

AI ASSISTED CODING

NAME: K.SUPRIYA

ENROLL NUMBER: 2403A52395

BATCH NUMBER :14

### Lab assignment-6.4

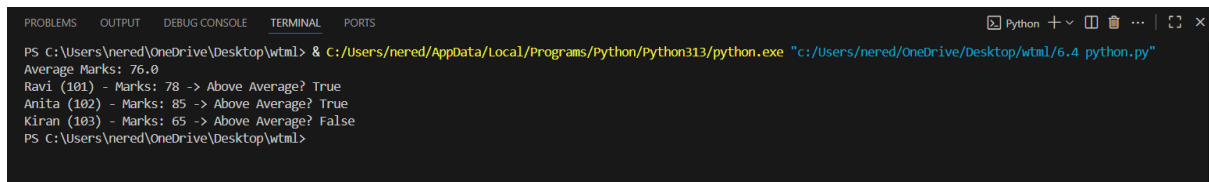
Prompt 1:

generate a Python class named Student with attributes name, roll\_number, and marks.  
Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average

code:

```
6.4 python.py > ...
1  class Student:
2      students = [] # class-level list to store all students
3
4      def __init__(self, name, roll_number, marks):
5          self.name = name
6          self.roll_number = roll_number
7          self.marks = marks
8          Student.students.append(self)
9
10     @classmethod
11     def calculate_average(cls):
12         if not cls.students:
13             return 0
14         total = sum(student.marks for student in cls.students)
15         return total / len(cls.students)
16
17     def is_above_average(self):
18         average = Student.calculate_average()
19         return self.marks > average
20
21
22 # Example usage
23 s1 = Student("Ravi", 101, 78)
24 s2 = Student("Anita", 102, 85)
25 s3 = Student("Kiran", 103, 65)
26
27 print(f"Average Marks: {Student.calculate_average()}")
28
29 for s in Student.students:
30     print(f"{s.name} ({s.roll_number}) - Marks: {s.marks} -> Above Average? {s.is_above_average()}")
31
```

Output:



```
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
Average Marks: 76.0
Ravi (101) - Marks: 78 -> Above Average? True
Anita (102) - Marks: 85 -> Above Average? True
Kiran (103) - Marks: 65 -> Above Average? False
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code defines a Student class for managing student records and checking if a student's marks are above average:

- **Class Attribute:**  
students is a class-level list that stores all student instances.
- **Constructor (\_\_init\_\_):**  
Initializes each student's name, roll\_number, and marks.  
Adds the new student to the students list.
- **Class Method (calculate\_average):**  
Calculates the average marks of all students in the list.  
Returns 0 if there are no students.
- **Instance Method (is\_above\_average):**  
Checks if the student's marks are greater than the average marks of all students.
- **Example Usage:**  
Creates three student objects.  
Prints the average marks.  
Iterates through all students, printing their details and whether their marks are above average.

Promote 2:

- generate a python code for first two lines of a for loop to iterate through a list of numbers. Use a comment prompt to let Copilot suggest how to calculate and print the square of even numbers only.

Code:

```
6.4 python.py > ...
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2 for num in numbers:
3     if num % 2 == 0:
4         print(f"The square of {num} is {num ** 2}")
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [ ] ... | [ ] [ ] X
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code iterates through a list of numbers from 1 to 10:

- `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`  
Defines a list of numbers.
- `for num in numbers:`  
Loops through each number in the list.
- `if num % 2 == 0:`  
Checks if the current number is even (divisible by 2).
- `print(f"The square of {num} is {num ** 2}")`  
If the number is even, prints its square using an f-string for formatting.

Promote 3:

- Generate a python code to Create a class called BankAccount with attributes account\_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance

Code:

6.4 python.py > ...

```
1 class BankAccount:
2     def __init__(self, account_holder, balance=0):
3         self.account_holder = account_holder
4         self.balance = balance
5
6     def deposit(self, amount):
7         """Add money to the account"""
8         if amount > 0:
9             self.balance += amount
10            print(f"Deposited {amount}. New balance: {self.balance}")
11        else:
12            print("Deposit amount must be positive.")
13
14    def withdraw(self, amount):
15        """Withdraw money, checking for insufficient balance"""
16        if amount <= 0:
17            print("Withdrawal amount must be positive.")
18        elif amount > self.balance:
19            print("Insufficient balance! Withdrawal failed.")
20        else:
21            self.balance -= amount
22            print(f"Withdrew {amount}. New balance: {self.balance}")
23
24    def get_balance(self):
25        """Check current balance"""
26        return self.balance
27
28
29 # Example usage
30 account1 = BankAccount("Ravi", 1000)
31
32 print(f"Account Holder: {account1.account_holder}")
33 print(f"Initial Balance: {account1.get_balance()}")
34
35 account1.deposit(500)
36 account1.withdraw(300)
37 account1.withdraw(1500) # insufficient balance case
38 class BankAccount:
39     def __init__(self, account_holder, balance=0):
40         self.account_holder = account_holder
41         self.balance = balance
```

