AI Assisted Coding

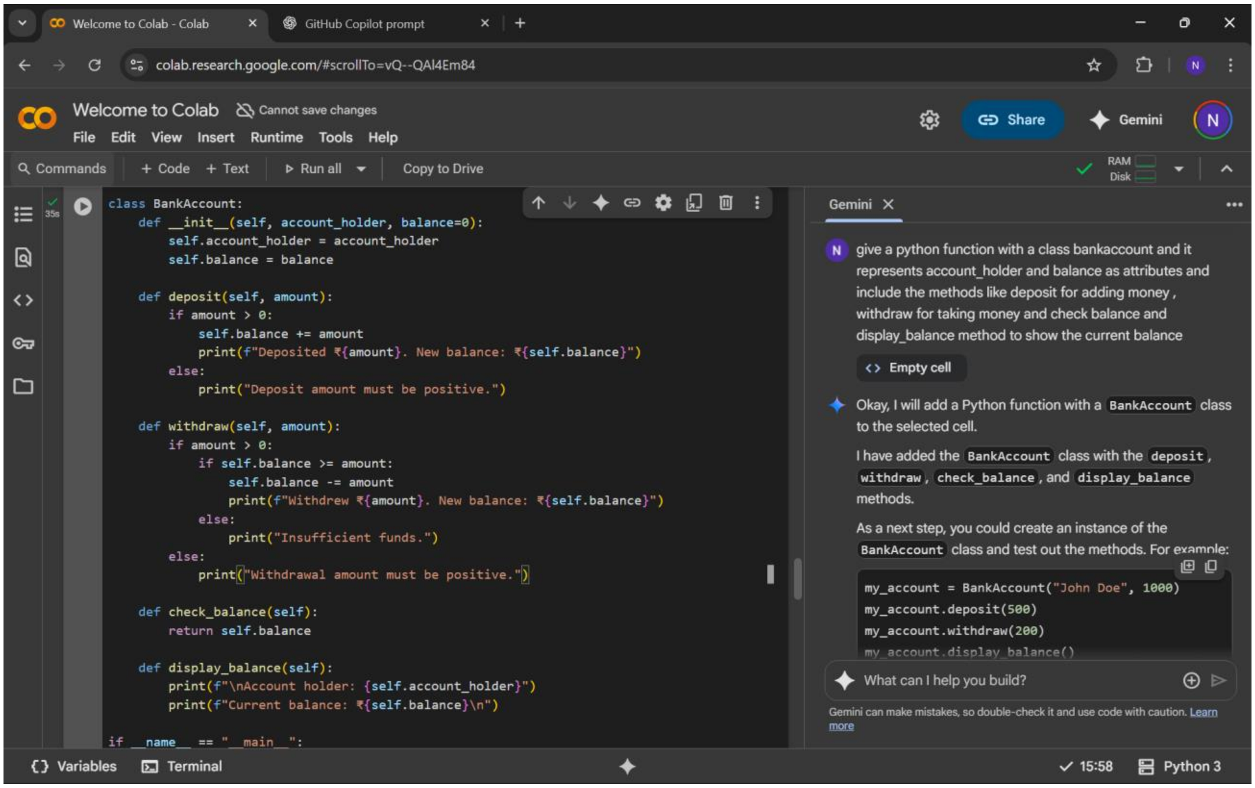
NAME: B.Tejasri

HTNO:2403a51292

BATCH:12

TASK1: Auto-Complete a Python Class for Bank Account

- Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance.



```
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 self.balance >= amount:
                self.balance -= amount
                print(f"Withdrew ₹{amount}. New balance: ₹{self.balance}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        return self.balance

    def display_balance(self):
        print(f"\nAccount holder: {self.account_holder}")
        print(f"Current balance: ₹{self.balance}\n")

if __name__ == "__main__":
```

Gemini

give a python function with a class bankaccount and it represents account_holder and balance as attributes and include the methods like deposit for adding money , withdraw for taking money and check balance and display_balance method to show the current balance

Empty cell

Okay, I will add a Python function with a `BankAccount` class to the selected cell.

