

AI ASSISTED CODING

ASSIGNMENT-6

NAME: G.DEEKSHITH RAJ

HT NO.: 2403A51248

Batch no.: 11

Prompt 1:

Create a Python class named Employee with attributes: name, id, and salary. Include a method to calculate the yearly salary.

Test 1:

- Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.
- Instructions:
 - Prompt AI to generate the Employee class.
 - Analyze the generated code for correctness and structure.
 - Ask AI to add a method to give a bonus and recalculate salary.

```

ALASS6 > TEST1 > ...
1 class Employee:
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     def display_details(self):
8         print(f"Name: {self.name}")
9         print(f"ID: {self.emp_id}")
10        print(f"Monthly Salary: ${self.salary}")
11
12        def calculate_yearly_salary(self):
13            return self.salary * 12
14
15        class Employee:
16            def __init__(self, name, emp_id, salary):
17                self.name = name
18                self.emp_id = emp_id
19                self.salary = salary # Monthly salary
20
21            def display_details(self):
22                print(f"Name: {self.name}")
23                print(f"ID: {self.emp_id}")
24                print(f"Monthly Salary: ${self.salary}")
25
26            def calculate_yearly_salary(self):
27                return self.salary * 12
28
29            def give_bonus(self, bonus_amount):
30                self.salary += bonus_amount
31                print(f"New Monthly Salary after bonus: ${self.salary}")
32                print(f"New Yearly Salary: ${self.calculate_yearly_salary()}")
33
34        class Employee:
35            def __init__(self, name, emp_id, salary):
36                self.name = name
37                self.emp_id = emp_id
38                self.salary = salary # Monthly salary

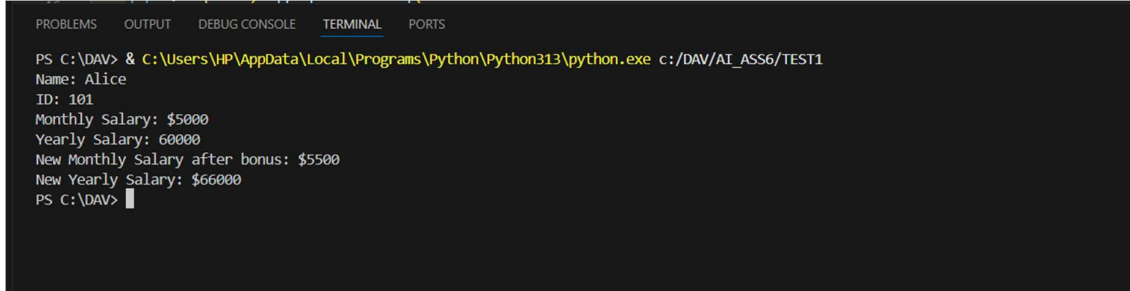
```

```

ALASS6 > TEST1 > Employee > display_details
24
25     def calculate_yearly_salary(self):
26         return self.salary * 12
27
28     def give_bonus(self, bonus_amount):
29         self.salary += bonus_amount
30         print(f"New Monthly Salary after bonus: ${self.salary}")
31         print(f"New Yearly Salary: ${self.calculate_yearly_salary()}")
32
33     class Employee:
34         def __init__(self, name, emp_id, salary):
35             self.name = name
36             self.emp_id = emp_id
37             self.salary = salary # Monthly salary
38
39         def display_details(self):
40             print(f"Name: {self.name}")
41             print(f"ID: {self.emp_id}")
42             print(f"Monthly Salary: ${self.salary}")
43
44         def calculate_yearly_salary(self):
45             return self.salary * 12
46
47         def give_bonus(self, bonus_amount):
48             self.salary += bonus_amount
49             print(f"New Monthly Salary after bonus: ${self.salary}")
50             print(f"New Yearly Salary: ${self.calculate_yearly_salary()}")
51
52     emp1 = Employee("Alice", 101, 5000)
53     emp1.display_details()
54     print("Yearly Salary:", emp1.calculate_yearly_salary())
55     emp1.give_bonus(500)

```

OUTPUT:

A screenshot of a terminal window with a dark background. The terminal shows the execution of a Python script. The prompt is 'PS C:\DAV>' and the command is '& c:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST1'. The output of the script is: 'Name: Alice', 'ID: 101', 'Monthly Salary: \$5000', 'Yearly Salary: 60000', 'New Monthly Salary after bonus: \$5500', and 'New Yearly Salary: \$66000'. The prompt returns to 'PS C:\DAV>' with a cursor.

```
PS C:\DAV> & c:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST1
Name: Alice
ID: 101
Monthly Salary: $5000
Yearly Salary: 60000
New Monthly Salary after bonus: $5500
New Yearly Salary: $66000
PS C:\DAV>
```