6.4 python.py > ...

```
38 class BankAccount:
42
43     def deposit(self, amount):
44         """Add money to the account"""
45         if amount > 0:
46             self.balance += amount
47             print(f"Deposited {amount}. New balance: {self.balance}")
48         else:
49             print("Deposit amount must be positive.")
50
51     def withdraw(self, amount):
52         """Withdraw money, checking for insufficient balance"""
53         if amount <= 0:
54             print("Withdrawal amount must be positive.")
55         elif amount > self.balance:
56             print("Insufficient balance! Withdrawal failed.")
57         else:
58             self.balance -= amount
59             print(f"Withdrew {amount}. New balance: {self.balance}")
60
61     def get_balance(self):
62         """Check current balance"""
63         return self.balance
64
65
66 # Example usage
67 account1 = BankAccount("Ravi", 1000)
68
69 print(f"Account Holder: {account1.account_holder}")
70 print(f"Initial Balance: {account1.get_balance()}")
71
72 account1.deposit(500)
73 account1.withdraw(300)
74 account1.withdraw(1500) # insufficient balance case
75 |
```

Output:

```
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
Account Holder: Ravi
Initial Balance: 1000
Deposited 500. New balance: 1500
Withdrew 300. New balance: 1200
Insufficient balance! Withdrawal failed.
Account Holder: Ravi
Initial Balance: 1000
Deposited 500. New balance: 1500
Withdrew 300. New balance: 1200
Insufficient balance! Withdrawal failed.
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code defines a [BankAccount](#) class to simulate basic banking operations:

- **[\\_\\_init\\_\\_](#) method:**  
Initializes the account with the holder's name and an optional starting balance (default is 0).
- **[deposit](#) method:**  
Adds money to the account if the amount is positive.  
Prints the deposited amount and new balance.  
If the amount is not positive, prints an error message.
- **[withdraw](#) method:**  
Withdraws money if the amount is positive and does not exceed the current balance.  
Prints the withdrawn amount and new balance.  
If the amount is not positive or exceeds the balance, prints an error message.
- **[get\\_balance](#) method:**  
Returns the current account balance.
- **Example usage:**
  - Creates an account for "Ravi" with an initial balance of 1000.
  - Prints the account holder and initial balance.
  - Deposits 500, withdraws 300, and attempts to withdraw 1500 (which fails due to insufficient balance).
  - All actions print relevant messages to the console.

Promote 4:

- generate a python code and Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

Code:



Promote 5:

- generate a python code for Begin writing a class ShoppingCart with an empty items list.
- Prompt Copilot to generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts.

Code:

```
6.4 python.py > ShoppingCart > remove_item
1 class ShoppingCart:
2     def __init__(self):
3         # Start with an empty shopping cart
4         self.items = []
5
6     def add_item(self, name, price, quantity=1):
7         """Add an item with price and quantity"""
8         self.items.append({"name": name, "price": price, "quantity": quantity})
9         print(f"Added {quantity} x {name} at {price} each.")
10
11    def remove_item(self, name):
12        """Remove an item by name"""
13        for item in self.items:
14            if item["name"].lower() == name.lower():
15                self.items.remove(item)
16                print(f"Removed {name}.")
17        return
18    print(f"{name} not found in cart.")
19
20    def calculate_total(self):
21        """Calculate total bill with discounts"""
22        total = 0
23        for item in self.items:
24            total += item["price"] * item["quantity"]
25
26        # Apply conditional discounts
27        if total > 5000:
28            discount = 0.20 # 20% discount
29        elif total > 2000:
30            discount = 0.10 # 10% discount
31        else:
32            discount = 0.0 # no discount
33
34        discounted_total = total - (total * discount)
35        return discounted_total, discount * 100
36
37
```

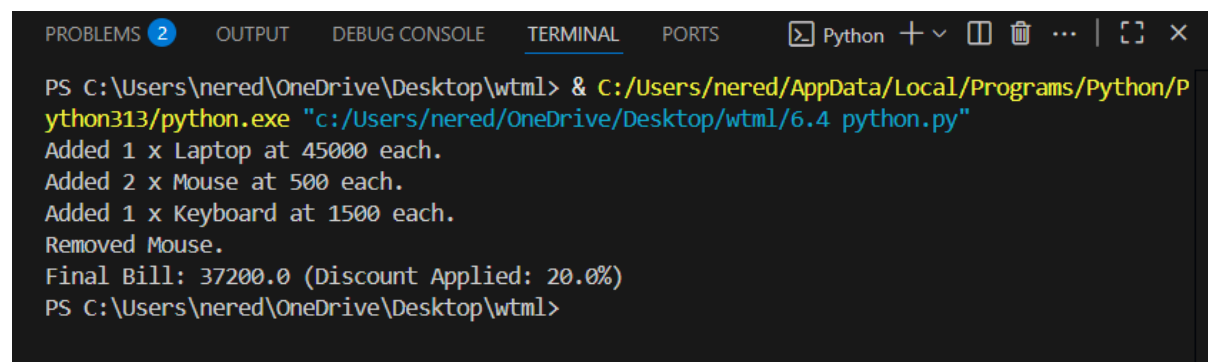


