

WEEK-6.1

(AI -Based Code Completion: Working with suggestions for classes, loops, conditionals)

HTNO:2303A51836

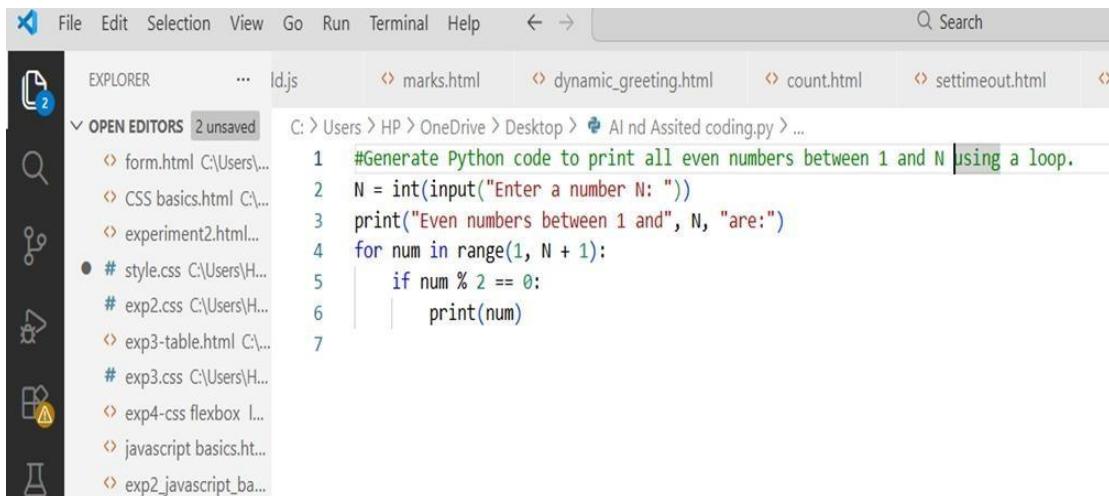
BATCH NO:27

Task Description #1: AI-Based Code Completion for Loops

Prompt:

"Generate Python code to print all even numbers between 1 and N using a loop."

AI-Generated Code



The screenshot shows a code editor interface with a sidebar containing file icons and a list of open editors. The main area displays Python code:

```
1 #Generate Python code to print all even numbers between 1 and N using a loop.
2 N = int(input("Enter a number N: "))
3 print("Even numbers between 1 and", N, "are:")
4 for num in range(1, N + 1):
5     if num % 2 == 0:
6         print(num)
7
```

Output:



The screenshot shows a terminal window with the following output:

```
PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/AI and Assisted coding.py"
Enter a number N: 9
Even numbers between 1 and 9 are:
2
4
6
8
PS C:\Users\HP>
```

Loop Type Used

- for loop

Explanation

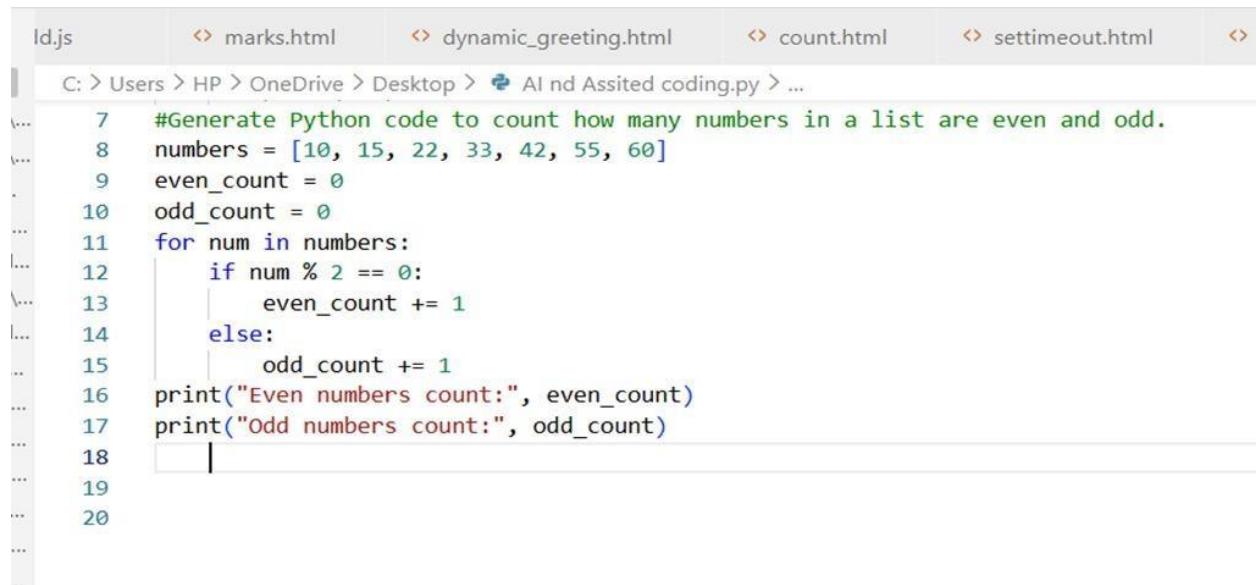
- `input()` takes the value of N
- `range(1, N + 1)` iterates from 1 to N
- `i % 2 == 0` checks if the number is even
- Even numbers are printed

