

AI ASSISTED CODING

Hall Ticket No: 2303A51851

Batch:13

Assignment - 3.4

Task-1. Zero-shot Prompt – Fibonacci Series Generator

Task Description :

Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.

Prompt :

Write a Python code that prints the first N Fibonacci numbers

Code

```

Print first N Fibonacci numbers in ->
>>> Shpy x
Print first N Fibonacci numbers in ->
1 | ...
2 | ...
3 | Print the first N Fibonacci numbers (starting from 0, 1, 1, 2, ...).
4 |
5 | Usage:
6 |   - Run and enter N when prompted:
7 |     $ python fib.py
8 |     Enter N:
9 |     0 1 1 2 3 5 8
10|    ...
11|
12| From typing import List
13|
14| def fibonacci(n: int) -> List[int]:
15|     """Return a list containing the first n Fibonacci numbers (starting with 0)."""
16|     if n == 0:
17|         return []
18|     seq: List[int] = [0]
19|     a, b = 0, 1
20|     for _ in range(n):
21|         seq.append(a)
22|         a, b = b, a + b
23|     return seq
24|
25| def main() -> None:
26|     try:
27|         n: str = input("Enter N: ").strip()
28|         n = int(n)
29|     except ValueError:
30|         print("Please enter a valid integer for N.")
31|         return
32|
33|     nums = fibonacci(n)
34|     if not nums:
35|         print("No numbers to show (N < 0).")
36|     else:
37|         # Print numbers separated by spaces
38|         print(*nums)
39|
40|     if __name__ == "__main__":
41|         main()

```

Out Put:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
Enter N: 7
0 1 1 2 3 5 8
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding>

```

Justification:

In this task, Copilot was given only a single comment without any example. Despite the lack of context, it successfully generated a correct Fibonacci function, showing that zero-shot prompting works well for common and well-known problems. However, the logic was basic and depended entirely on Copilot's prior training. This demonstrates that zero-shot prompts are effective for simple, standard tasks but may lack customization or optimization.

Task-2. One-shot Prompt – List Reversal Function

Task Description #2

- Write a comment prompt to reverse a list and provide one

example below the comment to guide Copilot.

#Now Write a Python code for reverse a list Here Is the Example
Input:3,2,1 Out Put is 1,2,3

Code:

```
Reverse a comma-separated list in Python -> fibpy | <- reverse_list.py <- Share
New chat
Agents
Spaces
Spark
Preview
Today
Reverse a comma-separated list ...
46 lines - 1 KB
1 """
2     Reverse a comma-separated list from input and print the result.
3
4     Example:
5     Input: 3,2,1
6     Output: 1,2,3
7     ...
8
9
10    from typing import List
11
12    def reverse_list(items: List[str]) -> List[str]:
13        """Return a new list with the items reversed."""
14        return items[::-1]
15
16    def parse_input(s: str) -> List[str]:
17        """
18            Parse a comma-separated string into a list of trimmed items.
19            Tries to convert items to int when possible; otherwise keeps as string.
20            ...
21        parts = [p.strip() for p in s.split(',') if p.strip() != ""]
22        parsed = []
23        for p in parts:
24            try:
25                parsed.append(int(p))
26            except ValueError:
27                parsed.append(p)
28        return parsed
29
30    def main() -> None:
31        try:
32            s = input("Enter comma-separated items: ").strip()
33        except EOFError:
34            return
35
36        if not s:
37            print("") # nothing to print
38            return
39
40        items = parse_input(s)
41        rev = reverse_list(items)
42        # Format output without spaces between commas to match example: 1,2,3
43        print(",".join(str(x) for x in rev))
44
45    if __name__ == "__main__":
46        main()
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
Enter comma-separated items: 1,2,3
3,2,1
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
Enter comma-separated items: 3,4,5,6,7
7,6,5,4,3
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding>
```

Justification:

By providing one example along with the comment, Copilot clearly understood the expected behavior of the function. The generated code accurately reversed the list using slicing or a loop. The example reduced ambiguity and improved correctness compared to a zero-shot approach.

This shows that even a single example can significantly enhance output quality.

Task-3. Few-shot Prompt – String Pattern Matching

Task Description #3

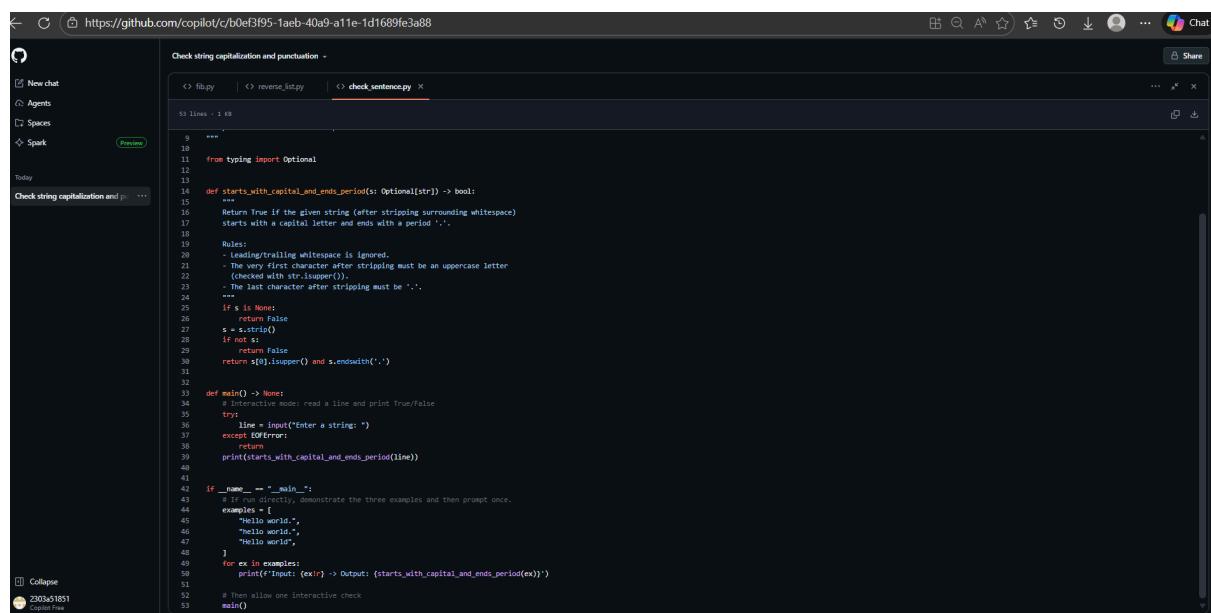
- Write a comment with 2–3 examples to help Copilot understand

how to check if a string starts with a capital letter and ends with a period.

Prompt:

