

PROGRAM : B.TECH/CSE
SPECIALIZATION : AIML
COURSE TITLE : AI ASSISTANT CODING
COURSE CODE : 24CS101PC214
SEMESTER : 3RD
NAME OF THE STUDENT : KARNAKANTI GODADEVI
ENROLLMENT NO : 2403A52003
BATCH NO : 01

Task Description#1

- Task #1 – Syntax Error in Conditionals

```
python  
  
a = 10  
if a = 10:  
    print("Equal")
```

Expected Output#1

- Corrected function with syntax fix

Prompt:

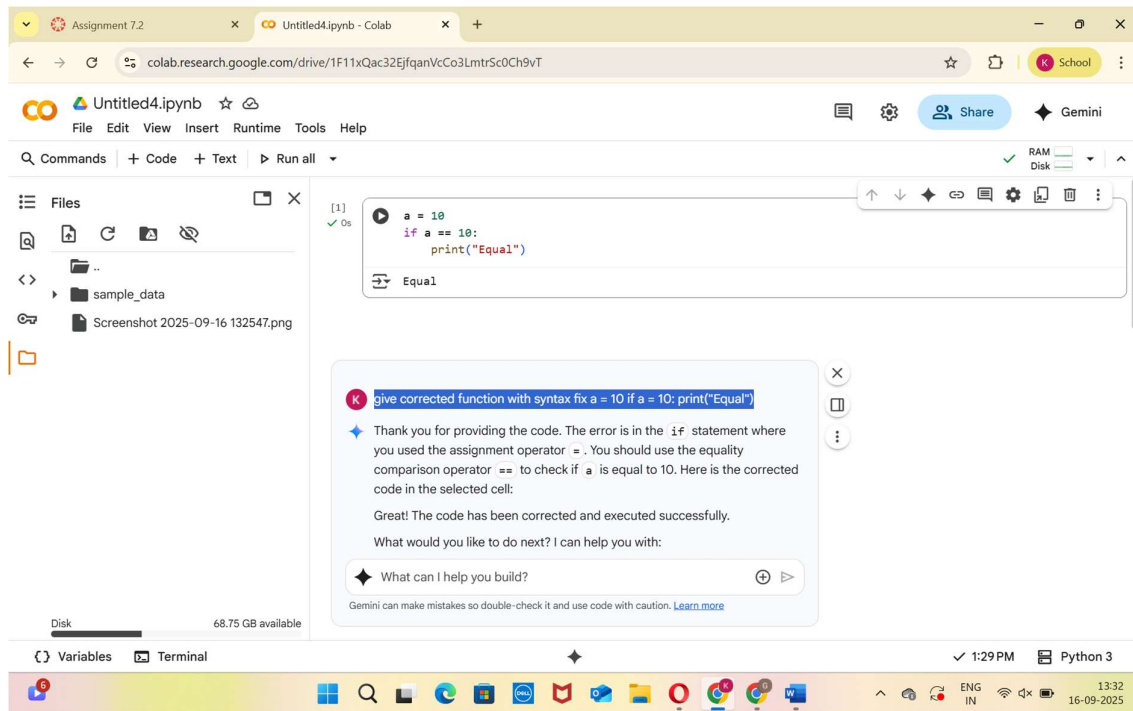
give corrected function with syntax fix

a = 10

if a = 10:

print("Equal")

Code & output:



Observation:

This code snippet demonstrates a basic conditional statement in Python. It initializes a variable `a` with the value 10 and then uses an `if` statement to check if `a` is equal to 10. If the condition is true, it prints the string "Equal". This is a simple example of controlling program flow based on a condition.

Task Description#2 (Loops)

- Task #2 – Loop Off-By-One Error.

```
def sum_upto_n(n):
    total = 0
    for i in range(1, n):
        total += i
    return total
```