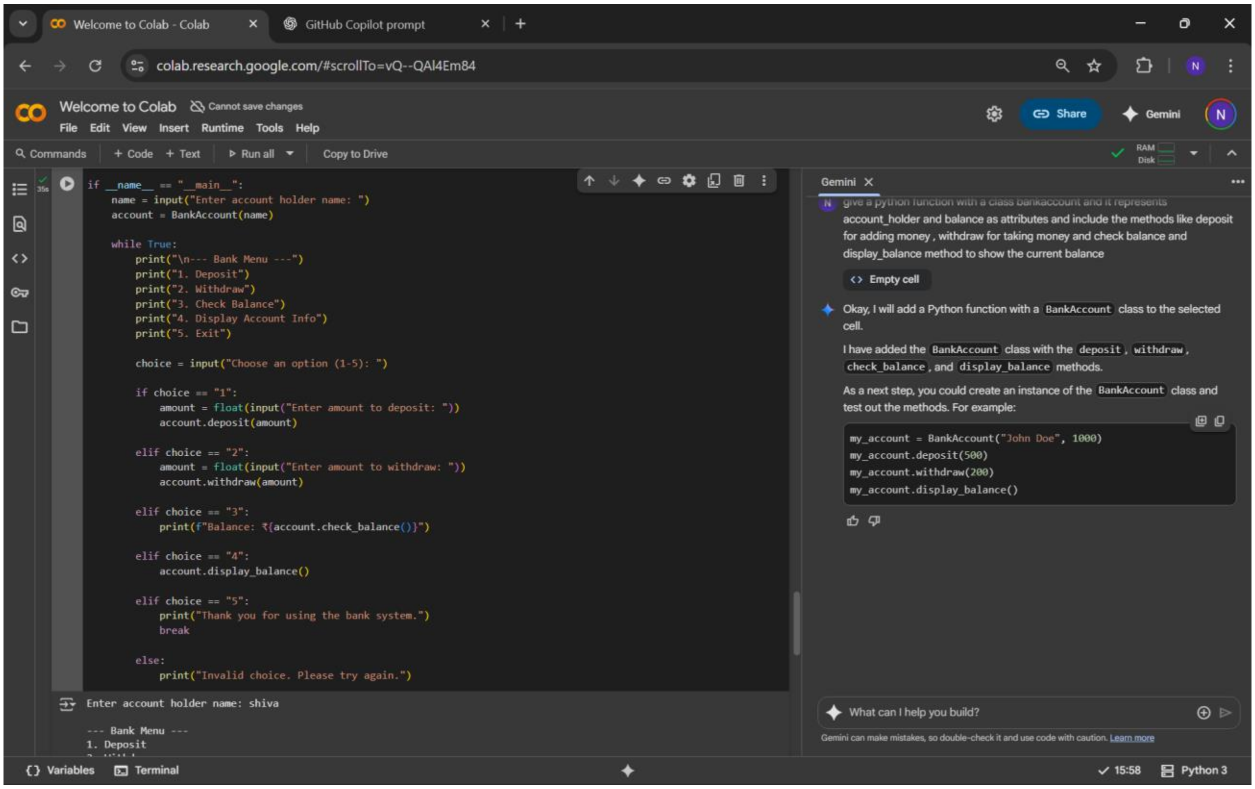
I have added the `BankAccount` class with the `deposit`, `withdraw`, `check_balance`, and `display_balance` methods.

As a next step, you could create an instance of the `BankAccount` class and test out the methods. For example:

```
my_account = BankAccount("John Doe", 1000)
my_account.deposit(500)
my_account.withdraw(200)
my_account.display_balance()
```

What can I help you build?

