WEEK-6.1

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

HTNO:2303A51910

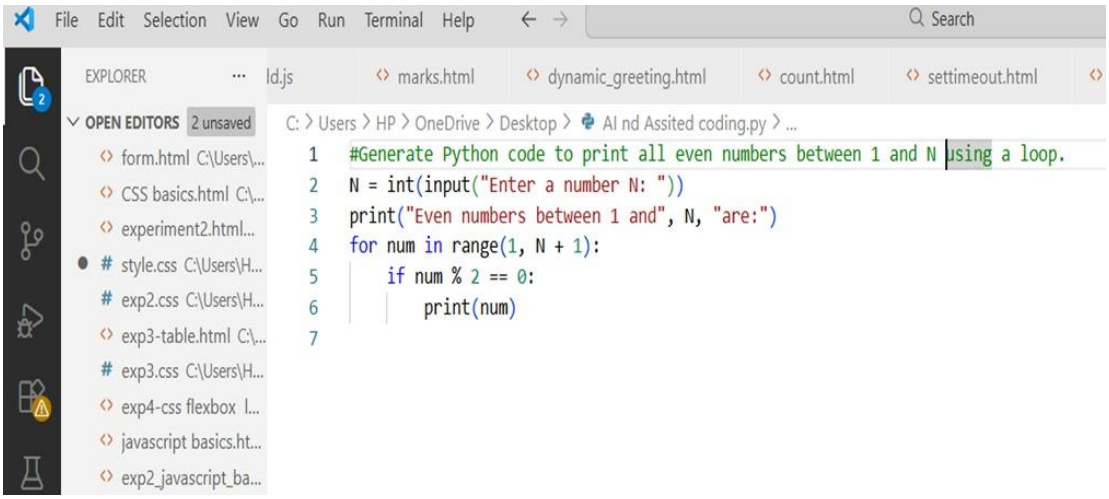
BATCH NO:28

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



Output:

Loop Type Used

- for loop

Explanation

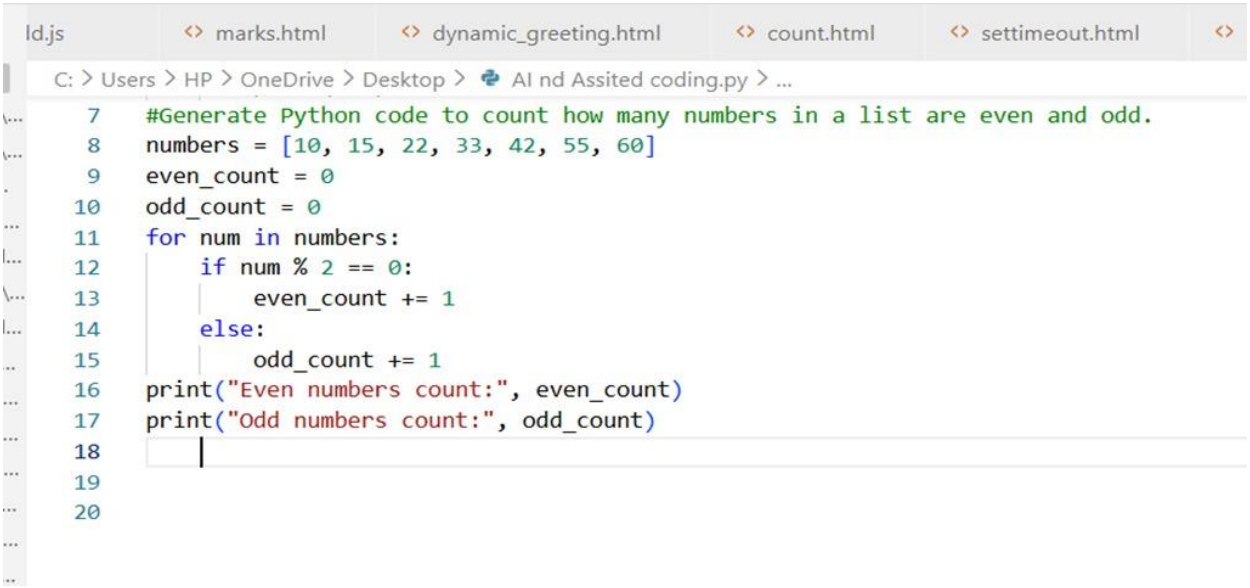
- input() takes the value of N
- range(1, N + 1) iterates from 1 to N
- i % 2 == 0 checks if the number is even
- Even numbers are printed

Task Description #2: **Loop with Conditionals**

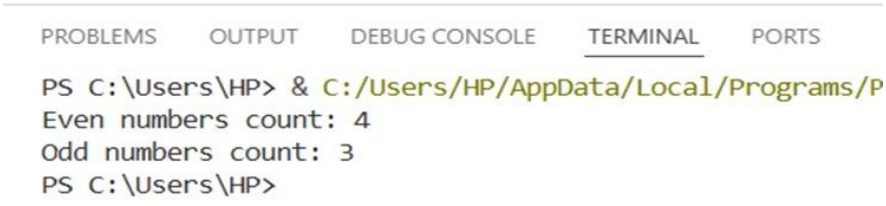
Prompt:

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

AI-Generated Code



Output:



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 nd Assited coding.py > ...

```python
#Generate a Python class User that validates age and email using conditional statements.
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email
        self.validate_age()
        self.validate_email()

    def validate_age(self):
        if self.age < 0:
            raise ValueError("Age cannot be negative.")
        elif self.age < 18:
            print(f"{self.name} is a minor.")
        else:
            print(f"{self.name} is an adult.")

    def validate_email(self):
        if "@" not in self.email or "." not in self.email.split("@")[-1]:
            raise ValueError("Invalid email address.")
        else:
            print(f"{self.name}'s email is valid.")
# Example usage:
user1 = User("Alice", 17, "alice@gmail.com")
user2 = User("Bob", 25, "bob@gmail.com")

```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORT

```
PS C:\Users\HP> & C:/Users/HP/AppData/Local/Progr
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 nd Assited coding.py > ...

```python
#Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total and average marks.
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks  # marks should be a list of integers

    def calculate_total(self):
        return sum(self.marks)

    def calculate_average(self):
        if len(self.marks) == 0:
            return 0
        return self.calculate_total() / len(self.marks)
# Example usage:
student1 = Student("John", 101, [85, 90, 78, 92])
total_marks = student1.calculate_total()
average_marks = student1.calculate_average()
print(f"Total marks for {student1.name} (Roll No: {student1.roll_number}): {total_marks}")
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:

```python
#Generate a Python program for a simple bank account system using class, loops, and conditional statements.
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount}. New balance: ${self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: ${amount}. New balance: ${self.balance}")
            else:
                print("Insufficient funds for withdrawal.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        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()
```

```
        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
Withdrew: $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
Withdrew: $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.