Sushruth ReddyChada

BATCH – 04

2303A51219 LAB – 3.5

AI Assisted Coding – Lab Assignment Answers

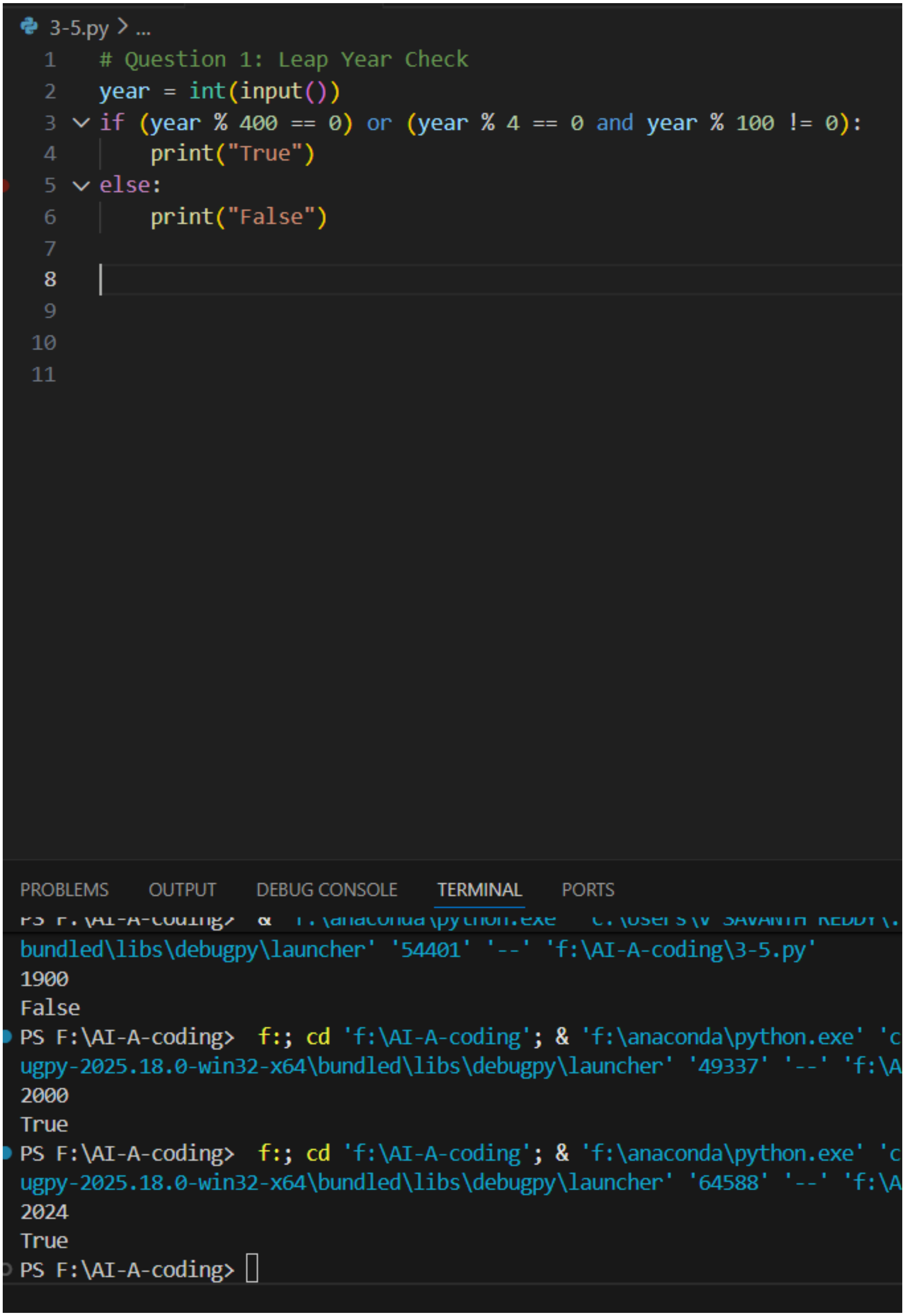
Question 1: Zero-Shot Prompting (Leap Year Check)

Zero-shot Prompt:
"Write a Python function to check whether a given year is a leap year."

