

# AI ASSISTED CODING

**NAME : NAREDDY JASMITHA**

**ENROLL NO : 2403A51281**

**COURSE : AI ASSISTED CODE**

**BRANCH : CSE-AIML**

**DATE : 16-09-2025**

## Task Description#1

- Task #1 – Syntax Error in Conditionals

```
python

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

## PROMPT:

Generate a function with syntax fix a=10 if a=10:  
print("Equal")

## Expected Output#1

- **Corrected function with syntax fix**

```
[1] ✓ 0s
def check_equality(a):
    """
    Checks if the input variable 'a' is equal to 10 and prints "Equal" if it is.

    Args:
        a: The variable to check.
    """
    if a == 10:
        print("Equal")

# Example usage:
check_equality(10)
check_equality(5)
```

Equal

## EXPLANATION:

Inside the function, it checks if the value of `a` is equal to 10 using an `if` statement. If the condition is true (i.e., `a` is equal to 10), it prints the word "Equal" to the console.

The code then provides two examples of how to use the function:

- `check_equality(10)` will print "Equal" because 10 is equal to 10.
- `check_equality(5)` will not print anything because 5 is not equal to 10.

## Task Description#2 (Loops)

- Task #2 – Loop Off-By-One Error.

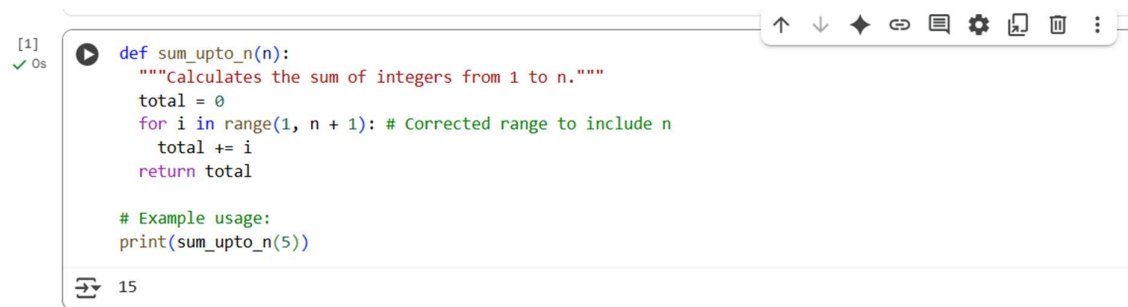
```
def sum_upto_n(n):
    total = 0
    for i in range(1, n):
        total += i
    return total
```

## PROMPT:

Generate AI fixes increment or decrement 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**



The screenshot shows a Jupyter Notebook interface. On the left, there is a status bar with '[1]' and '✓ 0s'. The main area contains a code cell with the following Python code:

```
def sum_upto_n(n):  
    """Calculates the sum of integers from 1 to n."""  
    total = 0  
    for i in range(1, n + 1): # Corrected range to include n  
        total += i  
    return total  
  
# Example usage:  
print(sum_upto_n(5))
```

Below the code cell, the output '15' is displayed.

## EXPLANATION:

- **def sum\_upto\_n(n):** : This line defines the function named `sum_upto_n` which takes one argument, `n`.
- **total = 0** : This initializes a variable `total` to 0. This variable will store the running sum.
- **for i in range(1, n + 1):** : This is a `for` loop that iterates through a sequence of numbers starting from 1 and going up to and including `n`. The `range(1, n + 1)` function generates this sequence.
- **total += i** : In each iteration of the loop, the current number `i` is added to the `total`.
- **return total** : After the loop finishes, the function returns the final calculated `total`.
- **print(sum\_upto\_n(5))** : This line calls the `sum_upto_n` function with `n=5` and prints the returned value, which is the sum of numbers from 1 to 5 ( $1+2+3+4+5 = 15$ ).

### Task Description#3

- Error: AttributeError

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

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

### PROMPT:

Generate identify missing method and correct the code class user: def \_\_int\_\_(self, name): self.name = name u = user ("Alice") print (u.getName())

### Expected Output#3

- **Identify the missing method and correct the code.**

```
class User: # Class names are typically capitalized
    def __init__(self, name): # __int__ should be __init__
        self.name = name

    def getName(self): # Added the missing getName method
        return self.name

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

Alice

### EXPLANATION:

---

Convention, class names in Python start with a capital letter.