Gemini can make mistakes, so double-check it and use code with caution. Learn more



```
if __name__ == "__main__":
    name = input("Enter account holder name: ")
    account = BankAccount(name)

    while True:
        print("\n--- Bank Menu ---")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Display Account Info")
        print("5. Exit")

        choice = input("Choose an option (1-5): ")

        if choice == "1":
            amount = float(input("Enter amount to deposit: "))
            account.deposit(amount)

        elif choice == "2":
            amount = float(input("Enter amount to withdraw: "))
            account.withdraw(amount)

        elif choice == "3":
            print(f"Balance: ₹{account.check_balance()}")

        elif choice == "4":
            account.display_balance()

        elif choice == "5":
            print("Thank you for using the bank system.")
            break

        else:
            print("Invalid choice. Please try again.")
```

```
Enter account holder name: shiva

--- Bank Menu ---
1. Deposit
```

give a python function with a class bankaccount and it represents account_holder and balance as attributes and include the methods like deposit for adding money , withdraw for taking money and check balance and display_balance method to show the current balance

<> Empty cell

Okay, I will add a Python function with a `BankAccount` class to the selected cell.

I have added the `BankAccount` class with the `deposit`, `withdraw`, `check_balance`, and `display_balance` methods.

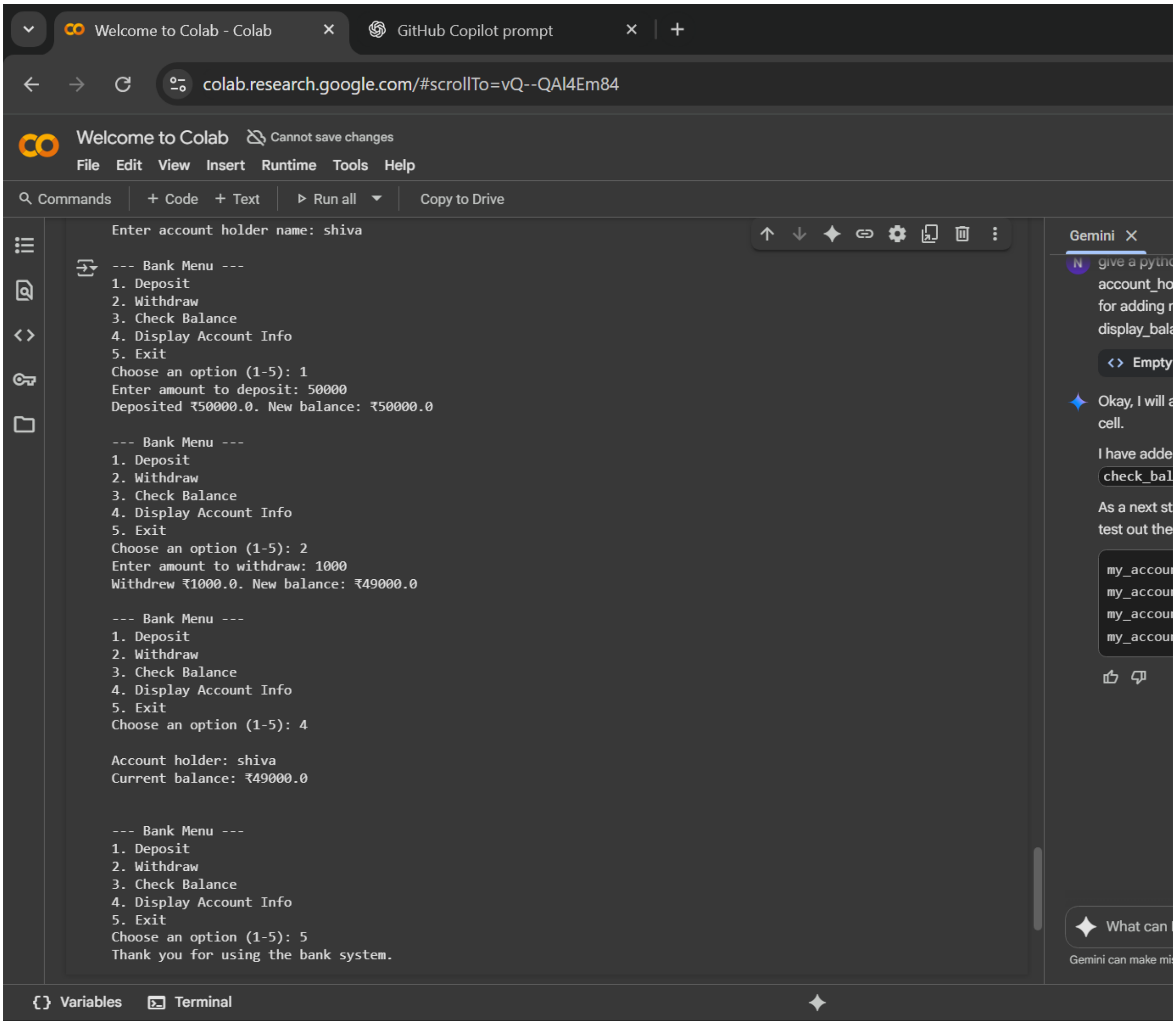
As a next step, you could create an instance of the `BankAccount` class and test out the methods. For example:

```
my_account = BankAccount("John Doe", 1000)
my_account.deposit(500)
my_account.withdraw(200)
my_account.display_balance()
```

