

## LAB ASSIGNMENT-7.2

< AI Assisted Coding >

NAME: V.VAMSHI

HALLTICKETNUMBER: 2403A52016

BATCH NUMBER: 02

### Task Description#1

- Task #1 – Syntax Error in Conditionals

```
a=10
```

```
if a=10:
```

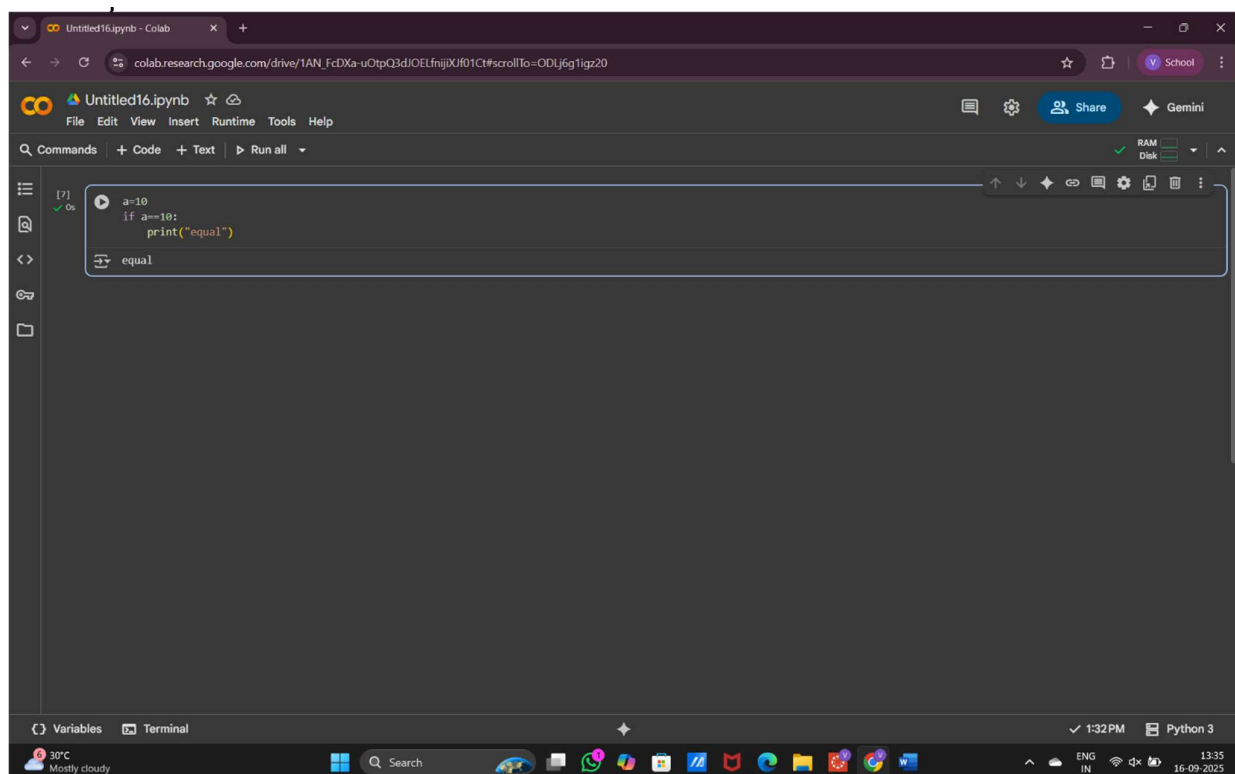
```
    print("equal")
```

Expected Output#1

- Corrected function with syntax fix

### PROMPT:

a=10 if a=10: print("equal").fix this code without any errors and give the



## OBSERVATION:

This code is run successfully using the assignment operator = inside an if statement condition. In Python, to check for equality, you should use the comparison operator ==. I will fix this error by changing = to == in the if statement.

## Task Description#2

- Task #2 – Loop Off-By-One Error.