Prompt 2:

Write a Python function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Test 2:

- Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
- Instructions:
 - Get AI-generated code to list Automorphic numbers using a for loop.
 - Analyze the correctness and efficiency of the generated logic.
 - Ask AI to regenerate using a while loop and compare both implementations.

```

ALASS6 > TEST2 > ...
1  def is_automorphic(num):
2      square = num ** 2
3      return str(square).endswith(str(num))
4
5  def find_automorphic_numbers_for():
6      print("Automorphic numbers between 1 and 1000 (using for loop):")
7      for i in range(1, 1001):
8          if is_automorphic(i):
9              print(i, end=' ')
10
11 find_automorphic_numbers_for()
12 def find_automorphic_numbers_while():
13     print("\nAutomorphic numbers between 1 and 1000 (using while loop):")
14     i = 1
15     while i <= 1000:
16         if is_automorphic(i):
17             print(i, end=' ')
18         i += 1
19
20 find_automorphic_numbers_while()
21 def is_automorphic(num):
22     square = num ** 2
23     return str(square).endswith(str(num))
24
25 def find_automorphic_numbers_for():
26     print("Automorphic numbers between 1 and 1000 (using for loop):")
27     for i in range(1, 1001):
28         if is_automorphic(i):
29             print(i, end=' ')
30
31 def find_automorphic_numbers_while():
32     print("\nAutomorphic numbers between 1 and 1000 (using while loop):")
33     i = 1
34     while i <= 1000:
35         if is_automorphic(i):
36             print(i, end=' ')

```

```

ALASS6 > TEST2 > ...
16         if is_automorphic(i):
17             print(i, end=' ')
18         i += 1
19
20 find_automorphic_numbers_while()
21 def is_automorphic(num):
22     square = num ** 2
23     return str(square).endswith(str(num))
24
25 def find_automorphic_numbers_for():
26     print("Automorphic numbers between 1 and 1000 (using for loop):")
27     for i in range(1, 1001):
28         if is_automorphic(i):
29             print(i, end=' ')
30
31 def find_automorphic_numbers_while():
32     print("\nAutomorphic numbers between 1 and 1000 (using while loop):")
33     i = 1
34     while i <= 1000:
35         if is_automorphic(i):
36             print(i, end=' ')
37         i += 1
38
39 # Run both
40 find_automorphic_numbers_for()
41 find_automorphic_numbers_while()
42

```

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST2
Automorphic numbers between 1 and 1000 (using for loop):
1 5 6 25 76 376 625
Automorphic numbers between 1 and 1000 (using while loop):
1 5 6 25 76 376 625 Automorphic numbers between 1 and 1000 (using for loop):
1 5 6 25 76 376 625
Automorphic numbers between 1 and 1000 (using while loop):
1 5 6 25 76 376 625
PS C:\DAV> █
```

Prompt 3:

Write a Python function using nested if-elif-else statements that classifies user feedback (1–5 rating) into Positive, Neutral, or Negative.

Test 3:

- Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).
- Instructions:
 - Generate initial code using nested if-elif-else.
 - Analyze correctness and readability.
 - Ask AI to rewrite using dictionary-based or match-case structure.

```

ALASS6 > TEST3 > ...
1 def classify_feedback(rating):
2     if rating >= 1 and rating <= 5:
3         if rating >= 4:
4             return "Positive"
5         elif rating == 3:
6             return "Neutral"
7         else:
8             return "Negative"
9     else:
10        return "Invalid rating"
11 def classify_feedback_match(rating):
12     match rating:
13         case 5 | 4:
14             return "Positive"
15         case 3:
16             return "Neutral"
17         case 2 | 1:
18             return "Negative"
19         case _:
20             return "Invalid rating"
21 def classify_feedback_dict(rating):
22     feedback_map = {}
23     1: "Negative",
24     2: "Negative",
25     3: "Neutral",
26     4: "Positive",
27     5: "Positive"
28     }
29     return feedback_map.get(rating, "Invalid rating")
30 def classify_feedback(rating):
31     if rating >= 1 and rating <= 5:
32         if rating >= 4:
33             return "Positive"
34         elif rating == 3:
35             return "Neutral"
36         else:

```

