

AI ASSISTED CODING

ASSIGNMENT-6.1

2403A51260

SWAPNA MADISHETTI

BATCH-11

Task-1:

Prompt: Create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary. class should contain constructor, display_details(), and calculate_bonus() methods.

Code:

```
AI_6_1_T1.PY > ...
1 class Employee:
    Tabnine | Edit | Test | Explain | Document
2     def __init__(self, name, emp_id, salary):
3         self.name = name
4         self.emp_id = emp_id
5         self.salary = salary # Monthly salary
6
7     Tabnine | Edit | Test | Explain | Document
8     def display_details(self):
9         print(f"Name: {self.name}")
10        print(f"ID: {self.emp_id}")
11        print(f"Monthly Salary: {self.salary}")
12        print(f"Yearly Salary: {self.yearly_salary()}")
13
14    Tabnine | Edit | Test | Explain | Document
15    def yearly_salary(self):
16        return self.salary * 12
17
18    Tabnine | Edit | Test | Explain | Document
19    def calculate_bonus(self, bonus_percent):
20        bonus = self.salary * bonus_percent / 100
21        print(f"Bonus amount: {bonus}")
22        print(f"Total after bonus (monthly): {self.salary + bonus}")
23
24    # Example usage
25    emp = Employee("Alice", 101, 5000)
26    emp.display_details()
27    emp.calculate_bonus(10)
```

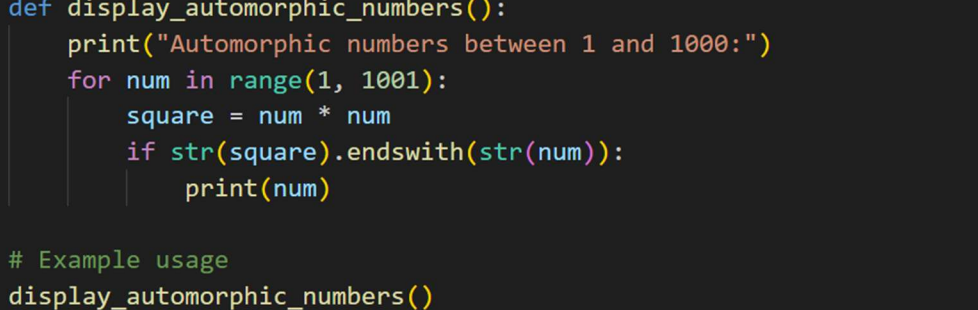
Output:

```
Python: AI_6_1_T1.PY
PROBLEMS OUTPUT TERMINAL ...
● PS C:\AICoding> & "C:/Program Files/Python313/python.exe" c:/AICoding/AI_6_1_T1.PY
Name: Alice
ID: 101
Monthly Salary: 5000
Yearly Salary: 60000
Bonus amount: 500.0
Total after bonus (monthly): 5500.0
○ PS C:\AICoding>
```

Task-2:

Prompt-1: Generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Code & Output:



The screenshot shows a VS Code editor with a Python file named `ai_6_1_t2.py`. The code defines a function `display_automorphic_numbers()` that prints automorphic numbers between 1 and 1000. The function iterates through numbers from 1 to 1000, calculates their squares, and checks if the square ends with the original number. An example usage is shown at the bottom of the code block.