```

36
37
38 # Example usage
39 cart = ShoppingCart()
40 cart.add_item("Laptop", 45000, 1)
41 cart.add_item("Mouse", 500, 2)
42 cart.add_item("Keyboard", 1500, 1)
43
44 cart.remove_item("Mouse")
45
46 total, discount = cart.calculate_total()
47 print(f"Final Bill: {total} (Discount Applied: {discount}%)"
48

```

Output:



The screenshot shows a Windows command prompt window with the following text:

```

PS C:\Users\nered\OneDrive\Desktop\wtm1> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtm1/6.4 python.py"
Added 1 x Laptop at 45000 each.
Added 2 x Mouse at 500 each.
Added 1 x Keyboard at 1500 each.
Removed Mouse.
Final Bill: 37200.0 (Discount Applied: 20.0%)
PS C:\Users\nered\OneDrive\Desktop\wtm1>

```

Code explanation:

This code iterates through a list of numbers and prints the square of each even number.

`numbers = [1, 2, 3, 4, 5, 6]` creates a list of numbers.

The for loop goes through each number in the list.

Inside the loop, `if number % 2 == 0:` checks if the number is even.

If the number is even, `print(f"Square of {number} is {number** 12}")` prints the result of raising the number to the 12th power (not the square).

Note:

To print the square, use `number ** 2` instead of `number ** 12`.

The current code prints the 12th power, not the square.

GPT

```
6.4 python.py > ...
1 class Student:
2     students = [] # class-level list to store all students
3
4     def __init__(self, name, roll_number, marks):
5         self.name = name
6         self.roll_number = roll_number
7         self.marks = marks
8         Student.students.append(self)
9
10    @classmethod
11    def calculate_average(cls):
12        if not cls.students:
13            return 0
14        total = sum(student.marks for student in cls.students)
15        return total / len(cls.students)
16
17    def is_above_average(self):
18        average = Student.calculate_average()
19        return self.marks > average
20
21
22 # Example usage
23 s1 = Student("Ravi", 101, 78)
24 s2 = Student("Anita", 102, 85)
25 s3 = Student("Kiran", 103, 65)
26
27 print(f"Average Marks: {Student.calculate_average()}")
28
29 for s in Student.students:
30     print(f"{s.name} ({s.roll_number}) - Marks: {s.marks} -> Above Average? {s.is_above_average()}")
31
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
Average Marks: 76.0
Ravi (101) - Marks: 78 -> Above Average? True
Anita (102) - Marks: 85 -> Above Average? True
Kiran (103) - Marks: 65 -> Above Average? False
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code defines a Student class for managing student records and checking if a student's marks are above average:

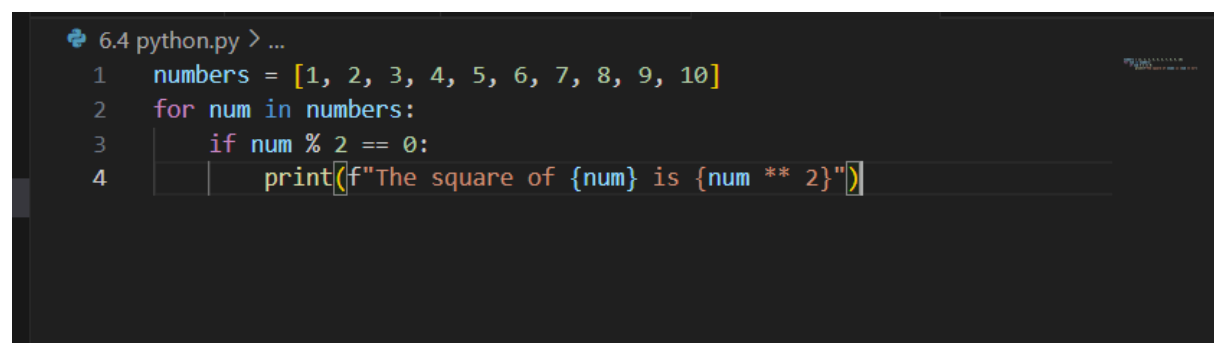
- **Class Attribute:**  
students is a class-level list that stores all student instances.