What can I help you build?

Gemini can make mistakes, so double-check it and use code with caution. Learn more

Output:



```
Enter account holder name: shiva

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Display Account Info
5. Exit
Choose an option (1-5): 1
Enter amount to deposit: 50000
Deposited ₹50000.0. New balance: ₹50000.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Display Account Info
5. Exit
Choose an option (1-5): 2
Enter amount to withdraw: 1000
Withdrew ₹1000.0. New balance: ₹49000.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Display Account Info
5. Exit
Choose an option (1-5): 4

Account holder: shiva
Current balance: ₹49000.0


--- Bank Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Display Account Info
5. Exit
Choose an option (1-5): 5
Thank you for using the bank system.
```

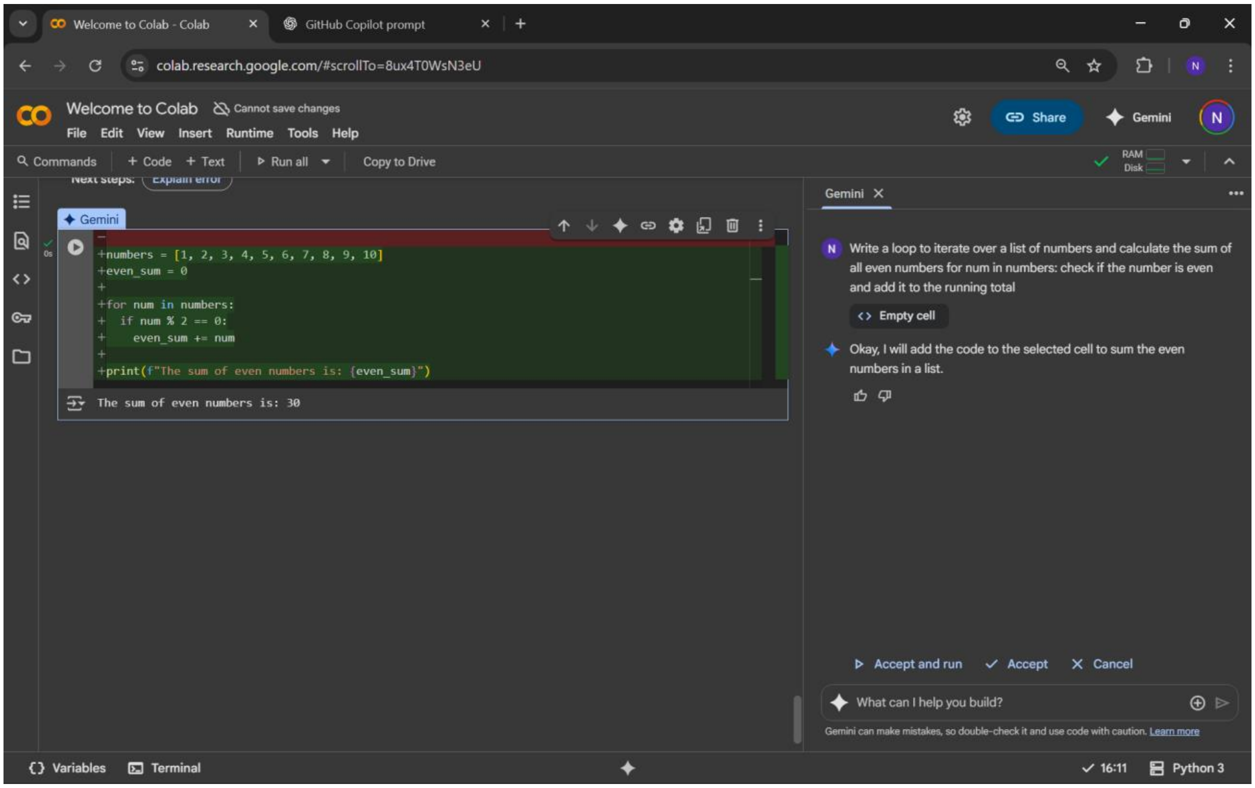
Task 2: Auto-Complete a For Loop to Sum Even Numbers in a List