Expected Output#2

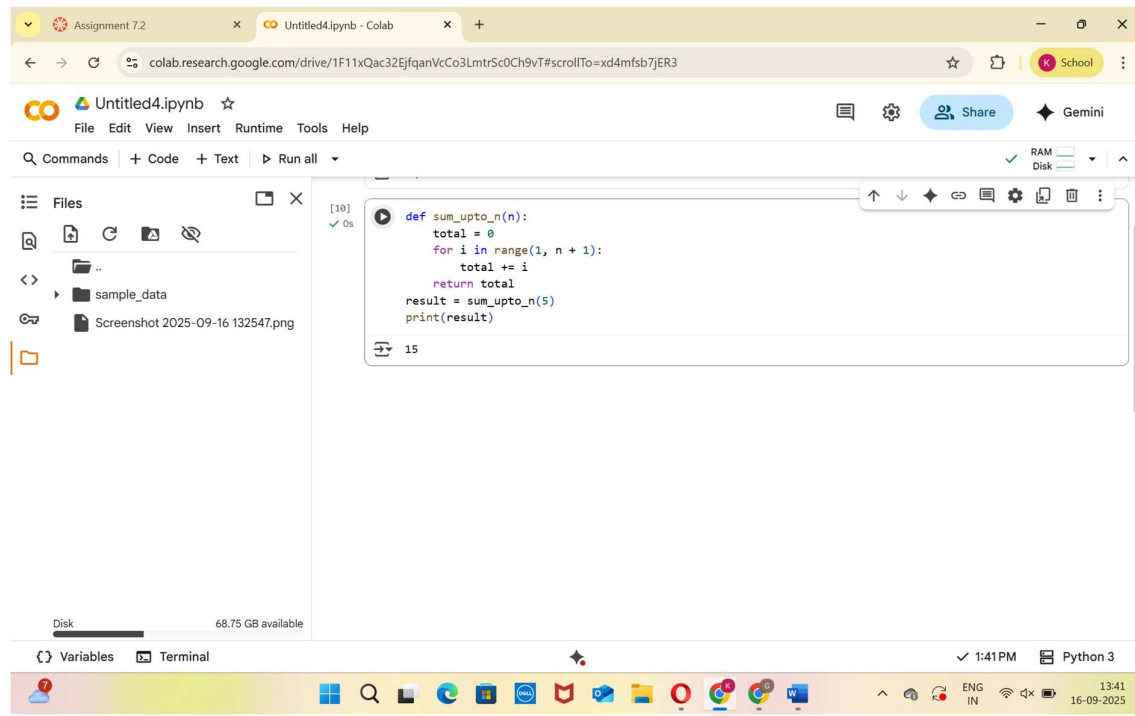
- AI fixes increment/decrement error

Prompt:

give the code by fixing the error

```
def sum_upto_n(n):  
    total = 0  
    for i in range(1, n):  
        total += i  
    return total
```

Code & output:



The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code cell contains the following Python code:

```
[10]: def sum_upto_n(n):  
      total = 0  
      for i in range(1, n + 1):  
          total += i  
      return total  
      result = sum_upto_n(5)  
      print(result)
```

The output of the code cell is 15, indicating that the sum of integers from 1 to 5 is 15. The notebook interface includes a file explorer on the left showing a folder named 'sample_data' and a file named 'Screenshot 2025-09-16 132547.png'. The bottom status bar shows the time as 1:41 PM and the Python version as Python 3.

Observation:

This code defines a function `sum_upto_n` that calculates the sum of all integers from 1 up to a given number `n`. It uses a `for` loop to iterate through the numbers and accumulate the sum in the `total` variable. Finally, it calls the function with `n=5` and prints the result.

Task Description#3

- Error: AttributeError

```
class User:
    def __init__(self, name):
        self.name = name

u = User("Alice")
print(u.getName())
```

Expected Output#3

- Identify the missing method and correct the code.

Prompt:

Give the corrected code

```
class User:
```

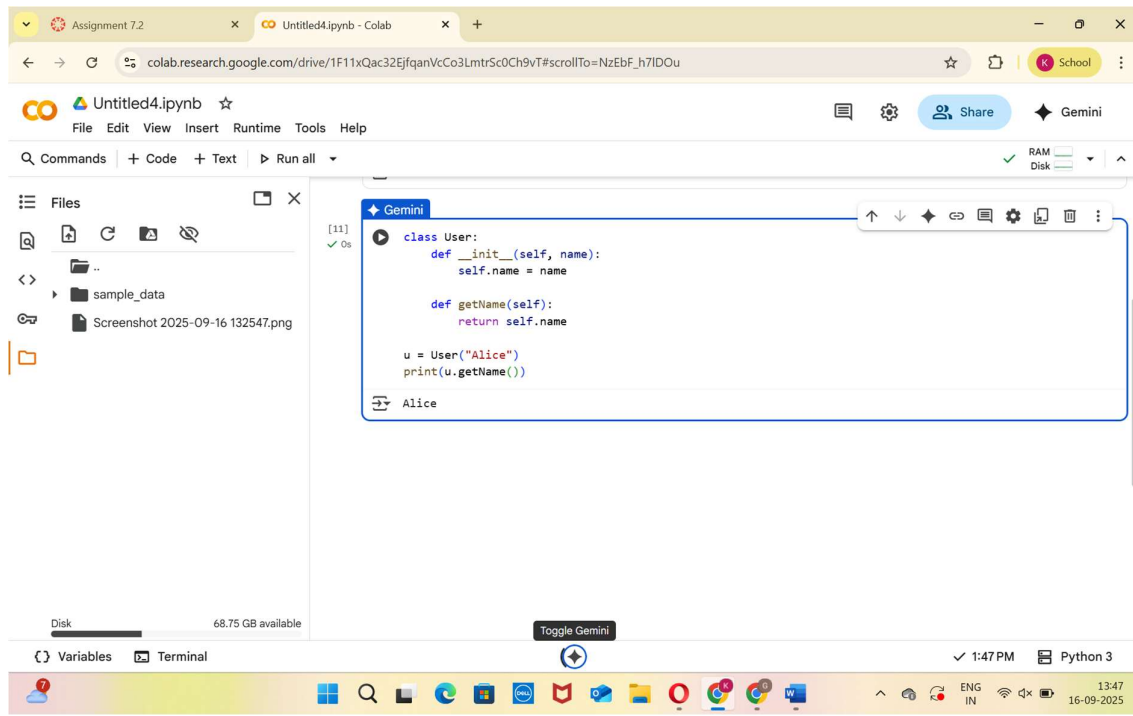
```
    def __init__(self, name):
```

```
        self.name = name
```

```
u = User("Alice")
```

```
print(u.getName())
```