- **Constructor (\_\_init\_\_):**  
Initializes each student's name, roll\_number, and marks.  
Adds the new student to the students list.
- **Class Method (calculate\_average):**  
Calculates the average marks of all students in the list.  
Returns 0 if there are no students.
- **Instance Method (is\_above\_average):**  
Checks if the student's marks are greater than the average marks of all students.
- **Example Usage:**  
Creates three student objects.  
Prints the average marks.  
Iterates through all students, printing their details and whether their marks are above average.

Promote 2:

- generate a python code for first two lines of a for loop to iterate through a list of numbers. Use a comment prompt to let Copilot suggest how to calculate and print the square of even numbers only.

Code:



```

6.4 python.py > ...
1  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2  for num in numbers:
3      if num % 2 == 0:
4          print(f"The square of {num} is {num ** 2}")

```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [ ] ... | [ ] [ ] X
PS C:\Users\nered\OneDrive\Desktop\wtm1> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtm1/6.4 python.py"
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
PS C:\Users\nered\OneDrive\Desktop\wtm1>
```

Code explanation:

This code iterates through a list of numbers from 1 to 10:

- `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`  
Defines a list of numbers.
- `for num in numbers:`  
Loops through each number in the list.
- `if num % 2 == 0:`  
Checks if the current number is even (divisible by 2).
- `print(f"The square of {num} is {num ** 2}")`  
If the number is even, prints its square using an f-string for formatting.

Promote 3:

- Generate a python code to Create a class called BankAccount with attributes account\_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance

Code:

6.4 python.py > ...

```
1 class BankAccount:
2     def __init__(self, account_holder, balance=0):
3         self.account_holder = account_holder
4         self.balance = balance
5
6     def deposit(self, amount):
7         """Add money to the account"""
8         if amount > 0:
9             self.balance += amount
10            print(f"Deposited {amount}. New balance: {self.balance}")
11        else:
12            print("Deposit amount must be positive.")
13
14    def withdraw(self, amount):
15        """Withdraw money, checking for insufficient balance"""
16        if amount <= 0:
17            print("Withdrawal amount must be positive.")
18        elif amount > self.balance:
19            print("Insufficient balance! Withdrawal failed.")
20        else:
21            self.balance -= amount
22            print(f"Withdrew {amount}. New balance: {self.balance}")
23
24    def get_balance(self):
25        """Check current balance"""
26        return self.balance
27
28
29 # Example usage
30 account1 = BankAccount("Ravi", 1000)
31
32 print(f"Account Holder: {account1.account_holder}")
33 print(f"Initial Balance: {account1.get_balance()}")
34
35 account1.deposit(500)
36 account1.withdraw(300)
37 account1.withdraw(1500) # insufficient balance case
38 class BankAccount:
39     def __init__(self, account_holder, balance=0):
40         self.account_holder = account_holder
41         self.balance = balance
```

6.4 python.py > ...

```
38 class BankAccount:
42
43     def deposit(self, amount):
44         """Add money to the account"""
45         if amount > 0:
46             self.balance += amount
47             print(f"Deposited {amount}. New balance: {self.balance}")
48         else:
49             print("Deposit amount must be positive.")
50
51     def withdraw(self, amount):
52         """Withdraw money, checking for insufficient balance"""
53         if amount <= 0:
54             print("Withdrawal amount must be positive.")
55         elif amount > self.balance:
56             print("Insufficient balance! Withdrawal failed.")
57         else:
58             self.balance -= amount
59             print(f"Withdrew {amount}. New balance: {self.balance}")
60
61     def get_balance(self):
62         """Check current balance"""
63         return self.balance
64
65
66 # Example usage
67 account1 = BankAccount("Ravi", 1000)
68
69 print(f"Account Holder: {account1.account_holder}")
70 print(f"Initial Balance: {account1.get_balance()}")
71
72 account1.deposit(500)
73 account1.withdraw(300)
74 account1.withdraw(1500) # insufficient balance case
75 |
```

Output:

```
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
Account Holder: Ravi
Initial Balance: 1000
Deposited 500. New balance: 1500
Withdrew 300. New balance: 1200
Insufficient balance! Withdrawal failed.
Account Holder: Ravi
Initial Balance: 1000
Deposited 500. New balance: 1500
Withdrew 300. New balance: 1200
Insufficient balance! Withdrawal failed.
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code defines a [BankAccount](#) class to simulate basic banking operations:

- **[\\_\\_init\\_\\_](#) method:**  
Initializes the account with the holder's name and an optional starting balance (default is 0).
- **[deposit](#) method:**  
Adds money to the account if the amount is positive.  
Prints the deposited amount and new balance.  
If the amount is not positive, prints an error message.
- **[withdraw](#) method:**  
Withdraws money if the amount is positive and does not exceed the current balance.  
Prints the withdrawn amount and new balance.  
If the amount is not positive or exceeds the balance, prints an error message.
- **[get\\_balance](#) method:**  
Returns the current account balance.
- **Example usage:**
  - Creates an account for "Ravi" with an initial balance of 1000.
  - Prints the account holder and initial balance.
  - Deposits 500, withdraws 300, and attempts to withdraw 1500 (which fails due to insufficient balance).
  - All actions print relevant messages to the console.

Promote 4:

- generate a python code and Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

Code:





Promote 5:

- generate a python code for Begin writing a class ShoppingCart with an empty items list.
- Prompt Copilot to generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts.

Code:

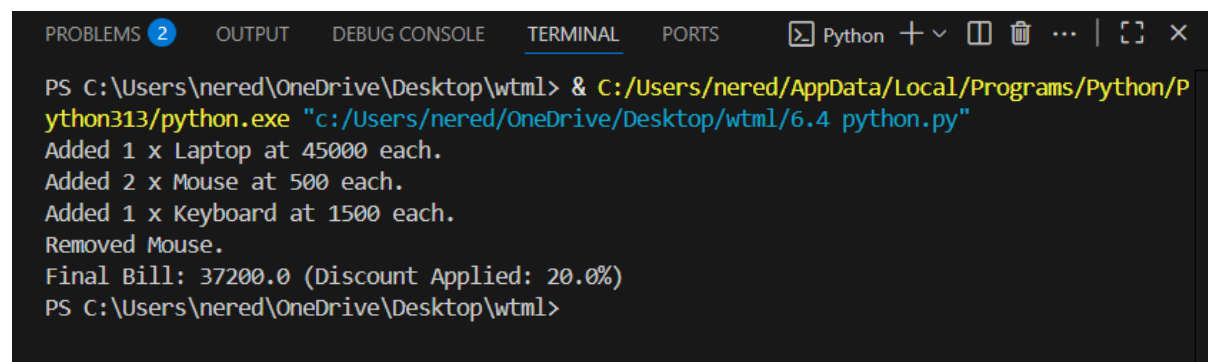
```
6.4 python.py > ShoppingCart > remove_item
1 class ShoppingCart:
2     def __init__(self):
3         # Start with an empty shopping cart
4         self.items = []
5
6     def add_item(self, name, price, quantity=1):
7         """Add an item with price and quantity"""
8         self.items.append({"name": name, "price": price, "quantity": quantity})
9         print(f"Added {quantity} x {name} at {price} each.")
10
11    def remove_item(self, name):
12        """Remove an item by name"""
13        for item in self.items:
14            if item["name"].lower() == name.lower():
15                self.items.remove(item)
16                print(f"Removed {name}.")
17        return
18    print(f"{name} not found in cart.")
19
20    def calculate_total(self):
21        """Calculate total bill with discounts"""
22        total = 0
23        for item in self.items:
24            total += item["price"] * item["quantity"]
25
26        # Apply conditional discounts
27        if total > 5000:
28            discount = 0.20 # 20% discount
29        elif total > 2000:
30            discount = 0.10 # 10% discount
31        else:
32            discount = 0.0 # no discount
33
34        discounted_total = total - (total * discount)
35        return discounted_total, discount * 100
36
37
```

```

36
37
38 # Example usage
39 cart = ShoppingCart()
40 cart.add_item("Laptop", 45000, 1)
41 cart.add_item("Mouse", 500, 2)
42 cart.add_item("Keyboard", 1500, 1)
43
44 cart.remove_item("Mouse")
45
46 total, discount = cart.calculate_total()
47 print(f"Final Bill: {total} (Discount Applied: {discount}%)"
48

```

Output:



The screenshot shows a Windows command prompt window with the following text:

```

PS C:\Users\nered\OneDrive\Desktop\wtm1> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtm1/6.4 python.py"
Added 1 x Laptop at 45000 each.
Added 2 x Mouse at 500 each.
Added 1 x Keyboard at 1500 each.
Removed Mouse.
Final Bill: 37200.0 (Discount Applied: 20.0%)
PS C:\Users\nered\OneDrive\Desktop\wtm1>

```

Code explanation:

This code iterates through a list of numbers and prints the square of each even number.

`numbers = [1, 2, 3, 4, 5, 6]` creates a list of numbers.

The for loop goes through each number in the list.

Inside the loop, `if number % 2 == 0:` checks if the number is even.