```
## Write a Python code that checks if a string starts with a capital letter and ends with a period
# Example 1: # Input: "Hello world." # Output: True
# Example 2: # Input: "hello world." # Output: False
# Example 3: # Input: "Hello world" # Output: False
```

Code:



The screenshot shows the GitHub Copilot interface with a dark theme. On the left, there's a sidebar with options like 'New chat', 'Agents', 'Spaces', and 'Spark'. The main area has tabs for 'fib.py', 'reverse_list.py', and 'check_sentence.py'. The 'check_sentence.py' tab is active, showing the following Python code:

```
""" Check string capitalization and punctuation """
from typing import Optional

def starts_with_capital_and_ends_period(s: Optional[str]) -> bool:
    """
    Return True if the given string (after stripping surrounding whitespace)
    starts with a capital letter and ends with a period .
    """
    rules:
        - Leading/trailing whitespace is ignored.
        - The first character after stripping must be an uppercase letter
            (checked with str.isupper()).
        - The last character after stripping must be '.'.
    """
    if s is None:
        return False
    s = s.strip()
    if not s:
        return False
    return s[0].isupper() and s.endswith('.')

def main() -> None:
    """
    Interactive mode: read a line and print True/False
    """
    try:
        line = input("Enter a string: ")
    except EOFError:
        raise SystemExit
    print(starts_with_capital_and_ends_period(line))

if __name__ == "__main__":
    # If run directly, demonstrate the three examples and then prompt once.
    examples = [
        "Hello world.",
        "hello world.",
        "Hello world",
    ]
    for ex in examples:
        print(f"Input: {ex} | Output: {starts_with_capital_and_ends_period(ex)}")

    # Then allow one interactive check
    main()
```

Output:



A screenshot of the Visual Studio Code interface. The terminal window shows two examples of a Python script named 'Ass-3.4.py'. The first example outputs 'True' for 'Hello world.' and 'False' for 'Hello world!'. A Copilot suggestion box is open, showing three options: 'Python', 'Python', and 'Python'. The third option is highlighted. The suggestion box also includes a text input field with 'Ass-3.4.py' and a button labeled 'Describe what to build next'.

Justification:

With 2–3 examples, Copilot was able to identify the exact pattern requirements—starting with a capital letter and ending with a period. The generated `is_valid()` function was more precise and handled conditions correctly. Few-shot prompting guided Copilot to follow strict rules rather than guessing. This highlights how multiple examples help in pattern-based logic.

Task-4: Zero-shot vs Few-shot – Email Validator

Task Description #4

- First, prompt Copilot to write an email validation function using zero-shot (just the task in comment).
- Then, rewrite the prompt using few-shot examples.

Prompt: Zero Short