- Write a comment and the initial line of a loop to iterate over a list. Allow GitHub Copilot to
  complete the logic to sum all even numbers in the list

Expected Output #2

- Code that:
- Iterates over a list
- Checks if the number is even using % 2 == 0

- Accumulates the sum
- Sample input/output:



```
-
+numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
+even_sum = 0
+
+for num in numbers:
+  if num % 2 == 0:
+    even_sum += num
+
+print(f"The sum of even numbers is: {even_sum}")
```

```
The sum of even numbers is: 30
```

Write a loop to iterate over a list of numbers and calculate the sum of all even numbers for num in numbers: check if the number is even and add it to the running total

Okay, I will add the code to the selected cell to sum the even numbers in a list.

Accept and run
Accept
Cancel
What can I help you build?
Gemini can make mistakes, so double-check it and use code with caution. Learn more

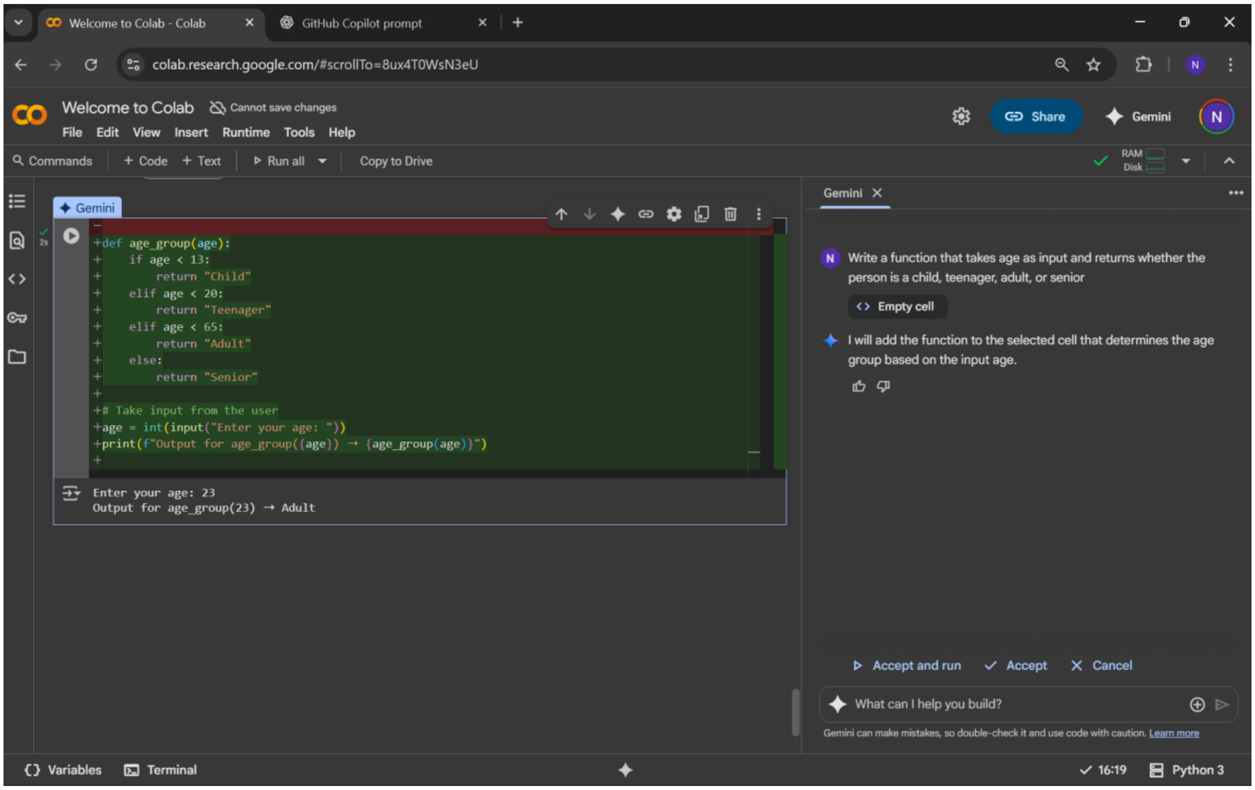
Task 3: Auto-Complete Conditional Logic to Check Age Group

- Start a function that takes age as input and returns whether the person is a child, teenager,
  adult, or senior using if-elif-else. Use Copilot to complete the conditionals.

Expected Output #3

- Function like:

Output for age_group(45) → "Adult”



```
-
+def age_group(age):
+    if age < 13:
+        return "Child"
+    elif age < 20:
+        return "Teenager"
+    elif age < 65:
+        return "Adult"
+    else:
+        return "Senior"
+
+# Take input from the user
+age = int(input("Enter your age: "))
+print(f"Output for age_group({age}) → {age_group(age)}")
+
```

```
Enter your age: 23
Output for age_group(23) → Adult
```

Write a function that takes age as input and returns whether the person is a child, teenager, adult, or senior

I will add the function to the selected cell that determines the age group based on the input age.