Def\_sum\_upto\_n(n):

Total=0

For i in range(1,n):

Total +=i

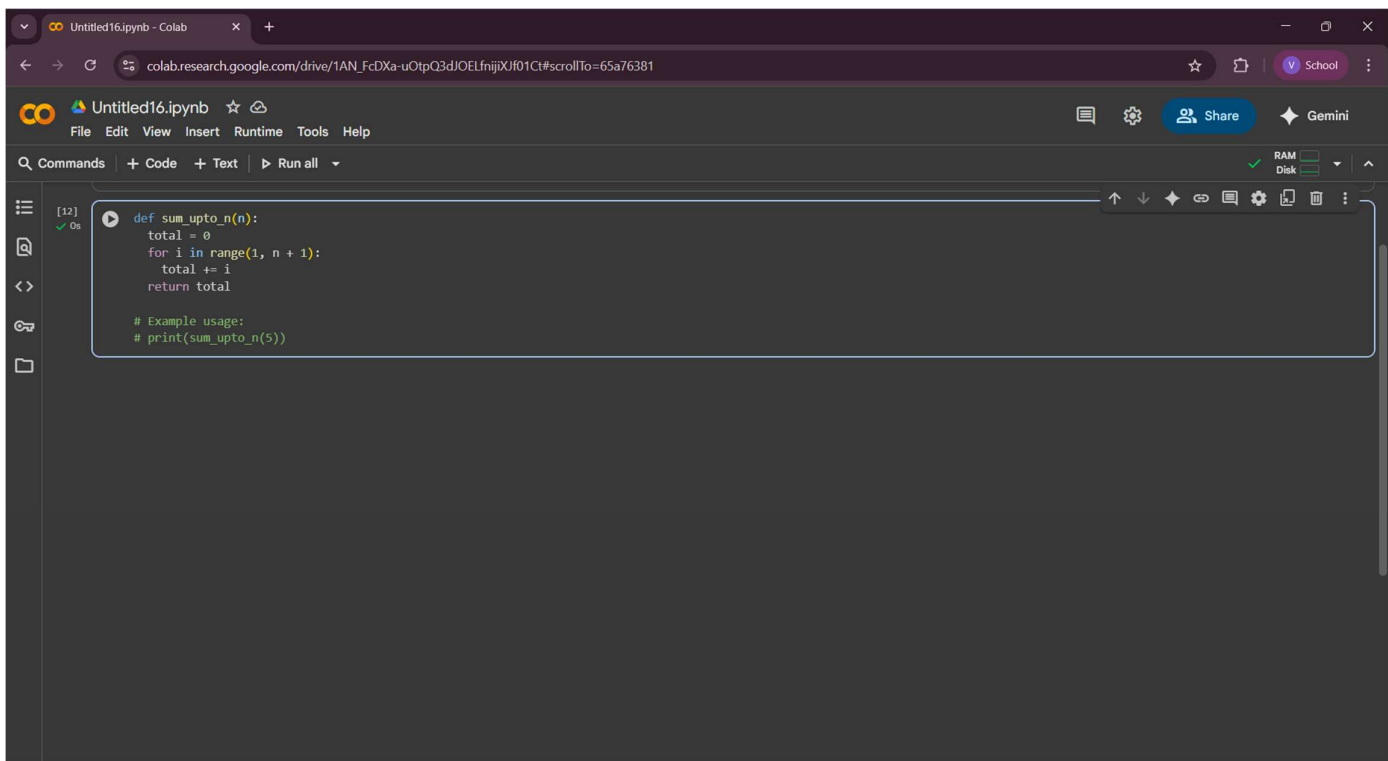
Return total

## Expected Output#2

- AI fixes increment/decrement error

## PROMPT:

Def\_sum\_upto\_n(n): Total=0 For i in range(1,n): Total +=i Return total. Fix this code without any errors.



The screenshot shows a Google Colab notebook interface. The browser address bar displays a URL from colab.research.google.com. The notebook is titled 'Untitled16.ipynb'. The code editor contains a Python function definition and an example usage. The function 'def sum\_upto\_n(n):' initializes 'total = 0', loops through 'range(1, n + 1)', increments 'total' by 'i', and returns 'total'. The example usage shows '# print(sum\_upto\_n(5))'. The output area shows a successful execution with the message '[12] ✓ 0s'.

```
[12] ✓ 0s
def sum_upto_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Example usage:
# print(sum_upto_n(5))
```

## OBSERVATION:

This code defines a function called `sum_upto_n` that takes one input, `n`. It calculates the sum of all whole numbers starting from 1 up to and including `n`. It does this by starting a total at 0, then looping through each number from 1 to `n` and adding it to the total. Finally, it gives back the final calculated total.