Task Description #2: Loop with Conditionals

Prompt:

“Generate Python code to count how many numbers in a list are even and odd.”

AI-Generated Code



The screenshot shows a code editor interface with a tab bar at the top containing files like 'Id.js', 'marks.html', 'dynamic_greeting.html', 'count.html', and 'settimeout.html'. Below the tab bar is a code editor window displaying Python code. The code is as follows:

```
7 #Generate Python code to count how many numbers in a list are even and odd.
8 numbers = [10, 15, 22, 33, 42, 55, 60]
9 even_count = 0
10 odd_count = 0
11 for num in numbers:
12     if num % 2 == 0:
13         even_count += 1
14     else:
15         odd_count += 1
16 print("Even numbers count:", even_count)
17 print("Odd numbers count:", odd_count)
18
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/P
Even numbers count: 4
Odd numbers count: 3
PS C:\Users\HP>

Explanation:

- Initialize counters
- Loop through the list
- Check even/odd using %
- Increment respective counter
- Print results

Task Description #3: Class Attributes Validation

Prompt:

“Generate a Python class User that validates age and email using conditional statements.”

AI-Generated Code

```
C: > Users > HP > OneDrive > Desktop > AI and Assisted coding.py > ...
18  #Generate a Python class User that validates age and email using conditional statements.
19  class User:
20      def __init__(self, name, age, email):
21          self.name = name
22          self.age = age
23          self.email = email
24          self.validate_age()
25          self.validate_email()
26
27      def validate_age(self):
28          if self.age < 0:
29              raise ValueError("Age cannot be negative.")
30          elif self.age < 18:
31              print(f"{self.name} is a minor.")
32          else:
33              print(f"{self.name} is an adult.")
34
35      def validate_email(self):
36          if "@" not in self.email or "." not in self.email.split("@")[-1]:
37              raise ValueError("Invalid email address.")
38          else:
39              print(f"{self.name}'s email is valid.")
40
41 # Example usage:
42 user1 = User("Alice", 17, "alice@gmail.com")
43 user2 = User("Bob", 25, "bob@gmail.com")
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORT:

```
PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python38-32/Scripts/python.exe C:/Users/HP/Desktop/AI and Assisted coding.py
Alice is a minor.
Alice's email is valid.
Bob is an adult.
Bob's email is valid.
PS C:\Users\HP>
```

Justification

- Conditional statements validate both age and email.
- Handles valid and invalid cases effectively.
- Email validation is basic but functional.

Task Description #4: AI-Based Code Completion for Classes

Prompt:

“Generate a Python class Student with attributes (name, roll number, marks) and methods to calculate total and average marks.”