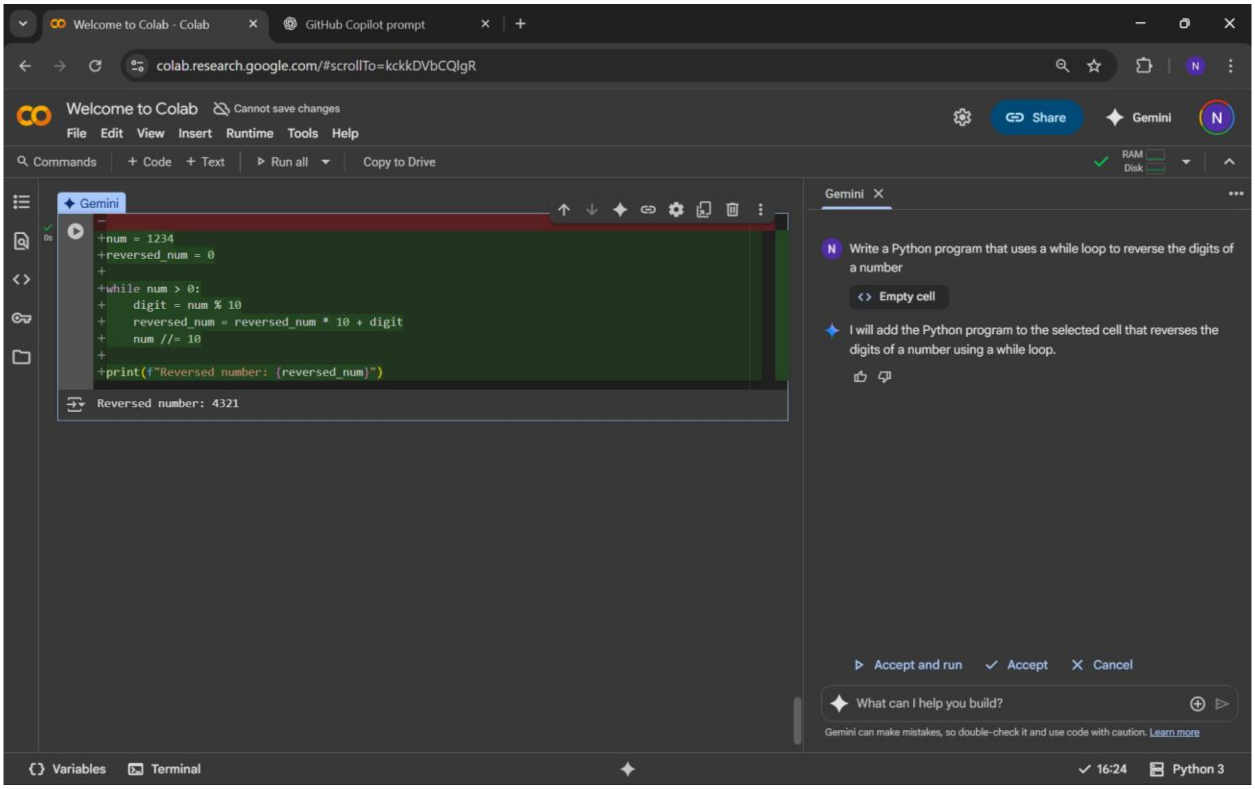
Gemini can make mistakes, so double-check it and use code with caution. Learn more

Task 4: Auto-Complete a While Loop to Reverse Digits of a Number

- **Write a comment and start a while loop to reverse the digits**
- **Write a comment and start a while loop to reverse the digits of a number. Let Copilot complete the loop logic.**

Expected Output #4

- **Functional loop: Output: 4321**



```
-
+num = 1234
+reversed_num = 0
+
+while num > 0:
+    digit = num % 10
+    reversed_num = reversed_num * 10 + digit
+    num //= 10
+
+print(f"Reversed number: {reversed_num}")
```

```
Reversed number: 4321
```

Write a Python program that uses a while loop to reverse the digits of a number

I will add the Python program to the selected cell that reverses the digits of a number using a while loop.