If the number is even, `print(f"Square of {number} is {number** 12}")` prints the result of raising the number to the 12th power (not the square).

Note:

To print the square, use `number ** 2` instead of `number ** 12`.

The current code prints the 12th power, not the square.

GPT

```
6.4 python.py > ...
1 class Student:
2     students = [] # class-level list to store all students
3
4     def __init__(self, name, roll_number, marks):
5         self.name = name
6         self.roll_number = roll_number
7         self.marks = marks
8         Student.students.append(self)
9
10    @classmethod
11    def calculate_average(cls):
12        if not cls.students:
13            return 0
14        total = sum(student.marks for student in cls.students)
15        return total / len(cls.students)
16
17    def is_above_average(self):
18        average = Student.calculate_average()
19        return self.marks > average
20
21
22 # Example usage
23 s1 = Student("Ravi", 101, 78)
24 s2 = Student("Anita", 102, 85)
25 s3 = Student("Kiran", 103, 65)
26
27 print(f"Average Marks: {Student.calculate_average()}")
28
29 for s in Student.students:
30     print(f"{s.name} ({s.roll_number}) - Marks: {s.marks} -> Above Average? {s.is_above_average()}")
31
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - [ ] ... [ ] x
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
Average Marks: 76.0
Ravi (101) - Marks: 78 -> Above Average? True
Anita (102) - Marks: 85 -> Above Average? True
Kiran (103) - Marks: 65 -> Above Average? False
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code defines a Student class for managing student records and checking if a student's marks are above average:

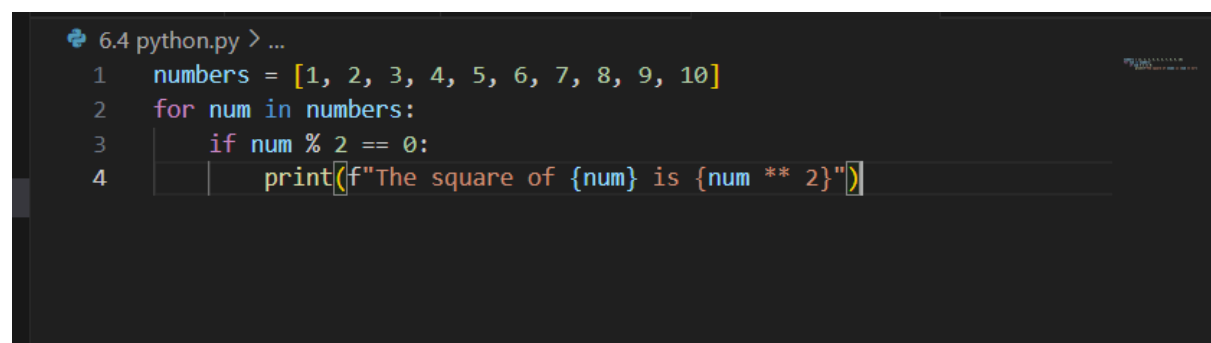
- **Class Attribute:**  
students is a class-level list that stores all student instances.

- **Constructor (\_\_init\_\_):**  
Initializes each student's name, roll\_number, and marks.  
Adds the new student to the students list.
- **Class Method (calculate\_average):**  
Calculates the average marks of all students in the list.  
Returns 0 if there are no students.
- **Instance Method (is\_above\_average):**  
Checks if the student's marks are greater than the average marks of all students.
- **Example Usage:**  
Creates three student objects.  
Prints the average marks.  
Iterates through all students, printing their details and whether their marks are above average.

Promote 2:

- generate a python code for first two lines of a for loop to iterate through a list of numbers. Use a comment prompt to let Copilot suggest how to calculate and print the square of even numbers only.

Code:

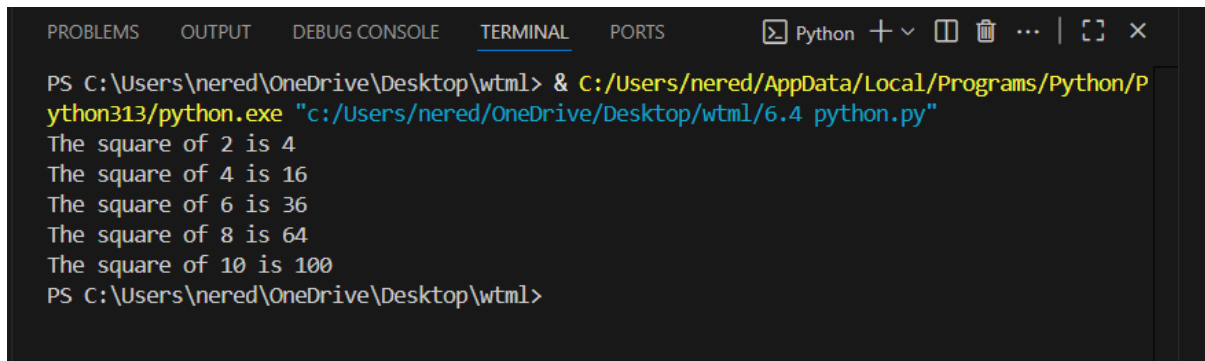