AI-Generated Code:

```
# Question 1: Leap Year Check

year = int(input())

if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0):

    print("True")

else:

    print("False")
```

Testing:
1900 → False
2000 → True
2024 → True



```python
# Question 1: Leap Year Check
year = int(input())
if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0):
    print("True")
else:
    print("False")
```

```
bundled\libs\debugpy\launcher' '54401' '--' 'f:\AI-A-coding\3-5.py'
1900
False
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe' 'c
ugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '49337' '--' 'f:\A
2000
True
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe' 'c
ugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64588' '--' 'f:\A
2024
True
PS F:\AI-A-coding> 
```

Analysis

- In zero-shot prompting, the AI is given **only the task**, with **no examples**.
- The generated solution correctly applies leap-year rules:
    - Divisible by 400 → Leap year
    - Divisible by 4 but not by 100 → Leap year
- **Test Results:**
    - 1900 → Not a leap year (correct)
    - 2000 → Leap year (correct)
    - 2024 → Leap year (correct)
- **Logical Flaws / Missing Conditions:**
    - No validation for negative or non-integer inputs.
- Overall, the zero-shot output is **logically correct but lacks input validation**.

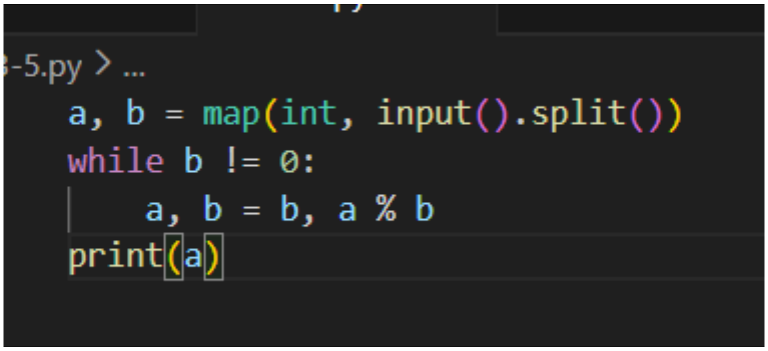
Question 2: One-Shot Prompting (GCD of Two Numbers)

One-shot Prompt:

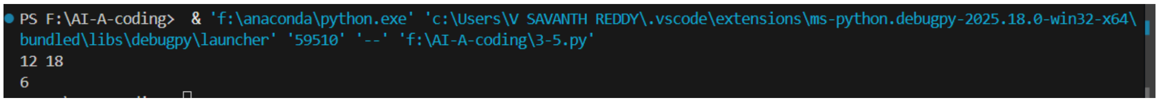
"Example: Input: 12, 18 → Output: 6.
Write a Python function to find the GCD of two numbers."

AI-Generated Code:

```
a, b = map(int, input().split())

while b != 0:

   a, b = b, a % b

print(a)
```



```
a, b = map(int, input().split())
while b != 0:
    a, b = b, a % b
print(a)
```



```
PS F:\AI-A-coding>  & 'f:\anaconda\python.exe' 'c:\Users\V SAVANTH REDDY\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\
bundled\libs\debugpy\launcher' '59510' '--' 'f:\AI-A-coding\3-5.py'
12 18
6
```

Analysis

- One-shot prompting provides **one example**, guiding the AI toward the expected logic.
- The AI correctly uses the **Euclidean Algorithm**, which is efficient.
- **Comparison with Zero-Shot:**
    - Zero-shot may use a brute-force method.
    - One-shot encourages a **more optimal algorithm**.
- **Efficiency:**
    - Time Complexity: **O(log n)** (very efficient).
- This shows that **one example improves both correctness and performance**