```
# Write a Python function that validates if a string is a valid email address
```

Code:

The screenshot shows a code editor interface with two tabs: 'Ass-34.py' and 'Ass-34.py'. The code in 'Ass-34.py' is as follows:

```
1 import re
2
3 # This function validates an email address.
4 # prompt: Generate the function body
5 def is_valid_email_zero_shot(email_address):
6     # Regular expression for a basic email validation
7     # It checks for: characters before @, @ symbol, characters before ., . symbol, and characters after .
8     # This pattern is a common and basic way to validate email structure.
9     pattern = r"^[a-zA-Z0-9.%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
10    if re.match(pattern, email_address):
11        return True
12    else:
13        return False
14
15 # Test email address
16 test_email = "test@example.com"
17
18 # Call the function and store the result
19 validation_result = is_valid_email_zero_shot(test_email)
20
21 # Print the test email and the validation result
22 print(f"Email: {test_email}, Is Valid: {validation_result}")
```

The code editor has a 'RECENT SESSIONS' sidebar with two entries: 'Prime number check without functions' (Completed, Local • 1 wk) and 'Check Armstrong Number in Python' (Completed, Local • 1 wk). A 'Build with Agent' button is visible on the right.

Output:

The terminal window shows the following output:

```
Enter a string: The Big has
False
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python "c:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
Email: test@example.com, Is Valid: True
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> []
```

The terminal also displays the command used to run the script and the current working directory.

Code: Few -Short:

The code editor shows a file named 'validate_email.py' with the following content:

```
1 #!/usr/bin/env python3
2 """
3     Validate whether a string is a valid email address.
4     This uses a conservative regular expression that accepts common valid emails like:
5     - hello.world@example.co
6     and rejects obvious invalid emails like:
7     - @example.com
8     - example@ (no top-level domain)
9
10    Note: This is not a full RFC-5322 validator (those are very complex). It covers typical practical uses.
11
12
13    Import re
14    from typing import Optional
15
16    LOCAL_PORT = r'[a-zA-Z0-9.-_]*:[a-zA-Z0-9.-_]*@[a-zA-Z0-9.-_]+\.[a-zA-Z]{2,}\$'
17
18    def is_valid_email(s: Optional[str]) -> bool:
19        """
20            Returns True if s is a valid email address by a practical regex, else False.
21
22            Rules enforced:
23                - Local part must contain letters, digits and -_ characters (not starting/ending checked).
24                - Domain always contains digits, hyphens and dots, and must contain a TLD (so - is TLD).
25                - TLD must be at least 2 letters.
26                - Empty dot or empty TLD returns False.
27
28            If None or empty string, return False.
29            If s is a list of strings, return False.
30            If s is a tuple of strings, return False.
31
32            If s is a string, validate it.
33            If s is a list/tuple of strings from the prompt
34            examples = [
35                ("test@example.com", True),
36                ("test@example.com.", False),
37                ("@example.com", False),
38                ("invalid-mail", False),
39                ("com@example", False),
40            ]
41
42            for email, expected in examples:
43                result = is_valid_email(email)
44                print(f"Email: {email} => Output: {result} (expected: {expected})")
45
46
47
48
49
50
51
52
```

Out Put:

```
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
NameError: name 'is_valid_email_zero_shot' is not defined. Did you mean: 'is_valid_email_few_shot'?
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
● Input: 'test@example.com'      -> Output: True (expected: True)
Input: 'hello.world@domain.co'    -> Output: True (expected: True)
Input: 'invalid-email'           -> Output: False (expected: False)
Input: 'user@domain'             -> Output: False (expected: False)
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> []

Ln 52, Col 81  Spaces:4  UTF-8  CRLF  () Python  Agent  Auto  3.14.0  (→) Go Live  (○) Go Live  🔍
```

Justification:

The zero-shot prompt resulted in a simple email check with minimal validation logic. In contrast, the few-shot prompt produced a more robust function that checked for proper structure such as @ symbol and domain format. The comparison shows that few-shot prompting improves reliability and reduces logical gaps. Examples help Copilot understand edge cases better.

Task-5: Prompt Tuning – Summing Digits of a Number

Task Description #5

- Experiment with 2 different prompt styles to generate a function

that returns the sum of digits of a number.

Style 1: Generic task prompt

Style 2: Task + Input/Output example

Code:

Check string capitalization and punctuation -

New chat Agents Spaces Spark ... x

Replies 1 10