```

ALASS6 > TEST3 > ...
30 def classify_feedback(rating):
31     if rating >= 1 and rating <= 5:
32         if rating >= 4:
33             return "Positive"
34         elif rating == 3:
35             return "Neutral"
36         else:
37             return "Negative"
38     else:
39         return "Invalid rating"
40
41 def classify_feedback_match(rating):
42     match rating:
43         case 5 | 4:
44             return "Positive"
45         case 3:
46             return "Neutral"
47         case 2 | 1:
48             return "Negative"
49         case _:
50             return "Invalid rating"
51
52 def classify_feedback_dict(rating):
53     feedback_map = {
54         1: "Negative",
55         2: "Negative",
56         3: "Neutral",
57         4: "Positive",
58         5: "Positive"
59     }
60     return feedback_map.get(rating, "Invalid rating")
61 for r in [5, 3, 1, 0, 6]:
62     print(f"Rating: {r}")
63     print(f"  if-elif:", classify_feedback(r))
64     print(f"  match-case:", classify_feedback_match(r))
65     print(f"  dictionary:", classify_feedback_dict(r))
66

```

OUTPUT:

```
PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST3
Rating: 5
  if-elif: Positive
  match-case: Positive
  dictionary: Positive
Rating: 3
  if-elif: Neutral
  match-case: Neutral
  dictionary: Neutral
Rating: 1
  if-elif: Negative
  match-case: Negative
  dictionary: Negative
Rating: 0
  if-elif: Invalid rating
  match-case: Invalid rating
  dictionary: Invalid rating
Rating: 6
  if-elif: Invalid rating
  match-case: Invalid rating
  dictionary: Invalid rating
PS C:\DAV> |
```

Prompt 4:

Write a Python function that prints all prime numbers between two user-specified numbers using a for loop.

Test 4:

- Task: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).
- Instructions:
 - Get AI-generated code to list all primes using a for loop.
 - Analyze the correctness and efficiency of the prime-checking logic.
 - Ask AI to regenerate an optimized version (e.g., using the square root method).

```

ALASS6 > TEST4 > find_primes_basic
1 def is_prime_basic(n):
2     if n <= 1:
3         return False
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     return True
8
9 def find_primes_in_range(start, end):
10    print(f"Prime numbers between {start} and {end}:")
11    for num in range(start, end + 1):
12        if is_prime_basic(num):
13            print(num, end=' ')
14    find_primes_in_range(1, 50)
15    import math
16
17 def is_prime_optimized(n):
18     if n <= 1:
19         return False
20     if n == 2:
21         return True
22     if n % 2 == 0:
23         return False
24     sqrt_n = int(math.sqrt(n)) + 1
25     for i in range(3, sqrt_n, 2): # check only odd numbers
26         if n % i == 0:
27             return False
28     return True
29
30 def find_primes_optimized(start, end):
31    print(f"\nOptimized Prime numbers between {start} and {end}:")
32    for num in range(start, end + 1):
33        if is_prime_optimized(num):
34            print(num, end=' ')
35    find_primes_optimized(1, 50)
36    import math

```

```

ALASS6 > TEST4 > find_primes_basic
34         print(num, end=' ')
35    find_primes_optimized(1, 50)
36    import math
37
38 def is_prime_basic(n):
39     if n <= 1:
40         return False
41     for i in range(2, n):
42         if n % i == 0:
43             return False
44     return True
45
46 def find_primes_basic(start, end):
47     print(f"\nPrime numbers between {start} and {end} (Basic):")
48     for num in range(start, end + 1):
49         if is_prime_basic(num):
50             print(num, end=' ')
51
52 def is_prime_optimized(n):
53     if n <= 1:
54         return False
55     if n == 2:
56         return True
57     if n % 2 == 0:
58         return False
59     sqrt_n = int(math.sqrt(n)) + 1
60     for i in range(3, sqrt_n, 2):
61         if n % i == 0:
62             return False
63     return True
64
65 def find_primes_optimized(start, end):
66     print(f"\nOptimized Prime numbers between {start} and {end}:")
67     for num in range(start, end + 1):
68         if is_prime_optimized(num):
69             print(num, end=' ')
70
71

```


OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST4
Prime numbers between 1 and 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Optimized Prime numbers between 1 and 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Prime numbers between 1 and 50 (Basic):
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Optimized Prime numbers between 1 and 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
PS C:\DAV> 
```

Prompt 5:

Create a Python class called Library with methods to add_book(), issue_book(), and display_books().