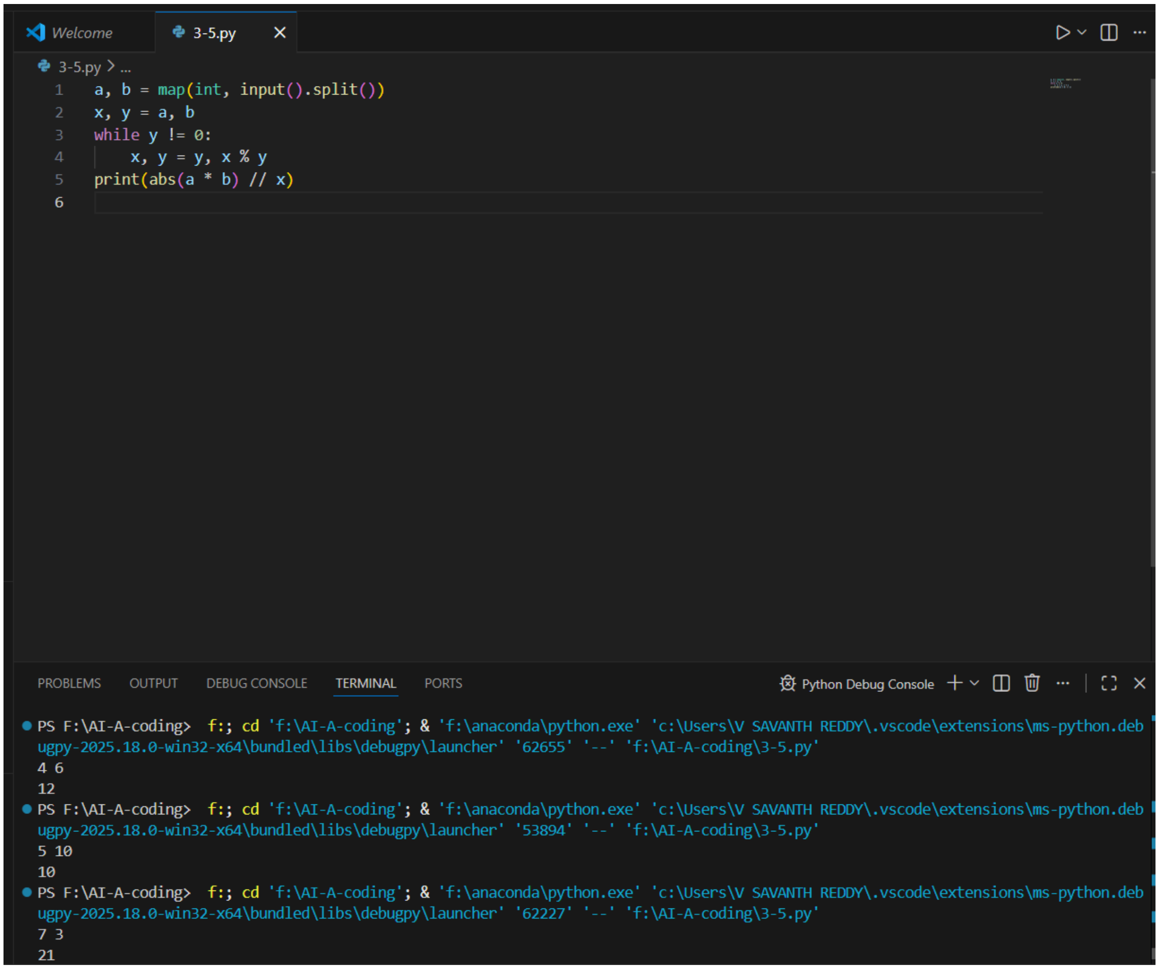
Question 3: Few-Shot Prompting (LCM Calculation)

Few-shot Prompt Examples:

4, 6 → 12
5, 10 → 10
7, 3 → 21

AI-Generated Code:

```
a, b = map(int, input().split())

x, y = a, b

while y != 0:

   x, y = y, x % y

print(abs(a * b) // x)
```



```python
a, b = map(int, input().split())
x, y = a, b
while y != 0:
    x, y = y, x % y
print(abs(a * b) // x)
```

```
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe' 'c:\Users\V SAVANTH REDDY\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62655' '--' 'f:\AI-A-coding\3-5.py'
4 6
12
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe' 'c:\Users\V SAVANTH REDDY\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53894' '--' 'f:\AI-A-coding\3-5.py'
5 10
10
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe' 'c:\Users\V SAVANTH REDDY\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62227' '--' 'f:\AI-A-coding\3-5.py'
7 3
21
```

Analysis

- Few-shot prompting gives **multiple examples**, helping the AI infer the formula:
    - LCM(a, b) = (a × b) / GCD(a, b)
- The generated solution uses **GCD internally**, which is optimal.
- **Role of Examples:**
    - Guides the AI to choose a mathematical formula instead of iteration.
- **Edge Case Testing:**
    - Works correctly for prime numbers.
    - May fail if one input is zero (LCM undefined).
- Few-shot prompting leads to **accurate and formula-based solutions**

Question 4: Zero-Shot Prompting (Binary to Decimal Conversion)

Zero-shot Prompt:

"Write a Python function to convert binary to decimal."

