**Assignment\_2**

**1.What are the two values of the Boolean data type? How do you write them?**

The Boolean data type in Python has two values: True and False. These values are used to represent the truthiness or falseness of expressions and conditions.

True: Represents a true condition or a positive result.

False: Represents a false condition or a negative result.

**2. What are the three different types of Boolean operators?**

In Python, there are three different types of Boolean operators used to perform logical operations on Boolean values or expressions. These operators are used to create complex conditions and control the flow of your programs. The three types of Boolean operators are:

Logical AND (and):

The and operator returns True if both operands are True, otherwise, it returns False.

Logical OR (or):

The or operator returns True if at least one of the operands is True, and it returns False only if both operands are False.

Logical NOT (not):

The not operator is a unary operator that returns the opposite of the operand's Boolean value. If the operand is True, not returns False, and if the operand is False, not returns True.

**3. Make a list of each Boolean operator's truth tables (i.e. every possible combination of Boolean values for the operator and what it evaluate ).**

1. Logical AND (`and`):

| Operand 1 | Operand 2 | Result |

|-----------|-----------|--------|

| False | False | False |

| False | True | False |

| True | False | False |

| True | True | True |

2. Logical OR (`or`):

| Operand 1 | Operand 2 | Result |

|-----------|-----------|--------|

| False | False | False |

| False | True | True |

| True | False | True |

| True | True | True |

3. Logical NOT (`not`):

| Operand | Result |

|---------|--------|

| False | True |

| True | False |

**4. What are the values of the following expressions?**

**(5 > 4) and (3 == 5)**

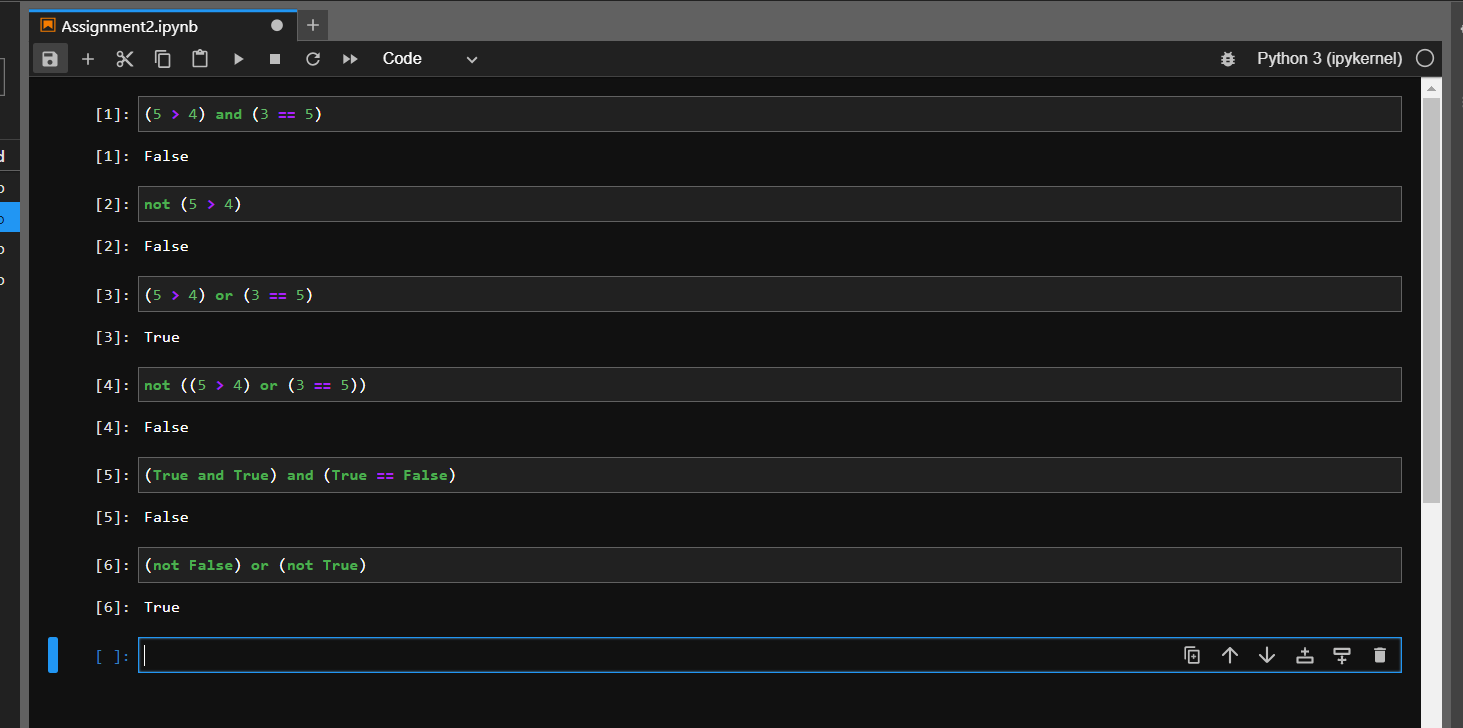
**not (5 > 4)**

**(5 > 4) or (3 == 5)**

**not ((5 > 4) or (3 == 5))**

**(True and True) and (True == False)**

**(not False) or (not True)**



Explanation for each expression:

* **1. `(5 > 4) and (3 == 5)`:**

The first part `(5 > 4)` is `True`, but the second part `(3 == 5)` is `False`. Using `and`, both parts need to be `True` for the result to be `True`, so the overall result is `False`.

