():        if ch in vowels:            v += 1        else:            c += 1    return v, c
dprint(analyze_string("Hello123"))

```

****Explanation:****

The function checks each character. If it is a digit, it increments digit count. If alphabet, checks vowels vs consonants. Finally returns counts.

The screenshot shows the Microsoft Visual Studio Code interface. In the Explorer sidebar, there is a folder named 'ASSIGNMENT2.4 AI' containing 'TASK1' and 'TASK2'. The 'TASK2' tab is active, displaying a Python script. The code defines a function 'count_vowels_consonants_digits(s)' that iterates through each character in the input string 's'. It uses conditional statements to check if a character is a vowel, consonant, or digit, and increments counters accordingly. An example usage is shown at the bottom of the script.

```

1 def count_vowels_consonants_digits(s):
2     vowels = 'aeiouAEIOU'
3     num_vowels = 0
4     num_consonants = 0
5     num_digits = 0
6     for char in s:
7         if char.isdigit():
8             num_digits += 1
9         elif char.isalpha():
10            if char in vowels:
11                num_vowels += 1
12            else:
13                num_consonants += 1
14    return num_vowels, num_consonants, num_digits
15
16 # Example usage
17 input_str = "Hello World! 123"
18 vowels, consonants, digits = count_vowels_consonants_digits(input_str)
19 print(f"Vowels: {vowels}, Consonants: {consonants}, Digits: {digits}")

```

In the Chat panel, a message from the AI tool says: 'Write a function that counts vowels, consonants, and digits in a string'. Below it, a note says: 'A function to count vowels, consonants, and digits in a string has been added to TASK2. It also includes an example usage and prints the results. Let me know if you want any modifications or further explanation!' There is also a link to 'TASK2'.

The Terminal tab shows the command 'python312\python.exe "c:/Users/nishc/OneDrive/Documents/Desktop/ASSIGNMENT2.4 AI/TASK2"' being run, followed by the output: 'Vowels: 3, Consonants: 7, Digits: 3'.

Output: Input: 'Hello123' → Output: Vowels=2, Consonants=3, Digits=3

Observation: The file was created, written to, and read successfully. This demonstrates how AI-generated code can automate basic file handling tasks without syntax errors. It also shows the practical importance of context management ('with` statement) in Python.

Task 3: File Handling (Create, Write, Read)

Prompt/Comment given to AI Tool: Generate code to create a text file, write sample text, and read it back

Generated Code:

```
# File handling with Cursor AI
filename = "sample.txt" # Write to file with open(filename, "w") as f:
f.write("This is a sample file.\nWelcome to AI coding tools.") # Read file content with open(filename, "r") as f:
content = f.read() print(content)
```

Explanation:

This program first creates a file named sample.txt, writes two lines into it, and then opens it again in read mode to display the content.

```

1 # Create a text file, write sample text, and read it back
2
3 def write_and_read_file(filename, text):
4     # Write sample text to the file
5     with open(filename, 'w') as f:
6         f.write(text)
7     # Read the text back from the file
8     with open(filename, 'r') as f:
9         content = f.read()
10    return content
11
12 # Example usage
13 file_name = 'sample.txt'
14 sample_text = 'This is a sample text written to the file.'
15 read_content = write_and_read_file(file_name, sample_text)
16 print('Content read from file:', read_content)
17

```

Observation: The calculator worked correctly for all four operations. The modular design of functions made the code easy to understand and extend. This observation highlights how AI helps in generating structured and reusable function-based programs.

Output: This is a sample file. Welcome to AI coding tools.

Task 4: Simple Calculator using Functions

****Prompt/Comment given to AI Tool:**** Generate a calculator with add, subtract, multiply, divide functions and explain the code

****Generated Code:****

```

def add(a,b): return a+b
def subtract(a,b): return a-b
def multiply(a,b): return a*b
def divide(a,b): return a/b if b!=0 else "Error"
choice = '+'
print("5 + 3 =", add(5,3))

```

****Explanation:****

Each function performs one arithmetic operation. The user selects an operator, and based on that the corresponding function is called. Gemini also provides a line-by-line explanation of this code.

The screenshot shows a Microsoft Visual Studio Code interface. On the left is the Explorer sidebar with a tree view of files under 'ASSIGNMENT2.4 AI'. The main area is a code editor with Python code for a simple calculator. The terminal at the bottom shows the output of running the script. A Chat panel on the right shows a conversation with an AI tool about generating a calculator with specific functions.