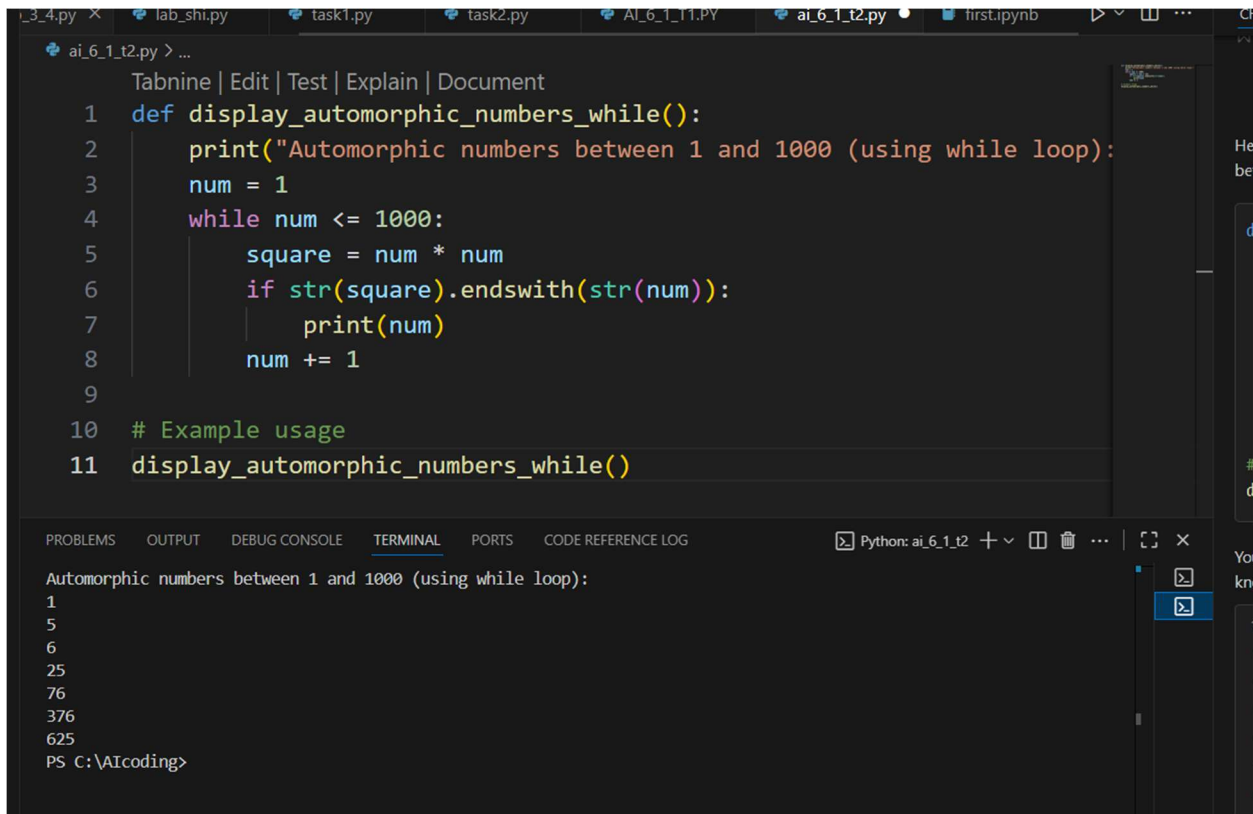
```
1 def display_automorphic_numbers():
2     print("Automorphic numbers between 1 and 1000:")
3     for num in range(1, 1001):
4         square = num * num
5         if str(square).endswith(str(num)):
6             print(num)
7
8 # Example usage
9 display_automorphic_numbers()
10
```

The terminal output shows the execution of the script, displaying the automorphic numbers: 1, 5, 6, 25, 76, 376, and 625.

```
PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t2.py
Automorphic numbers between 1 and 1000:
1
5
6
25
76
376
625
PS C:\AIcoding>
```

Prompt-2: Now generate the code using while loop.

Code & Output:



```
ai_6_1_t2.py > ...
Tabnine | Edit | Test | Explain | Document
1 def display_automorphic_numbers_while():
2     print("Automorphic numbers between 1 and 1000 (using while loop):")
3     num = 1
4     while num <= 1000:
5         square = num * num
6         if str(square).endswith(str(num)):
7             print(num)
8         num += 1
9
10 # Example usage
11 display_automorphic_numbers_while()
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
Python: ai_6_1_t2
Automorphic numbers between 1 and 1000 (using while loop):
1
5
6
25
76
376
625
PS C:\AIcoding>
```

Comparison :

- Both codes achieve the same result: listing Automorphic numbers from 1 to 1000.
- The for loop is more concise and idiomatic for a known range.
- The while loop gives more control over the loop variable and is useful when the end condition is more complex or not based on a simple range.

Task-3:

Prompt: Write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5)

Code & Output:

```
ai_6_1_t3.py > ...
Tabnine | Edit | Test | Explain | Document
1 def classify_feedback(rating):
2     """
3     Classify online shopping feedback based on a numerical rating (1-5).
4     4-5: Positive, 3: Neutral, 1-2: Negative
5     """
6     if rating == 5:
7         return "Positive"
8     elif rating == 4:
9         return "Positive"
10    elif rating == 3:
11        return "Neutral"
12    elif rating == 2:
13        return "Negative"
14    elif rating == 1:
15        return "Negative"
16    else:
17        return "Invalid rating"
18
19 # Example usage
20 for r in [1, 2, 3, 4, 5, 0]:
21     print(f"Rating: {r} -> Feedback: {classify_feedback(r)}")
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG Python: ai_6_1_t3 + - [] ...