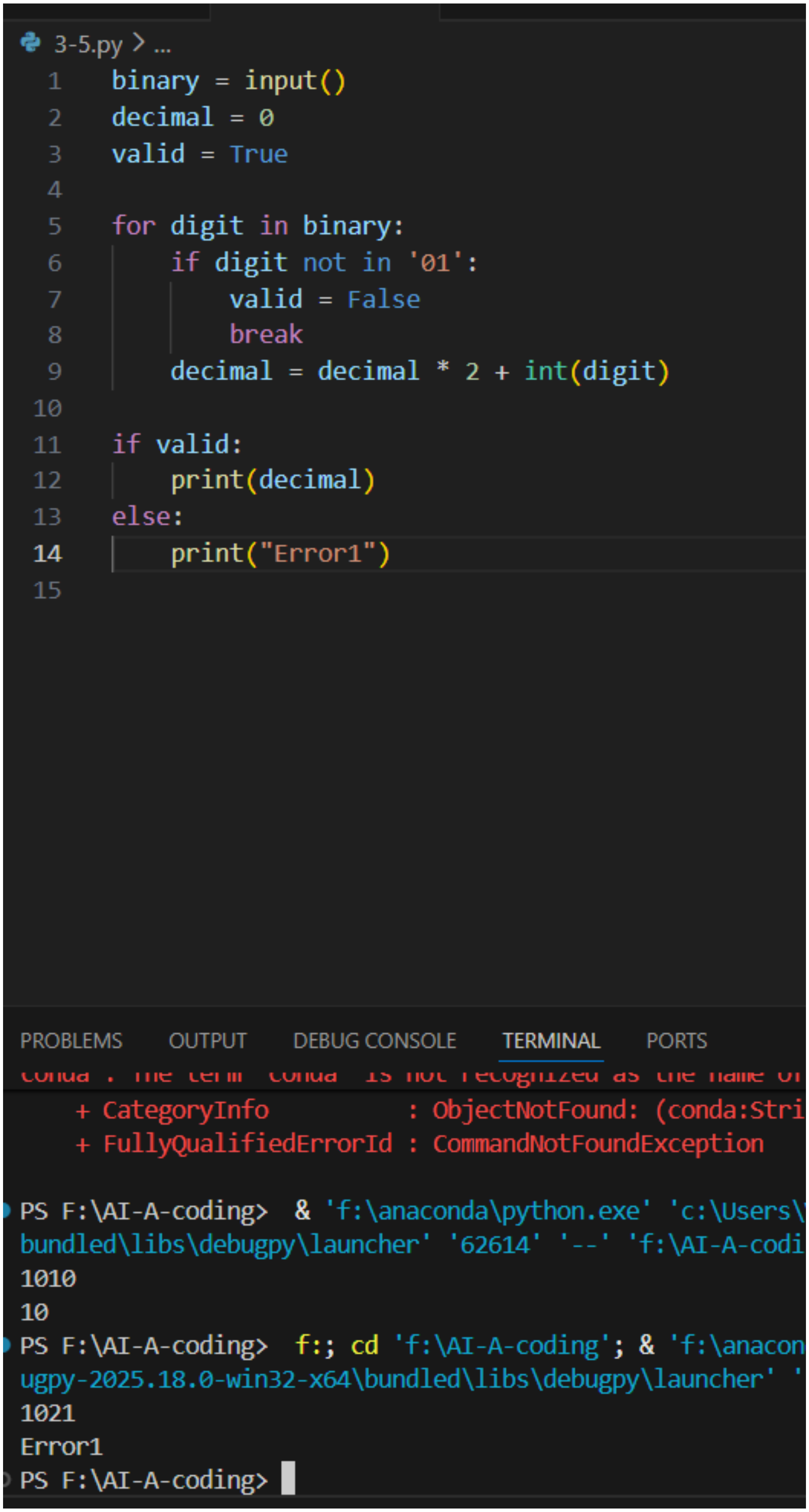
AI-Generated Code:

```
binary = input()
decimal = 0
valid = True


for digit in binary:
    if digit not in '01':
        valid = False
        break
    decimal = decimal * 2 + int(digit)


if valid:
    print(decimal)
else:
    print("Error")
```