```

6.4 python.py > ...
1  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2  for num in numbers:
3      if num % 2 == 0:
4          print(f"The square of {num} is {num ** 2}")

```

Output:



```
PS C:\Users\nered\OneDrive\Desktop\wtm1> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtm1/6.4 python.py"
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
PS C:\Users\nered\OneDrive\Desktop\wtm1>
```

Code explanation:

This code iterates through a list of numbers from 1 to 10:

- `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`  
Defines a list of numbers.
- `for num in numbers:`  
Loops through each number in the list.
- `if num % 2 == 0:`  
Checks if the current number is even (divisible by 2).
- `print(f"The square of {num} is {num ** 2}")`  
If the number is even, prints its square using an f-string for formatting.

Promote 3:

- Generate a python code to Create a class called BankAccount with attributes account\_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance

Code:

6.4 python.py > ...

```
1 class BankAccount:
2     def __init__(self, account_holder, balance=0):
3         self.account_holder = account_holder
4         self.balance = balance
5
6     def deposit(self, amount):
7         """Add money to the account"""
8         if amount > 0:
9             self.balance += amount
10            print(f"Deposited {amount}. New balance: {self.balance}")
11        else:
12            print("Deposit amount must be positive.")
13
14    def withdraw(self, amount):
15        """Withdraw money, checking for insufficient balance"""
16        if amount <= 0:
17            print("Withdrawal amount must be positive.")
18        elif amount > self.balance:
19            print("Insufficient balance! Withdrawal failed.")
20        else:
21            self.balance -= amount
22            print(f"Withdrew {amount}. New balance: {self.balance}")
23
24    def get_balance(self):
25        """Check current balance"""
26        return self.balance
27
28
29 # Example usage
30 account1 = BankAccount("Ravi", 1000)
31
32 print(f"Account Holder: {account1.account_holder}")
33 print(f"Initial Balance: {account1.get_balance()}")
34
35 account1.deposit(500)
36 account1.withdraw(300)
37 account1.withdraw(1500) # insufficient balance case
38 class BankAccount:
39     def __init__(self, account_holder, balance=0):
40         self.account_holder = account_holder
41         self.balance = balance
```

6.4 python.py > ...

```
38 class BankAccount:
42
43     def deposit(self, amount):
44         """Add money to the account"""
45         if amount > 0:
46             self.balance += amount
47             print(f"Deposited {amount}. New balance: {self.balance}")
48         else:
49             print("Deposit amount must be positive.")
50
51     def withdraw(self, amount):
52         """Withdraw money, checking for insufficient balance"""
53         if amount <= 0:
54             print("Withdrawal amount must be positive.")
55         elif amount > self.balance:
56             print("Insufficient balance! Withdrawal failed.")
57         else:
58             self.balance -= amount
59             print(f"Withdrew {amount}. New balance: {self.balance}")
60
61     def get_balance(self):
62         """Check current balance"""
63         return self.balance
64
65
66 # Example usage
67 account1 = BankAccount("Ravi", 1000)
68
69 print(f"Account Holder: {account1.account_holder}")
70 print(f"Initial Balance: {account1.get_balance()}")
71
72 account1.deposit(500)
73 account1.withdraw(300)
74 account1.withdraw(1500) # insufficient balance case
75 |
```

Output:

```
PS C:\Users\nered\OneDrive\Desktop\wtml> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtml/6.4 python.py"
Account Holder: Ravi
Initial Balance: 1000
Deposited 500. New balance: 1500
Withdrew 300. New balance: 1200
Insufficient balance! Withdrawal failed.
Account Holder: Ravi
Initial Balance: 1000
Deposited 500. New balance: 1500
Withdrew 300. New balance: 1200
Insufficient balance! Withdrawal failed.
PS C:\Users\nered\OneDrive\Desktop\wtml>
```

Code explanation:

This code defines a [BankAccount](#) class to simulate basic banking operations:

- **[\\_\\_init\\_\\_](#) method:**  
Initializes the account with the holder's name and an optional starting balance (default is 0).
- **[deposit](#) method:**  
Adds money to the account if the amount is positive.  
Prints the deposited amount and new balance.  
If the amount is not positive, prints an error message.
- **[withdraw](#) method:**  
Withdraws money if the amount is positive and does not exceed the current balance.  
Prints the withdrawn amount and new balance.  
If the amount is not positive or exceeds the balance, prints an error message.
- **[get\\_balance](#) method:**  
Returns the current account balance.
- **Example usage:**
  - Creates an account for "Ravi" with an initial balance of 1000.
  - Prints the account holder and initial balance.
  - Deposits 500, withdraws 300, and attempts to withdraw 1500 (which fails due to insufficient balance).
  - All actions print relevant messages to the console.

Promote 4:

- generate a python code and Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

Code:





Promote 5:

- generate a python code for Begin writing a class ShoppingCart with an empty items list.
- Prompt Copilot to generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts.

Code:

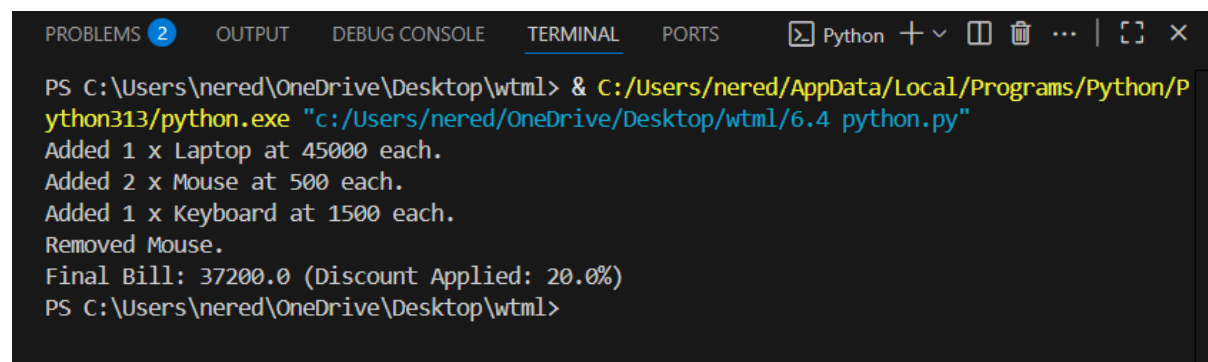
```
6.4 python.py > ShoppingCart > remove_item
1 class ShoppingCart:
2     def __init__(self):
3         # Start with an empty shopping cart
4         self.items = []
5
6     def add_item(self, name, price, quantity=1):
7         """Add an item with price and quantity"""
8         self.items.append({"name": name, "price": price, "quantity": quantity})
9         print(f"Added {quantity} x {name} at {price} each.")
10
11    def remove_item(self, name):
12        """Remove an item by name"""
13        for item in self.items:
14            if item["name"].lower() == name.lower():
15                self.items.remove(item)
16                print(f"Removed {name}.")
17        return
18    print(f"{name} not found in cart.")
19
20    def calculate_total(self):
21        """Calculate total bill with discounts"""
22        total = 0
23        for item in self.items:
24            total += item["price"] * item["quantity"]
25
26        # Apply conditional discounts
27        if total > 5000:
28            discount = 0.20 # 20% discount
29        elif total > 2000:
30            discount = 0.10 # 10% discount
31        else:
32            discount = 0.0 # no discount
33
34        discounted_total = total - (total * discount)
35        return discounted_total, discount * 100
36
37
```

```

36
37
38 # Example usage
39 cart = ShoppingCart()
40 cart.add_item("Laptop", 45000, 1)
41 cart.add_item("Mouse", 500, 2)
42 cart.add_item("Keyboard", 1500, 1)
43
44 cart.remove_item("Mouse")
45
46 total, discount = cart.calculate_total()
47 print(f"Final Bill: {total} (Discount Applied: {discount}%)"
48

```

Output:



The screenshot shows a Windows Command Prompt window with the following text:

```

PS C:\Users\nered\OneDrive\Desktop\wtm1> & C:/Users/nered/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nered/OneDrive/Desktop/wtm1/6.4 python.py"
Added 1 x Laptop at 45000 each.
Added 2 x Mouse at 500 each.
Added 1 x Keyboard at 1500 each.
Removed Mouse.
Final Bill: 37200.0 (Discount Applied: 20.0%)
PS C:\Users\nered\OneDrive\Desktop\wtm1>

```

Code explanation:

This code iterates through a list of numbers and prints the square of each even number.

`numbers = [1, 2, 3, 4, 5, 6]` creates a list of numbers.

The for loop goes through each number in the list.

Inside the loop, `if number % 2 == 0:` checks if the number is even.

If the number is even, `print(f"Square of {number} is {number** 12}")` prints the result of raising the number to the 12th power (not the square).

Note:

To print the square, use `number ** 2` instead of `number ** 12`.

The current code prints the 12th power, not the square.

GPT