```

1 # Simple Calculator with add, subtract, multiply, and divide functions
2
3 def add(a, b):
4     """Return the sum of a and b."""
5     return a + b
6
7 def subtract(a, b):
8     """Return the difference of a and b."""
9     return a - b
10
11 def multiply(a, b):
12     """Return the product of a and b."""
13     return a * b
14
15 def divide(a, b):
16     """Return the quotient of a and b. Raises an error if b is zero."""
17     if b == 0:
18         raise ValueError("Cannot divide by zero.")
19     return a / b
20
21 # Example usage
22 x = 10
23 y = 5
24 print(f"(x) + (y) = {add(x, y)}")

```

TERMINAL:

```

PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe "c:/Users/nishc/OneDrive/Documents/Desktop/ASSIGNMENT2.4 AI/TASK4"
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI>

```

CHAT:

Generate a calculator with add, subtract, multiply, divide functions and explain the code

Created: TASK4

A calculator with add, subtract, multiply, and divide functions has been created in TASK4. The code includes example usage and an explanation as comments at the end. Let me know if you need further details or modifications!

****Output:****

Input: 10+5 → Output:15.

Input:10-5->output:5

Input:10*5->output:50

Input:10/5->output:2.0

Observation: Both versions (if-else and lambda one-liner) worked correctly. Version 1 was clearer for learning purposes, while Version 2 was more concise but less readable. This shows how AI tools can generate multiple valid solutions depending on user preference for clarity or brevity.

Task 5: Leap Year Check with Different Prompts

****Prompt/Comment given to AI Tool:**** Prompt 1: Generate a function to check leap year
Prompt 2: Write a one-liner leap year check

****Generated Code:****

```
# Version 1 - Using if-else
def is_leap_year(year):
    if (year % 400 == 0) or ((year % 4 == 0) and (year % 100 != 0)):
        return True
    return False
print(is_leap_year(2024))

# Version 2 - One-liner
is_leap = lambda y: (y%400==0) or (y%4==0 and y%100!=0)
print(is_leap(2023))
```

****Explanation:****

Version 1 uses clear if-else statements, easier for beginners to understand. Version 2 uses a lambda function for compactness but may reduce readability.

The screenshot shows a VS Code interface with the following details:

- Explorer:** Shows a folder named "ASSIGNMENT2.4 AI" containing files: sample.txt, TASK1, TASK2, TASK3, TASK4, and task5.
- Code Editor:** Task5 file contains Python code to check if a year is a leap year. The code includes a function `is_leap_year` and example usage for the year 2024.
- Terminal:** Shows the output of running the script with Python 3.11.2. The terminal output is:

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI> & C:\Users\nishc\AppData\Local\Programs\Python\Python311\python.exe "c:/Users/nishc/OneDrive/Documents/Desktop/ASSIGNMENT2.4 AI/task5"
2024 is a leap year.
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI>
```

The screenshot shows a VS Code interface with the following details:

- Explorer:** Shows a folder named "ASSIGNMENT2.4 AI" containing files: sample.txt, TASK1, TASK2, TASK3, TASK4, and task5.
- Code Editor:** Task5 file contains Python code for a one-liner leap year check using a lambda function. It also includes example usage for the year 2024.
- Terminal:** Shows the output of running the script with Python 3.11.2. The terminal output is:

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI> & C:\Users\nishc\AppData\Local\Programs\Python\Python311\python.exe "c:/Users/nishc/OneDrive/Documents/Desktop/ASSIGNMENT2.4 AI/task5"
2024 is a leap year.
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI> & C:\Users\nishc\AppData\Local\Programs\Python\Python311\python.exe "c:/Users/nishc/OneDrive/Documents/Desktop/ASSIGNMENT2.4 AI/task5"
2024 is a leap year.
Using one-liner: 2024 is a leap year? True
PS C:\Users\nishc\OneDrive\Documents\Desktop\ASSIGNMENT2.4 AI>
```

Output:

Input: 2024 → Output: True

observation: Version 1 is better for readability and teaching, while Version 2 is concise for experienced coders.