```
● PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t3.py
Rating: 1 -> Feedback: Negative
Rating: 2 -> Feedback: Negative
Rating: 3 -> Feedback: Neutral
Rating: 4 -> Feedback: Positive
Rating: 5 -> Feedback: Positive
Rating: 0 -> Feedback: Invalid rating
○ PS C:\AIcoding>
```

Prompt: Rewrite using dictionary-based structure.

Code & Output:

```
ai_6_1_t3.py > ...
1
Tabnine | Edit | Test | Explain | Document
2 def classify_feedback_dict(rating):
3     """
4     Classify online shopping feedback using a dictionary mapping.
5     4-5: Positive, 3: Neutral, 1-2: Negative
6     """
7     mapping = {5: "Positive", 4: "Positive", 3: "Neutral", 2: "Negative", 1: "Negative"}
8     return mapping.get(rating, "Invalid rating")
9
10 # Example usage
11 for r in [1, 2, 3, 4, 5, 0]:
12     print(f"Rating: {r} -> Feedback (dict): {classify_feedback_dict(r)}")
13
```

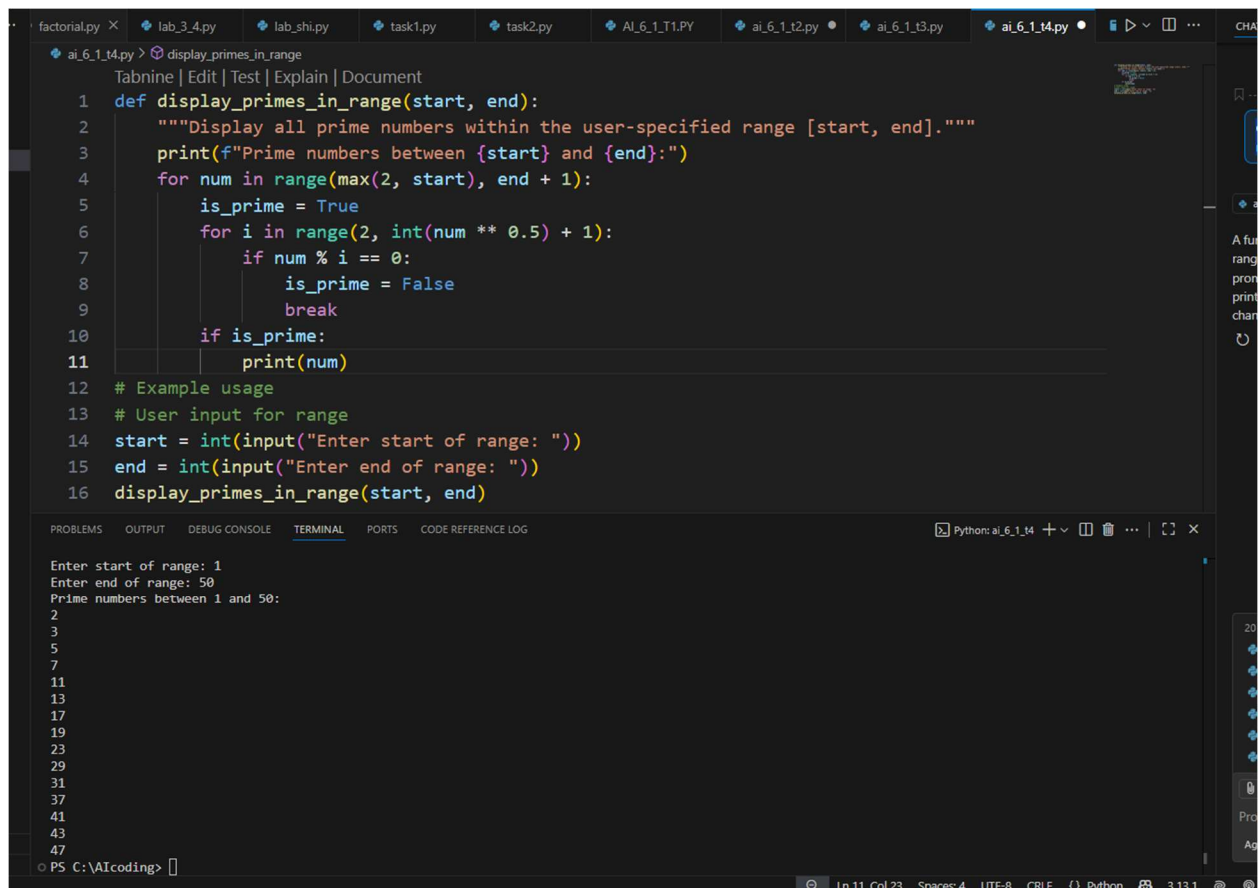
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG Python: ai_6_1_t3