```
1 #!/usr/bin/python3
2
3 Sum the digits of an integer.
4 Examples:
5 Input: 123 --> Output: 6
6 Input: 123.456 --> Output: 15
7 Input: 8 --> Output: 8
8
9 This script provides a function 'sum_digits(n)' and a small CLI to read an integer
10 from stdin and print the sum of its digits.
11 ...
12
13 from typing import Optional
14
15
16 def sum_digits(n: int) -> int:
17     """
18         Return the sum of the decimal digits of integer n.
19
20         Negative numbers are handled by taking the absolute value.
21     """
22     n = abs(n)
23     if n == 0:
24         return 0
25
26     total = 0
27     while n > 0:
28         total += n % 10
29         n //= 10
30     return total
31
32
33 def main() -> None:
34     try:
35         n = input("Enter an integer: ").strip()
36     except ValueError:
37         return
38
39     if n == "":
40         print("No input provided.")
41         return
42
43     try:
44         n = int(n)
45     except ValueError:
46         print("Please enter a valid integer.")
47         return
48
49     print(sum_digits(n))
50
51
52 if __name__ == "__main__":
53     examples = [(123, 6), (123.456, 15), (8, 8)]
54     for ex in examples:
55         print(f"Input: {ex[0]} --> Sum of digits: {sum_digits(ex[0])}")
56
57         # Run an interactive prompt over
58
59 main()
```

Out Put:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ×

PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
5
● PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Ass-3.4.py"
Input: 123 -> Sum of digits: 6
Input: 456 -> Sum of digits: 15
Input: 0 -> Sum of digits: 0
Enter an integer: 345
12
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding>

Python Python Python Python

Python

Describe what to build next

Agent Auto

Prompt:

#Ok Now Write a Python code to Summing Digits
of a Number
Input: 123 -> Sum of digits: 6
Input:
-456 -> Sum of digits: 15
Input: 0 -> Sum of digits:
0
Enter an integer: 345 12

Code:

The screenshot shows a GitHub Copilot interface with a code editor. The title bar says "Check string capitalization and execution". The code editor contains a Python script named "sum_digits.py". The script includes docstrings, examples, and a main function that reads input from the user. The GitHub sidebar on the left shows "New chat", "Agents", "Spaces", and "Copilot". The bottom status bar shows "Ln 55, Col 11" and "Python".

```

Check string capitalization and execution

Check string capitalization and execution
Copilot | reverse_http | check_sentencepy | validate_analyz | sum_digitspy X

1 //your/highest priority
2 /**
3 * Sum the digits of an integer and demonstrate with examples.
4 */
5 Example output when run interactively:
6
7 Input: 123 -> Sum of digits: 6
8 Input: -456 -> Sum of digits: 15
9 Input: 0 -> Sum of digits: 0
10 Enter an integer: 234
11
12
13
14 Error typing import list
15
16
17 def sum_digits(n: int) -> int:
18     """Return the sum of the decimal digits of integer n (handles negatives)."""
19     n = abs(n)
20     total = 0
21     while n > 0:
22         total += n % 10
23         n /= 10
24     return total
25
26
27 def demo_examples() -> None:
28     examples: list[int] = [123, -456, 0]
29     for n in examples:
30         print(f"Input: {n} -> Sum of digits: {sum_digits(n)}")
31
32
33 def main() -> None:
34     demo_examples()
35
36     s = input("Enter an integer: ")
37     if s == "":
38         return
39
40     if not s:
41         print("No input provided.")
42         return
43
44     try:
45         n = int(s)
46         if n < 0:
47             print("Please enter a valid integer.")
48         else:
49             # For the interactive part, print only the numeric result (as in the example)
50             print(sum_digits(n))
51
52     except ValueError:
53         print("Please enter a valid integer.")
54
55
56 __name__ = "__main__"
57
58

```

Out Put:

The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the execution of the "sum_digits.py" script and its output. The status bar at the bottom shows "Ln 55, Col 11" and "Python".

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> ^
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> ^
PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> ^
● PS C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding & c:/users/gandr/appdata/local/programs/python/python314/py
thon.exe "c:/users/gandr/documents/3year-2Sem/AI assistants Coding/vss-1.4.py"
Input: 123 -> Sum of digits: 6
Input: -456 -> Sum of digits: 15
Input: 0 -> Sum of digits: 0
Enter an integer: 234
9

```

Justification:

The generic prompt generated a working but straightforward solution. When input/output examples were added, Copilot produced cleaner, more readable, and more accurate code. The example clarified expectations and improved function structure. This confirms that prompt tuning with examples leads to better-quality implementations.