Test 5:

- Task: Use AI to build a Library class with methods to add_book(), issue_book(), and display_books().
- Instructions:
 - Generate Library class code using AI.
 - Analyze if methods handle edge cases (e.g., issuing unavailable books).
 - Ask AI to add comments and documentation.

```

AI_ASS6 > TESTS > ...
1 class Library:
2     def __init__(self):
3         # Initialize an empty dictionary to store book titles and their quantities
4         self.books = {}
5
6     def add_book(self, title, quantity=1):
7         """
8         Add a book or increase its quantity in the library.
9         """
10        if title in self.books:
11            self.books[title] += quantity
12        else:
13            self.books[title] = quantity
14        print(f"{quantity} copy/copies of '{title}' added to the library.")
15
16    def issue_book(self, title):
17        """
18        Issue a book to a user if available.
19        """
20        if title not in self.books:
21            print(f"'{title}' is not available in the library.")
22        elif self.books[title] == 0:
23            print(f"'{title}' is currently out of stock.")
24        else:
25            self.books[title] -= 1
26            print(f"'{title}' has been issued. Remaining copies: {self.books[title]}")
27
28    def display_books(self):
29        """
30        Display all books with their available quantities.
31        """
32        if not self.books:
33            print("The library has no books.")
34        else:
35            print("Books available in the library:")
36            for title, quantity in self.books.items():

```

```

AI_ASS6 > TESTS > ...
35 print("Books available in the library:")
36 for title, quantity in self.books.items():
37     print(f"+ {title} (Available: {quantity})")
38
39 class Library:
40     def __init__(self):
41         # Dictionary to hold book titles and their quantities
42         self.books = {}
43
44     def add_book(self, title, quantity=1):
45         """
46         Add a book to the library or increase the quantity if it already exists.
47
48         Parameters:
49         title (str): The title of the book to add.
50         quantity (int): Number of copies to add. Defaults to 1.
51         """
52         if title in self.books:
53             self.books[title] += quantity
54         else:
55             self.books[title] = quantity
56         print(f"{quantity} copy/copies of '{title}' added to the library.")
57
58     def issue_book(self, title):
59         """
60         Issue a book to a user if available.
61
62         Parameters:
63         title (str): The title of the book to issue.
64         """
65         if title not in self.books:
66             print(f"'{title}' is not available in the library.") # Book doesn't exist
67         elif self.books[title] == 0:
68             print(f"'{title}' is currently out of stock.") # No copies left
69         else:
70             self.books[title] -= 1 # Reduce available count
71             print(f"'{title}' has been issued. Remaining copies: {self.books[title]}")

```

```

AI_ASS6 > TEST5 > ...
38 class Library:
72     def display_books(self):
73         """
74         Display all books in the library with their available quantities.
75         """
76         if not self.books:
77             print("The library has no books.") # Empty library
78         else:
79             print("Books available in the library:")
80             for title, quantity in self.books.items():
81                 print(f"{title} (Available: {quantity})")
82     # Create library object
83     my_library = Library()
84
85     # Add books
86     my_library.add_book("Python Programming", 3)
87     my_library.add_book("Data Structures", 2)
88     my_library.add_book("Python Programming", 2) # Adding more to existing
89
90     # Display books
91     my_library.display_books()
92
93     # Issue a book
94     my_library.issue_book("Python Programming")
95
96     # Try to issue a book not in library
97     my_library.issue_book("Machine Learning")
98
99     # Try to issue a book until it's out of stock
100    my_library.issue_book("Data Structures")
101    my_library.issue_book("Data Structures")
102    my_library.issue_book("Data Structures") # Out of stock now
103
104    # Display updated book list
105    my_library.display_books()

```

OUTPUT:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST5
3 copy/copies of 'Python Programming' added to the library.
2 copy/copies of 'Data Structures' added to the library.
2 copy/copies of 'Python Programming' added to the library.
Books available in the library:
• Python Programming (Available: 5)
• Data Structures (Available: 2)
'Python Programming' has been issued. Remaining copies: 4
'Machine Learning' is not available in the library.
'Data Structures' has been issued. Remaining copies: 1
'Data Structures' has been issued. Remaining copies: 0
'Data Structures' is currently out of stock.
Books available in the library:
• Python Programming (Available: 4)
• Data Structures (Available: 0)
PS C:\DAV>

```