```
PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t3.py
Rating: 1 -> Feedback (dict): Negative
Rating: 2 -> Feedback (dict): Negative
Rating: 3 -> Feedback (dict): Neutral
Rating: 4 -> Feedback (dict): Positive
Rating: 5 -> Feedback (dict): Positive
Rating: 0 -> Feedback (dict): Invalid rating
PS C:\AIcoding>
```

Task-4:

Prompt: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).

Code & Output:



The screenshot shows a Python IDE with a file explorer at the top containing several files: factorial.py, lab_3_4.py, lab_shi.py, task1.py, task2.py, AI_6_1_T1.PY, ai_6_1_t2.py, ai_6_1_t3.py, and ai_6_1_t4.py. The active file is ai_6_1_t4.py, which contains a function named display_primes_in_range. The function takes start and end parameters and prints all prime numbers within that range. It uses a simple trial division method. Below the code editor is a terminal window showing the execution of the program. The user enters a start range of 1 and an end range of 50. The program outputs the prime numbers between 1 and 50: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, and 47.

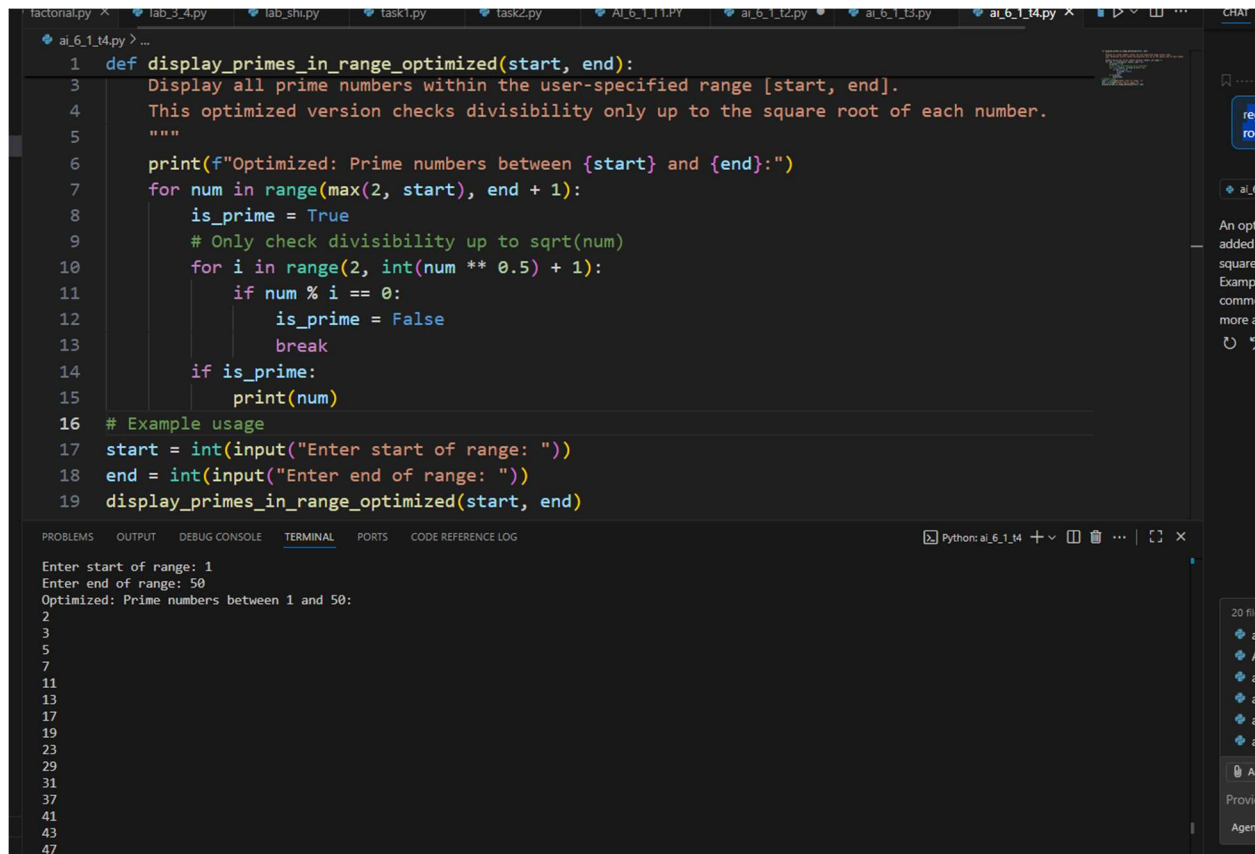
```
def display_primes_in_range(start, end):
    """Display all prime numbers within the user-specified range [start, end]."""
    print(f"Prime numbers between {start} and {end}:")
    for num in range(max(2, start), end + 1):
        is_prime = True
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            print(num)