Code & output:



Observation:

This code demonstrates the creation of a simple Python class `User` with an initializer (`__init__`) to set the user's name and a method `getName` to retrieve the name. It then creates an instance of the `User` class and calls the `getName` method to print the user's name. This illustrates basic object-oriented programming concepts like encapsulation and methods.

Task Description#4

- **Incorrect Class Attribute Initialization**

```
class Car:
    def start():
        print("Car started")

mycar = Car()
mycar.start()
```

Expected Output#4

- Detect missing `self` and initialize attributes properly.

Prompt:

give the corrected code by detecting missing `self` and also initialize attributes properly.

```
class Car:
```

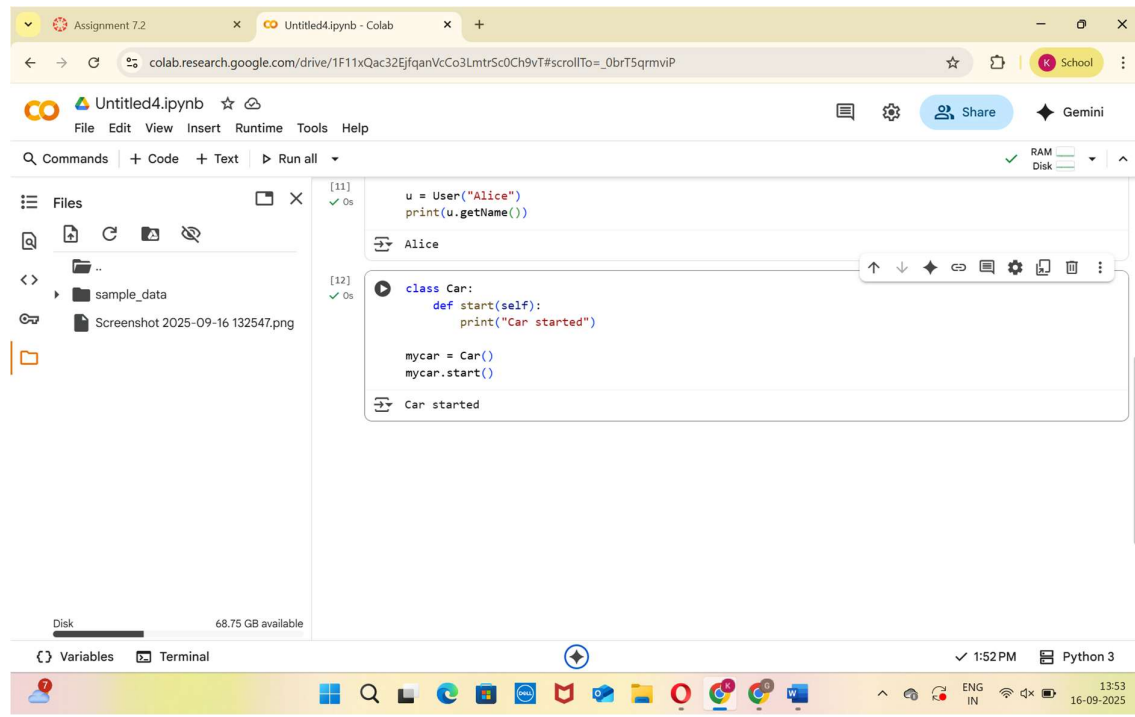
```
    def start():
```

```
        print("Car started")
```

```
mycar = Car()
```

```
mycar.start()
```

Code & output:



Observation:

This code demonstrates the basic concept of creating a class and an object in Python, and calling a method on that object. It shows how to define a class with a method and then instantiate that class to create an object that can access the method.