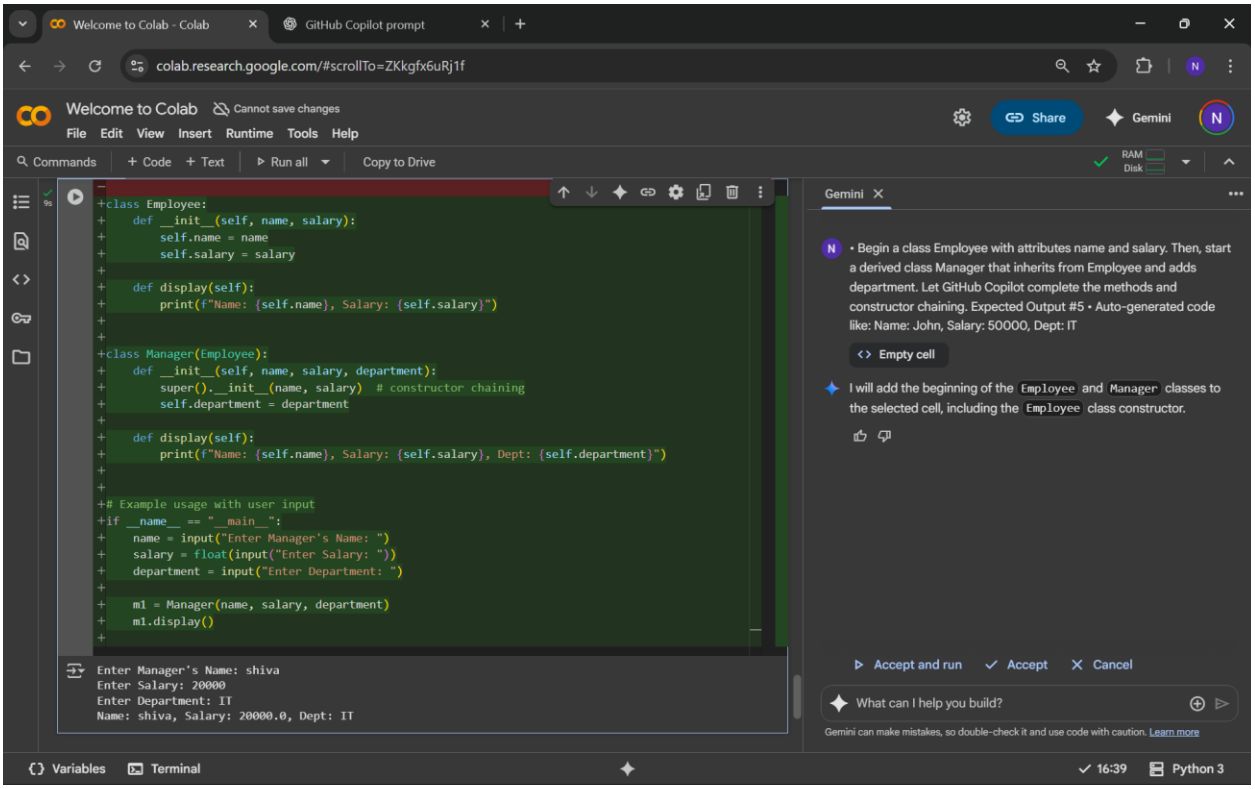
Accept and run
Accept
Cancel
What can I help you build?
Gemini can make mistakes, so double-check it and use code with caution. Learn more
Variables
Terminal
16:24
Python 3

Task 5: Auto-Complete Class with Inheritance (Employee → Manager)

- Begin a class Employee with attributes name and salary. Then, start a derived class Manager
  that inherits from Employee and adds department. Let GitHub Copilot complete the methods
  and constructor chaining.

Expected Output #5

- Auto-generated code like:
  Name: John, Salary: 50000, Dept: IT



```
-
+class Employee:
+    def __init__(self, name, salary):
+        self.name = name
+        self.salary = salary
+
+    def display(self):
+        print(f"Name: {self.name}, Salary: {self.salary}")
+
+
+class Manager(Employee):
+    def __init__(self, name, salary, department):
+        super().__init__(name, salary)  # constructor chaining
+        self.department = department
+
+    def display(self):
+        print(f"Name: {self.name}, Salary: {self.salary}, Dept: {self.department}")
+
+
+# Example usage with user input
+if __name__ == "__main__":
+    name = input("Enter Manager's Name: ")
+    salary = float(input("Enter Salary: "))
+    department = input("Enter Department: ")
+
+    m1 = Manager(name, salary, department)
+    m1.display()
+
```

```
Enter Manager's Name: shiva
Enter Salary: 20000
Enter Department: IT
Name: shiva, Salary: 20000.0, Dept: IT
```

Gemini

N • Begin a class Employee with attributes name and salary. Then, start a derived class Manager that inherits from Employee and adds department. Let GitHub Copilot complete the methods and constructor chaining. Expected Output #5 • Auto-generated code like: Name: John, Salary: 50000, Dept: IT

<> Empty cell

I will add the beginning of the `Employee` and `Manager` classes to the selected cell, including the `Employee` class constructor.

Gemini can make mistakes, so double-check it and use code with caution. Learn more