AI-Generated Code

```
C:\> Users > HP > OneDrive > Desktop > AI and Assisted coding.py > ...
43 #Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total and average marks.
44 class Student:
45     def __init__(self, name, roll_number, marks):
46         self.name = name
47         self.roll_number = roll_number
48         self.marks = marks # marks should be a list of integers
49
50     def calculate_total(self):
51         return sum(self.marks)
52
53     def calculate_average(self):
54         if len(self.marks) == 0:
55             return 0
56         return self.calculate_total() / len(self.marks)
57
58 # Example usage:
59 student1 = Student("John", 101, [85, 90, 78, 92])
60 total_marks = student1.calculate_total()
61 average_marks = student1.calculate_average()
62 print(f"Total marks for {student1.name} (Roll No: {student1.roll_number}): {total_marks}")
63 print(f"Average marks for {student1.name} (Roll No: {student1.roll_number}): {average_marks}"
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP
Total marks for John (Roll No: 101): 345
Average marks for John (Roll No: 101): 86.25
PS C:\Users\HP>
```

Explanation:

- Correct class structure
- Uses list of marks
- Methods are reusable

Task Description #5: AI-Assisted Code Completion Review

Prompt:

“Generate a Python program for a simple bank account system using class, loops, and conditional statements.”

AI-Generated code:

```
63 #Generate a Python program for a simple bank account system using class, loops, and conditional statements.
64 class BankAccount:
65     def __init__(self, account_holder, balance=0):
66         self.account_holder = account_holder
67         self.balance = balance
68
69     def deposit(self, amount):
70         if amount > 0:
71             self.balance += amount
72             print(f"Deposited: ${amount}. New balance: ${self.balance}")
73         else:
74             print("Deposit amount must be positive.")
75
76     def withdraw(self, amount):
77         if amount > 0:
78             if amount <= self.balance:
79                 self.balance -= amount
80                 print(f"Withdrew: ${amount}. New balance: ${self.balance}")
81             else:
82                 print("Insufficient funds for withdrawal.")
83         else:
84             print("Withdrawal amount must be positive.")
85
86     def display_balance(self):
87         print(f"Account holder: {self.account_holder}, Balance: ${self.balance}")
88 # Example usage:
89 account = BankAccount("Alice", 1000)
90 account.display_balance()
91 account.deposit(500)
92 account.withdraw(200)
93 account.withdraw(2000)
94 account.display_balance()
95 # Interactive loop for multiple transactions
```

```

    print(f"Account holder: {self.account_holder}, Balance: ${self.balance}")
# Example usage:
account = BankAccount("Alice", 1000)
account.display_balance()
account.deposit(500)
account.withdraw(200)
account.withdraw(2000)
account.display_balance()
# Interactive loop for multiple transactions
while True:
    action = input("Enter 'd' to deposit, 'w' to withdraw, 'b' to check balance, or 'q' to quit: ").lower()
    if action == 'd':
        amount = float(input("Enter amount to deposit: "))
        account.deposit(amount)
    elif action == 'w':
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
    elif action == 'b':
        account.display_balance()
    elif action == 'q':
        print("Exiting the banking system. Goodbye!")
        break
    else:
        print("Invalid option. Please try again.")

```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Us
 Account holder: Alice, Balance: \$1000
 Deposited: \$500. New balance: \$1500
 Withdraw: \$200. New balance: \$1300
 Insufficient funds for withdrawal.
 Account holder: Alice, Balance: \$1300
 Enter 'd' to deposit, 'w' to withdraw, 'b' to check balance, or 'q' to quit: d
 Enter amount to deposit: 1000
 Deposited: \$1000.0. New balance: \$2300.0
 Enter 'd' to deposit, 'w' to withdraw, 'b' to check balance, or 'q' to quit: w
 Enter amount to withdraw: 1300
 Withdraw: \$1300.0. New balance: \$1000.0
 Enter 'd' to deposit, 'w' to withdraw, 'b' to check balance, or 'q' to quit: b
 Account holder: Alice, Balance: \$1000.0
 Enter 'd' to deposit, 'w' to withdraw, 'b' to check balance, or 'q' to quit: q
 Exiting the banking system. Goodbye!

PS C:\Users\HP>

Explanation:

- The BankAccount class manages balance and banking operations.
- Methods are used for deposit, withdrawal, and balance display.
- A while loop creates a menu-driven system.

- Conditional statements handle user choices and validations.
- AI helped combine classes, loops, and conditionals into one program.