Task Description#5

- Conditional Logic Error in Grading System

```
def grade_student(score):  
    if score < 40:  
        return "A"  
    elif score < 70:  
        return "B"  
    else:  
        return "C"
```

Expected Output#5

- Detect illogical grading and correct the grade levels.

Prompt:

Give correct version of code by detecting illogical grading and correct grade levels

```
def grade_student(score):
```

```
    if score<40:
```

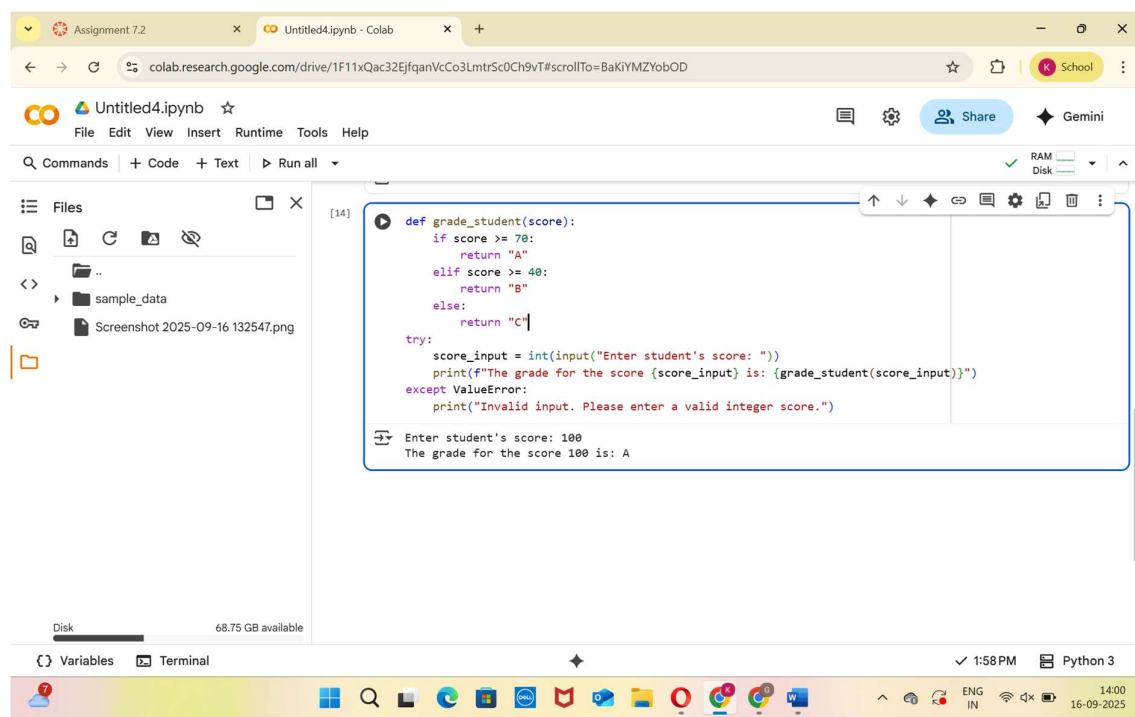
```
        return "A"
```

```
    elif score<70:
```

```
        return "B"
```

```
    else: return "C"
```

Code & output:



The screenshot shows a Google Colab notebook titled 'Untitled4.ipynb'. The code cell contains a function `grade_student(score)` that returns 'A' for scores < 40, 'B' for scores < 70, and 'C' otherwise. Below the function definition, there is a `try` block that prompts the user to enter a score, calls the function, and prints the result. An `except ValueError` block handles invalid input. The output of the code cell shows the user entering '100' and the program printing 'The grade for the score 100 is: A'. The interface includes a file explorer on the left, a command bar at the top, and a status bar at the bottom.

```
[14]: def grade_student(score):  
        if score <= 70:  
            return "A"  
        elif score <= 40:  
            return "B"  
        else:  
            return "C"  
  
    try:  
        score_input = int(input("Enter student's score: "))  
        print(f"The grade for the score {score_input} is: {grade_student(score_input)}")  
    except ValueError:  
        print("Invalid input. Please enter a valid integer score.")  
  
    Enter student's score: 100  
    The grade for the score 100 is: A
```

Observation:

This code defines a function `grade_student` that takes a score as input and returns a letter grade based on the provided grading scale. It then prompts the user to enter a score, handles potential invalid input, and prints the corresponding grade.