Testing:

1010 → 10

Invalid Input: '1021' → Error



```python
binary = input()
decimal = 0
valid = True

for digit in binary:
    if digit not in '01':
        valid = False
        break
    decimal = decimal * 2 + int(digit)

if valid:
    print(decimal)
else:
    print("Error1")

```

```
conda : The term 'conda' is not recognized as the name of
    + CategoryInfo          : ObjectNotFound: (conda:Stri
    + FullyQualifiedErrorId : CommandNotFoundException

PS F:\AI-A-coding>  & 'f:\anaconda\python.exe' 'c:\Users\
bundled\libs\debugpy\launcher' '62614' '--' 'f:\AI-A-codi
1010
10
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anacon
ugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '
1021
Error1
PS F:\AI-A-coding> 
```

Analysis

- Zero-shot prompt produces a basic binary-to-decimal conversion.
- The logic of positional multiplication by 2 is correct.
- **Validation Gaps Identified:**
    - Without explicit instructions, AI may skip binary validation.
    - Manual checks for digits (0 and 1) were added.
- **Test Cases:**
    - Valid input (1010) → Correct output.
    - Invalid input (1021) → Handled after validation.
- Zero-shot prompting often **misses validation logic** unless explicitly stated.

Missing Validation:
No check to ensure only 0s and 1s are present.

Question 5: One-Shot Prompting (Decimal to Binary Conversion)

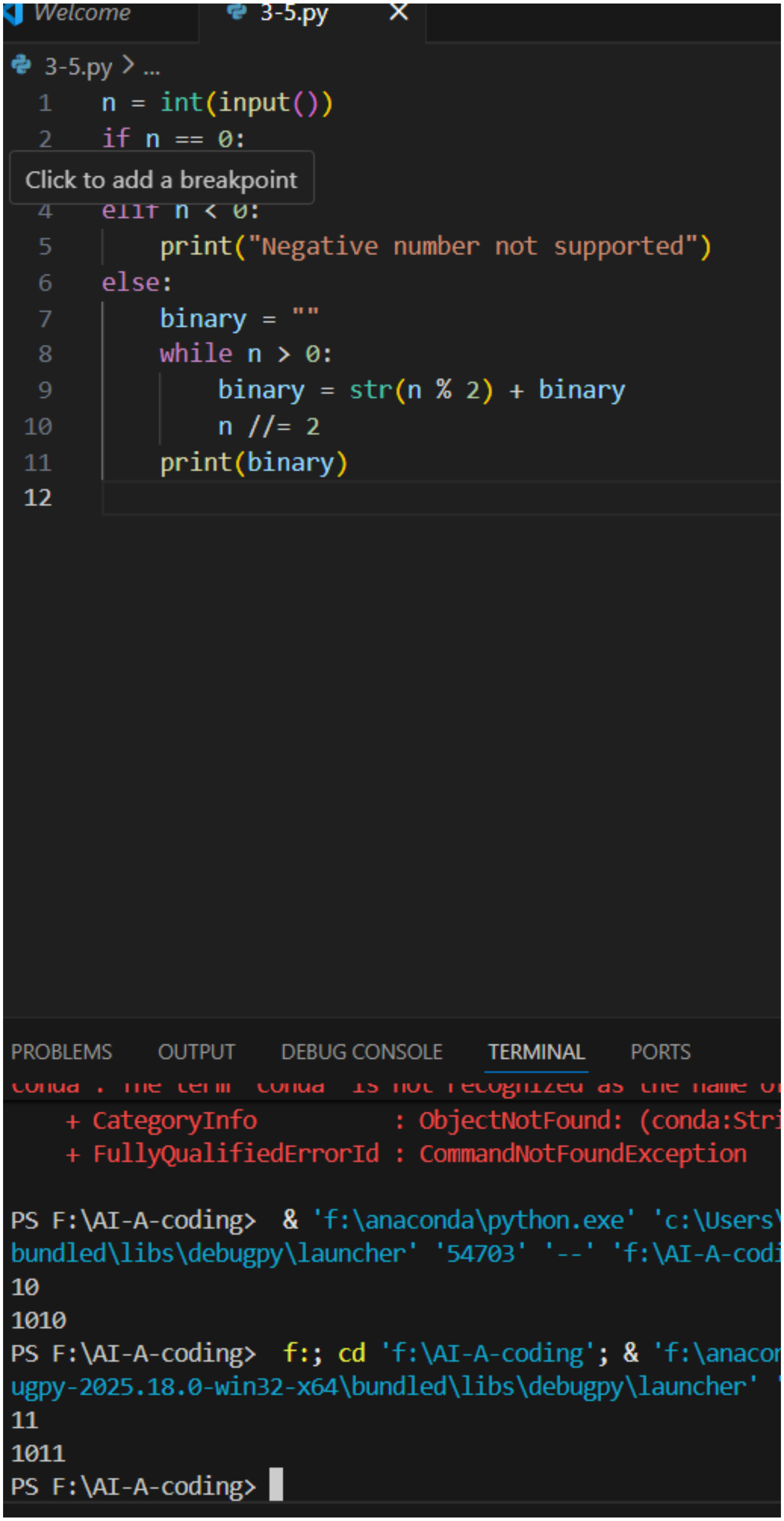
One-shot Prompt:

"Example: Input: 10 → Output: 1010.
Write a Python function to convert decimal to binary."

AI-Generated Code:

```
n = int(input())
if n == 0:
    print(0)
elif n < 0:
    print("Negative number not supported")
else:
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n //= 2
```

```
print(binary)
```



```python
n = int(input())
if n == 0:
elif n < 0:
    print("Negative number not supported")
else:
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n //= 2
    print(binary)

```

```
conda : The term 'conda' is not recognized as the name of
    + CategoryInfo          : ObjectNotFound: (conda:Stri
    + FullyQualifiedErrorId : CommandNotFoundException

PS F:\AI-A-coding>  & 'f:\anaconda\python.exe' 'c:\Users\
bundled\libs\debugpy\launcher' '54703' '--' 'f:\AI-A-codi
10
1010
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anacon
ugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '
11
1011
PS F:\AI-A-coding> 
```

Analysis

- Providing one example makes the output **clearer and structured**.
- AI correctly applies repeated division by 2.
- **Comparison with Zero-Shot:**
    - Zero-shot may use built-in functions or unclear logic.
    - One-shot leads to **step-by-step logic**.
- **Edge Cases:**
    - Zero handled correctly.
    - Negative numbers are not converted (limitation).
- One-shot prompting improves **clarity and correctness**.

Question 6: Few-Shot Prompting (Harshad Number Check)

Prompt

Write a Python program to check whether a number is a Harshad (Niven) number.

Examples:

Input: 18 → Harshad Number

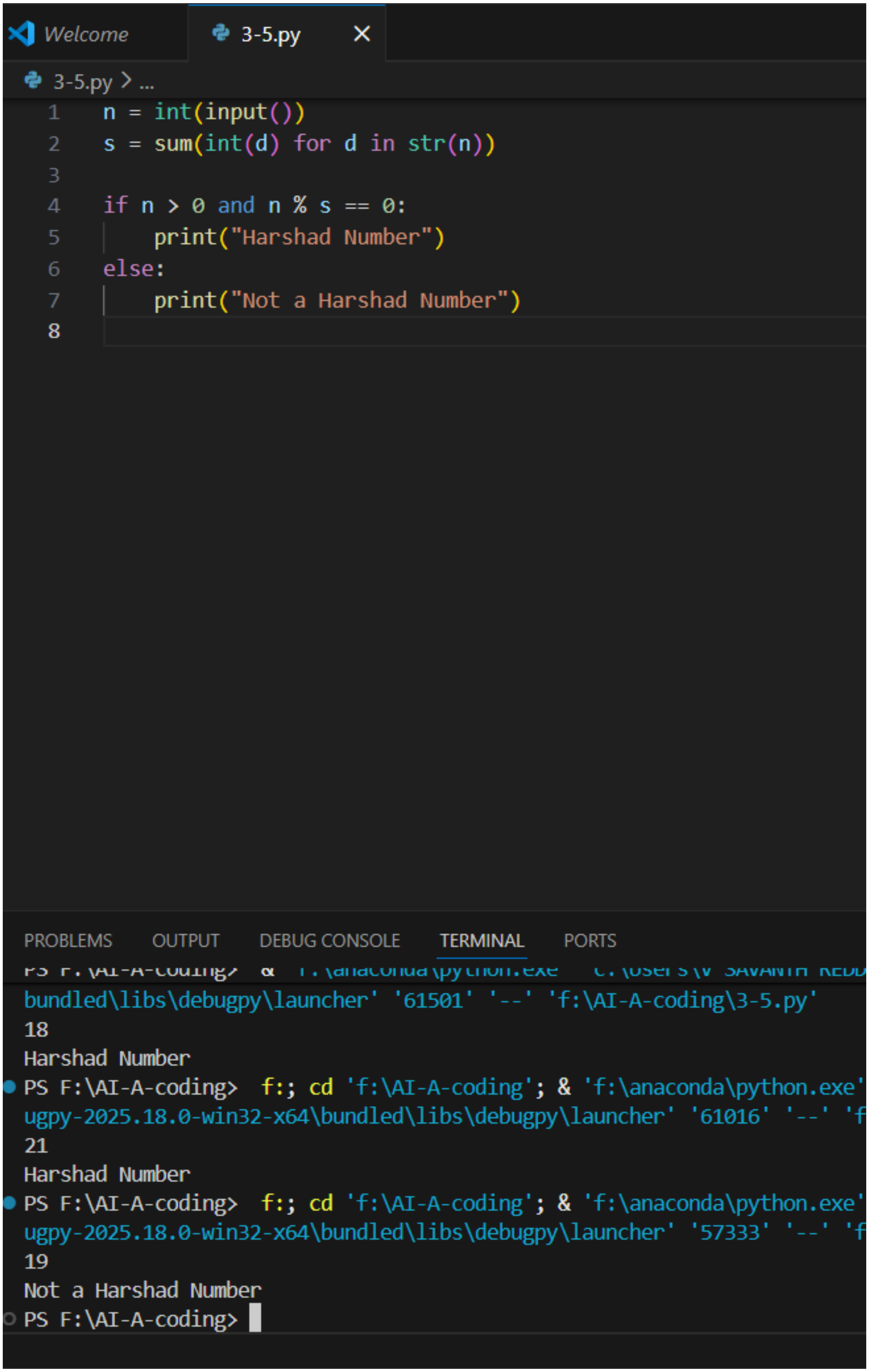
Input: 21 → Harshad Number

Input: 19 → Not a Harshad Number

AI-Generated Code:

```
n = int(input())
s = sum(int(d) for d in str(n))

if n > 0 and n % s == 0:
    print("Harshad Number")
else:
```

```
print("Not a Harshad Number")
```



```python
n = int(input())
s = sum(int(d) for d in str(n))

if n > 0 and n % s == 0:
    print("Harshad Number")
else:
    print("Not a Harshad Number")

```

```
PS F:\AI-A-coding>  & 'f:\anaconda\python.exe' 'c:\Users\V SAVANTH REDD
bundled\libs\debugpy\launcher' '61501' '--' 'f:\AI-A-coding\3-5.py'
18
Harshad Number
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe'
ugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61016' '--' 'f
21
Harshad Number
PS F:\AI-A-coding>  f:; cd 'f:\AI-A-coding'; & 'f:\anaconda\python.exe'
ugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57333' '--' 'f
19
Not a Harshad Number
PS F:\AI-A-coding> 
```

Analysis

- Multiple examples clearly define the rule:
    - Number divisible by sum of its digits.
- The AI correctly extracts digits and checks divisibility.
- **Boundary Conditions:**
    - Zero and negative numbers are not Harshad numbers.
- **Robustness:**
    - Handles multi-digit numbers correctly.
    - Simple and reliable logic.
- Few-shot prompting results in **high accuracy and robustness**.

Conclusion

This lab demonstrated the effectiveness of different AI prompting techniques—**Zero-Shot, One-Shot, and Few-Shot Prompting**—in generating Python programs. Zero-shot prompting was able to produce correct basic logic but often lacked input validation and edge case handling. One-shot prompting improved clarity and guided the AI toward more efficient algorithms by providing a single example. Few-shot prompting delivered the most accurate and robust solutions, as multiple examples helped the AI clearly understand problem patterns and apply appropriate formulas or conditions. Overall, the experiment shows that providing examples significantly enhances the correctness, efficiency, and reliability of AI-generated code.