## Task Description#3

- Error: `AttributeError`

Class user:

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

```
    Self.name=name
```

```
U=user("Alice")
```

```
Print(u.getName())
```

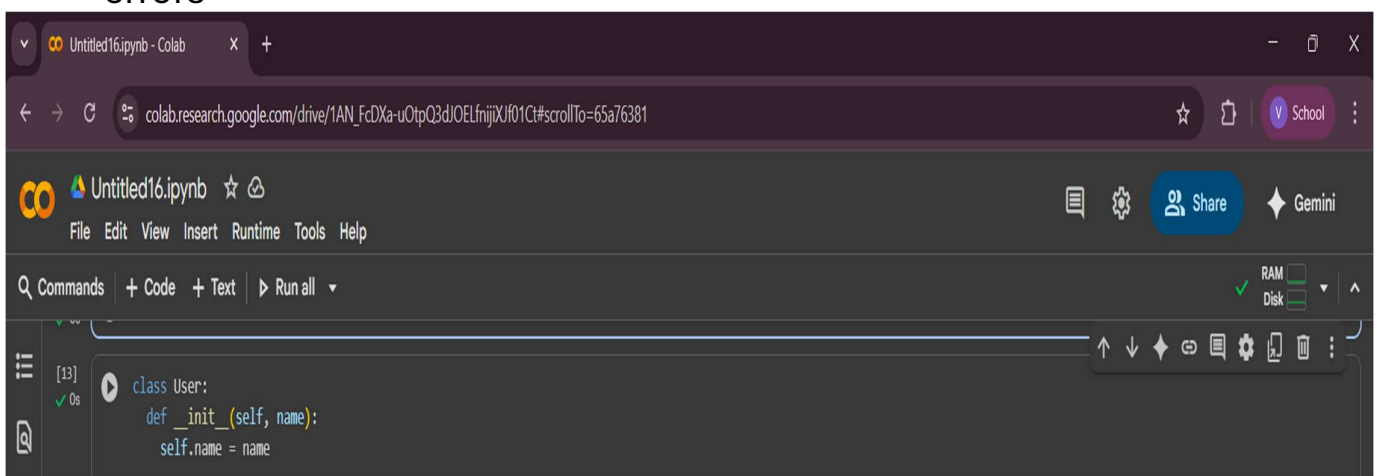
## Expected Output#3

- Identify the missing method and correct the code.

## PROMPT:

Class user: `Def __init__(self,name): Self.name=name`

`U=user("Alice") Print(u.getName())` fix this code without any errors



### OBSERVATION:

This code defines a simple blueprint for creating "User" objects. When you create a User object, you give it a name. The `__init__` method is like a setup process that stores this name inside the object. After creating a User object named `u` with the name "Alice", the code then prints out the name that's stored within that `u` object, which is "Alice".

## Task Description#4

### Incorrect Class Attribute Initialization

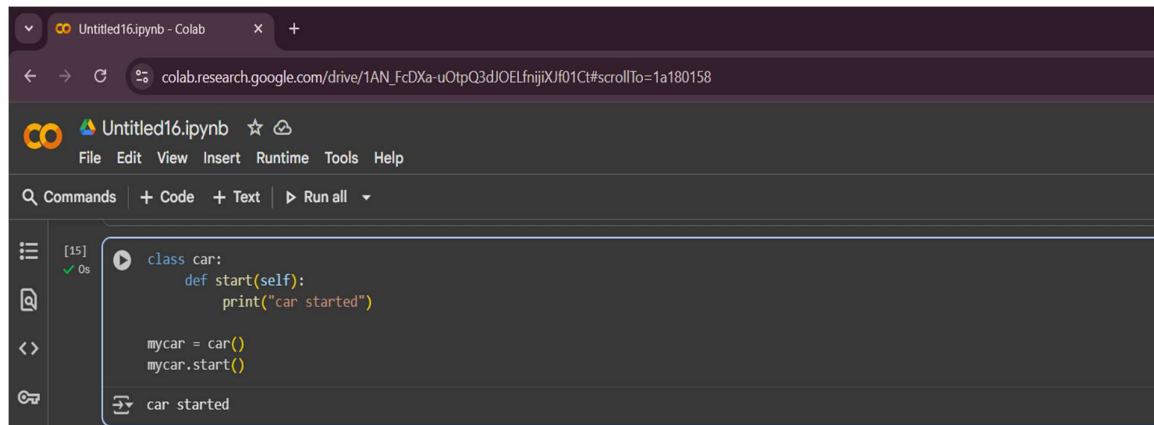
```
class car:
    def start():
        print("car started")
mycar = car()
mycar.start()
```

### Expected Output#4

- Detect missing self and initialize attributes properly.