- `def __init__(self, name):`: This is the constructor method for the `User` class. It's called automatically when you create a new `User` object.
  - `self`: Refers to the instance of the class being created.
  - `name`: This is a parameter that will be passed when you create a `User` object (e.g., "Alice").
  - `self.name = name`: This line assigns the value of the `name` parameter to an attribute of the object called `name`. Each `User` object will have its own `name`.
- `def getName(self):`: This defines a method called `getName` within the `User` class.
  - `self`: Refers to the instance of the class that the method is called on.
  - `return self.name`: This method returns the value of the `name` attribute for that specific `User` object.
- `u = User("Alice")`: This line creates a new instance (object) of the `User` class and assigns it to the variable `u`. When this line is executed, the `__init__` method is called with `"Alice"` as the `name`.

## Task Description#4

- **Incorrect Class Attribute Initialization**

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

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

## PROMPT:

Generate missing self and initialize attributes properly

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

## Expected Output#4

- **Detect missing self and initialize attributes properly**

```
class car:
    def start(self): # Added 'self' as the first parameter
        print("car started")

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

car started

## EXPLANATION:

- **class car:** : This line defines a new class named **car**.
- **def start(self):** : This defines a method called **start** within the **car** class.
  - **self** : This is a required parameter in instance methods of a class. It refers to the instance of the class that the method is called on.
  - **print("car started")** : This line simply prints the text "car started" to the console when the **start** method is called.
- **mycar = car()** : This line creates a new instance (object) of the **car** class and assigns it to the variable **mycar**.
- **mycar.start()** : This line calls the **start** method on the **mycar** object. This will execute the code inside the **start** method, printing "car started".

## Task Description#5

- Conditional Logic Error in Grading System

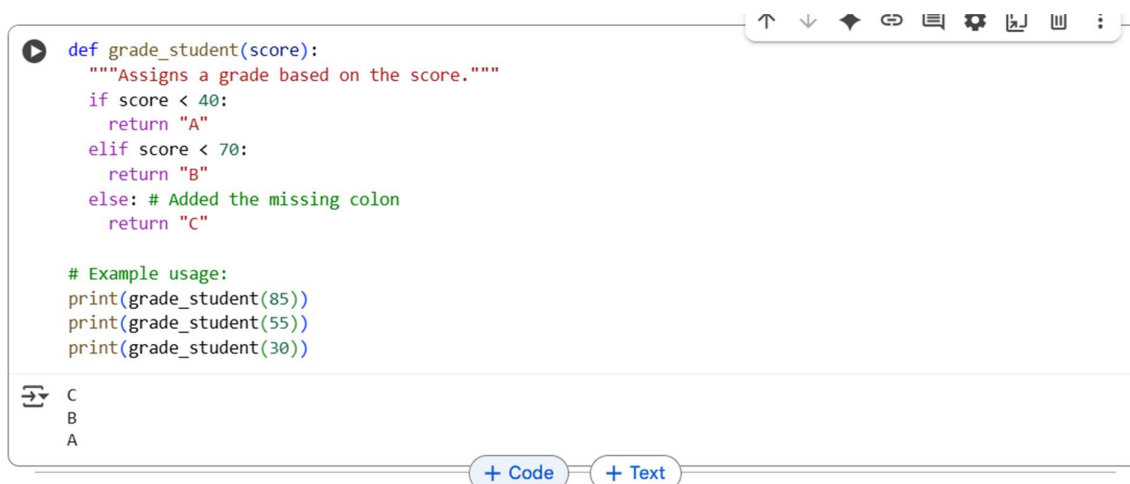
```
def grade_student(score):  
    if score < 40:  
        return "A"  
    elif score < 70:  
        return "B"  
    else:  
        return "C"
```

## PROMPT:

Generate Detect illogical grading and correct the grade levels 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



```
def grade_student(score):  
    """Assigns a grade based on the score."""  
    if score < 40:  
        return "A"  
    elif score < 70:  
        return "B"  
    else: # Added the missing colon  
        return "C"  
  
# Example usage:  
print(grade_student(85))  
print(grade_student(55))  
print(grade_student(30))
```

C  
B  
A

+ Code + Text

## EXPLANATION:

- `def grade_student(score):` : This defines a function named `grade_student` that takes one argument, `score`.
- `if score < 40:` : This is the first condition. If the `score` is less than 40, the function immediately returns the string "A".
- `elif score < 70:` : This is the second condition, which is checked only if the first condition is false (i.e., if the score is 40 or greater). If the `score` is less than 70 (but 40 or greater), the function returns the string "B".
- `else:` : This is the final case, which is executed if neither of the above conditions is true (i.e., if the score is 70 or greater). In this case, the function returns the string "C".
- The lines starting with `# Example usage:` show how to call the `grade_student` function with different scores and print the returned grade.