# Example usage
# User input for range
start = int(input("Enter start of range: "))
end = int(input("Enter end of range: "))
display_primes_in_range(start, end)
```

Enter start of range: 1
Enter end of range: 50
Prime numbers between 1 and 50:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47

Prompt-2: Generate an optimized version (e.g., using the square root method). with explanation.

Code & Output:



```
1 def display_primes_in_range_optimized(start, end):
2     """
3     Display all prime numbers within the user-specified range [start, end].
4     This optimized version checks divisibility only up to the square root of each number.
5     """
6     print(f"Optimized: Prime numbers between {start} and {end}:")
7     for num in range(max(2, start), end + 1):
8         is_prime = True
9         # Only check divisibility up to sqrt(num)
10        for i in range(2, int(num ** 0.5) + 1):
11            if num % i == 0:
12                is_prime = False
13                break
14        if is_prime:
15            print(num)
16
17 # Example usage
18 start = int(input("Enter start of range: "))
19 end = int(input("Enter end of range: "))
20 display_primes_in_range_optimized(start, end)
```

Enter start of range: 1
Enter end of range: 50
Optimized: Prime numbers between 1 and 50:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47

Explanation:

- The function prints all prime numbers between start and end.
- For each number in the range, it checks if the number is prime.
- Instead of checking all numbers up to num-1, it only checks divisibility up to the square root of num ($\text{int}(\text{num} ** 0.5) + 1$). This is because if a number has a factor larger than its square root, the corresponding smaller factor would have already been found.
- If no divisors are found, the number is prime and printed.

Why is this optimized?

- Checking up to the square root reduces the number of checks, making the function much faster for large ranges compared to checking all possible divisors.

Task-5:

Prompt: Build a Library class with methods to `add_book()`, `issue_book()`, and `display_books()`. Add comments and documentation.

Code & Output:


```
lab_shi.py × task1.py task2.py AI_6_1_T1.PY ai_6_1_t2.py ai_6_1_t3.py ai_6_1_t4.py ai_6_1_t5.py × first.ipynb
ai_6_1_t5.py > ...
1 class Library:
2     """
3     A simple Library class to manage books.
4     Methods:
5         add_book(title): Adds a book to the library.
6         issue_book(title): Issues a book if available.
7         display_books(): Displays all available books.
8     """
9
10    Tabnine | Edit | Test | Explain | Document
11    def __init__(self):
12        # Initialize an empty list to store books
13        self.books = []
14
15    Tabnine | Edit | Test | Explain | Document
16    def add_book(self, title):
17        """Add a book to the library."""
18        self.books.append(title)
19        print(f"Book '{title}' added to the library.")
20
21    Tabnine | Edit | Test | Explain | Document
22    def issue_book(self, title):
23        """Issue a book if available."""
24        if title in self.books:
25            self.books.remove(title)
26            print(f"Book '{title}' has been issued.")
27        else:
28            print(f"Book '{title}' is not available.")
29
30    Tabnine | Edit | Test | Explain | Document
31    def display_books(self):
32        """Display all available books in the library."""
33        if self.books:
```

lab_shi.py × task1.py task2.py AI_6_1_T1.PY ai_6_1_t2.py ai_6_1_t3.py ai_6_1_t4.py ai_6_1_t5.py × first.ipynb

ai_6_1_t5.py > ...

```
1 class Library:
18     def issue_book(self, title):
23         else:
24             print(f"Book '{title}' is not available.")
25
26     def display_books(self):
27         """Display all available books in the library."""
28         if self.books:
29             print("Available books:")
30             for book in self.books:
31                 print(f"- {book}")
32         else:
33             print("No books available in the library.")
34
35 # Example usage
36 if __name__ == "__main__":
37     lib = Library()
38     lib.add_book("Python Programming")
39     lib.add_book("Data Science Essentials")
40     lib.display_books()
41     lib.issue_book("Python Programming")
42     lib.display_books()
```

Tabnine | Edit | Test | Explain | Document

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG

Python: ai_6_1_t5

& "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t5.py

Book 'Python Programming' added to the library.
Book 'Data Science Essentials' added to the library.
Available books:
- Python Programming
- Data Science Essentials
○ Book 'Python Programming' has been issued.
Available books:
- Data Science Essentials
PS C:\AIcoding> []