### PROMPT:

```
class car:
    def start():
        print("car started")
mycar = car()
mycar.start()
fix this code without any errors
```



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1AN_FcDXa-uOtpQ3dJOELfnijiXJf01Ct#scrollTo=1a180158`. The notebook title is "Untitled16.ipynb". The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar, there are tabs for "Commands", "+ Code", "+ Text", and a "Run all" button. The code editor shows the following Python code:

```
[15] class car:
      def start(self):
          print("car started")

      mycar = car()
      mycar.start()
```

The output of the code is displayed below the editor: `car started`. The output is preceded by a green checkmark and the text "[15] ✓ Os".

## OBSERVATION:

This code defines a basic blueprint for a car. Inside this blueprint, there's a function called `start`. When you create a specific car based on this blueprint (like `mycar`), you can then tell that specific car to `start()`. The `start` function simply prints "car started". So, when you run this code, it creates a car object and then tells it to start, resulting in "car started" being printed.

## Task Description#5

- Conditional Logic Error in Grading System

Def `grade_student(score)`:

If `score < 40`:

Return "A"

If `score < 70`:

Return "B"

Else:

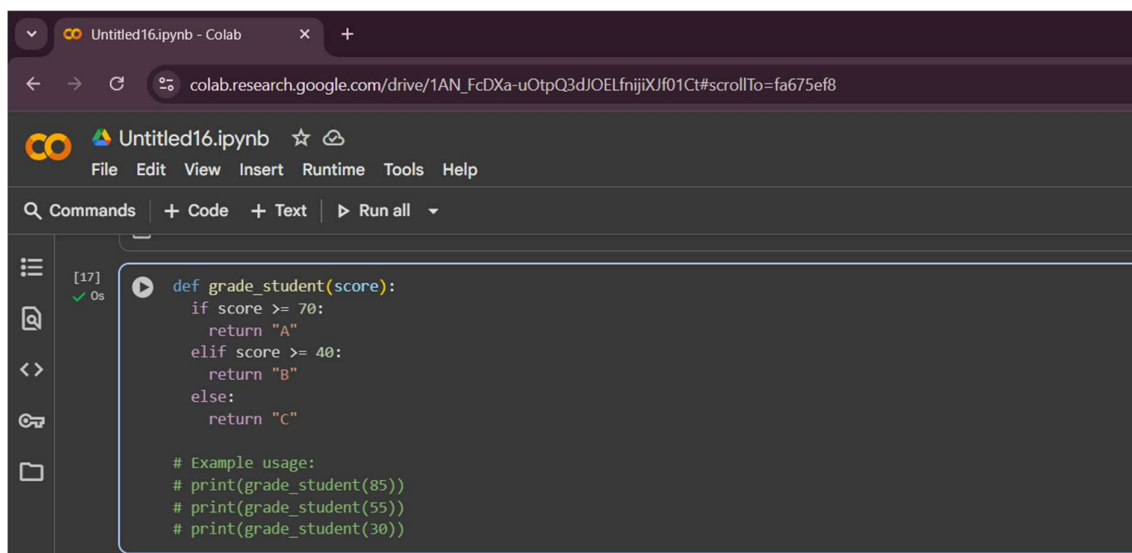
Return "c"

## Expected Output#5

- Detect illogical grading and correct the grade levels

## PROMPT:

Def grade\_student(score): If score < 40: Return "A" If score < 70:  
Return "B" Else: Return "c" fix this code without any errors

A screenshot of a Google Colab notebook interface. The browser address bar shows a URL from colab.research.google.com. The notebook is titled 'Untitled16.ipynb'. The code editor shows a Python function 'def grade\_student(score):' with an if-elif-else structure. The function returns 'A' for scores 70 and above, 'B' for scores 40 and above, and 'c' for scores below 40. Below the function definition, there are example usage lines: '# print(grade\_student(85))', '# print(grade\_student(55))', and '# print(grade\_student(30))'. The left sidebar shows the notebook's file explorer and a list of cells, with the current cell labeled '[17]' and '0s'.

## OBSERVATION:

This function grade\_student takes a student's score as input and figures out their letter grade based on that score. It first checks if the score is 70 or higher; if it is, they get an "A". If not, it then checks if the score is 40 or higher; if it is, they get a "B". If the score isn't 70 or above *and* isn't 40 or above, then it must be below 40, and in that case, the student gets a "C". So, it's a straightforward way to assign grades based on different score ranges.