* **2. `not (5 > 4)`:**

The expression `(5 > 4)` is `True`, and using `not` negates it, resulting in `False`.

* **3. `(5 > 4) or (3 == 5)`:**

The first part `(5 > 4)` is `True`, and the second part `(3 == 5)` is `False`. Using `or`, if at least one part is `True`, the result is `True`, so the overall result is `True`.

* **4. `not ((5 > 4) or (3 == 5))`:**

Similar to the previous explanation, the overall expression evaluates to `False`, and using `not` negates it to `True`.

* **5. `(True and True) and (True == False)`:**

The expression `(True == False)` evaluates to `False`. So, even though the other parts are `True`, using `and` requires all parts to be `True` for the result to be `True`, making the overall result `False`.

* **6. `(not False) or (not True)`:**

`not False` is `True`, and `not True` is `False`. Using `or`, if at least one part is `True`, the result is `True`, so the overall result is `True`.

**5. What are the six comparison operators?**

The six comparison operators in Python are used to compare values and return Boolean results (`True` or `False`). Here they are:

* Equal (`==`): Returns `True` if the values on both sides of the operator are equal.
* Not Equal (`!=`): Returns `True` if the values on both sides of the operator are not equal.
* Greater Than (`>`): Returns `True` if the value on the left side is greater than the value on the right side.
* Less Than (`<`): Returns `True` if the value on the left side is less than the value on the right side.
* Greater Than or Equal To (`>=`): Returns `True` if the value on the left side is greater than or equal to the value on the right side.
* Less Than or Equal To (`<=`): Returns `True` if the value on the left side is less than or equal to the value on the right side.

These operators are commonly used in conditional statements, loops, and other situations where comparisons are required to make decisions or control the flow of a program.

**6. How do you tell the difference between the equal to and assignment operators? Describe a condition and when you would use one.**

The equal to (= =) operator and the assignment (=) operator serve different purposes in Python. It's important to understand the difference between them to avoid confusion.

**Equal To Operator (= =):**

The equal to operator is used to compare two values to check if they are equal. It returns True if the values are equal and False if they are not.

**Assignment Operator (=):**

The assignment operator is used to assign a value to a variable. It takes the value on the right and assigns it to the variable on the left.

**Condition for Usage:**

You would use the equal to operator (==) when you want to compare two values to determine their equality. This is commonly used in conditional statements, such as if statements, to make decisions based on the comparison results.

You would use the assignment operator (=) when you want to assign a value to a variable. This is essential for storing and manipulating data in your program.

In summary, the key difference is that the equal to operator compares values for equality, while the assignment operator assigns a value to a variable. Always be careful not to confuse these operators, as misusing them can lead to unintended behavior in your code.

**7. Identify the three blocks in this code:**

**spam = 0**

**if spam == 10:**

**print('eggs')**

**if spam > 5:**

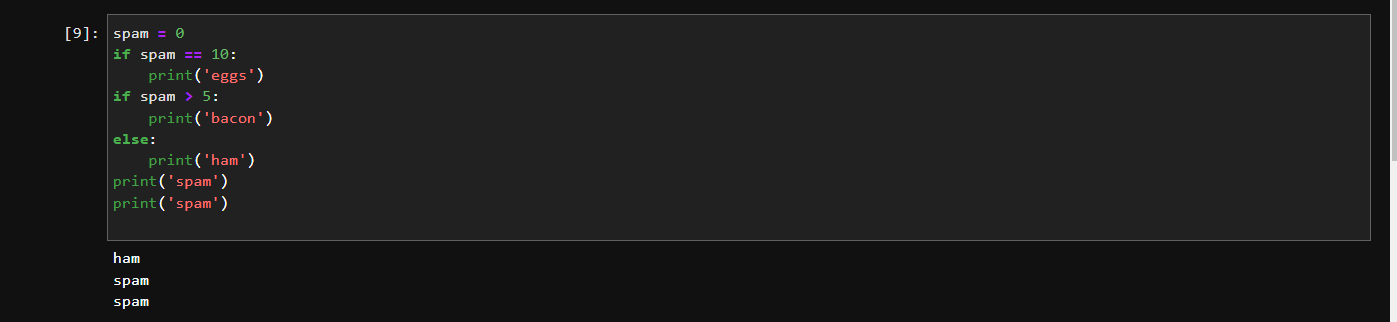
**print('bacon')**

**else:**

**print('ham')**

**print('spam')**

**print('spam')**



**8. Write code that prints Hello if 1 is stored in spam, prints Howdy if 2 is stored in spam, and prints Greetings! if anything else is stored in spam.**

spam = int(input("Enter a value for spam: ")) # Accept user input

if spam == 1:

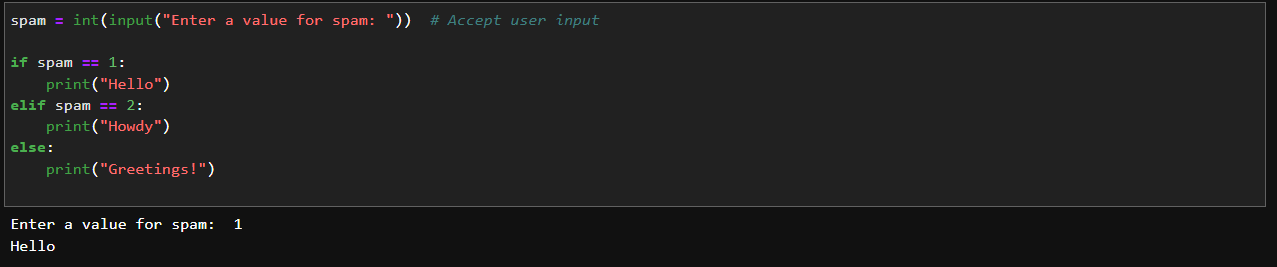
print("Hello")

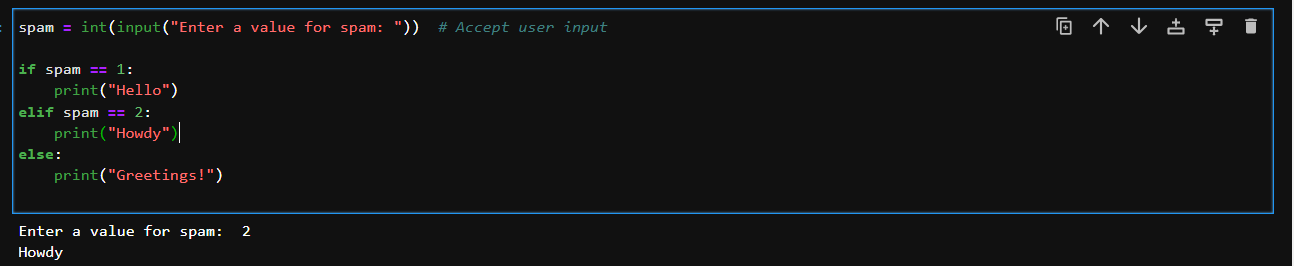
elif spam == 2:

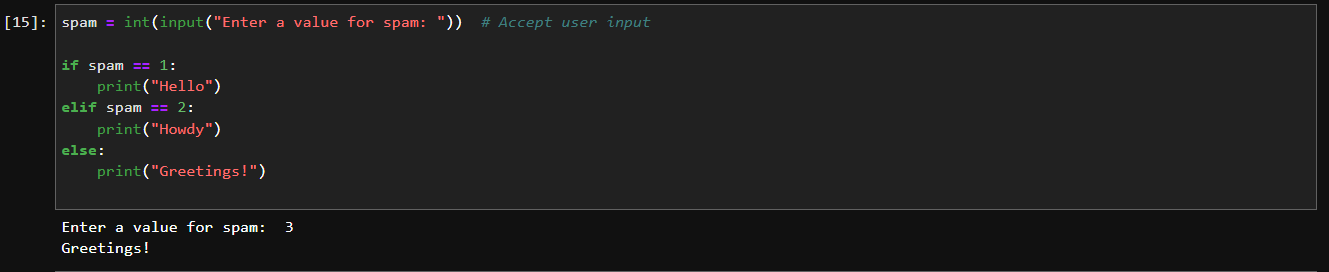
print("Howdy")

else:

print("Greetings!")







**9. If your programme is stuck in an endless loop, what keys you’ll press?**

Ctrl + C (on Windows and Linux) or Command + C (on macOS)

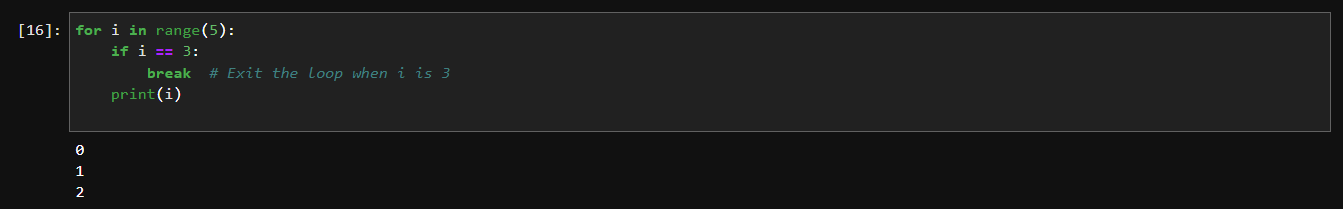
**10. How can you tell the difference between break and continue?**

`break` and `continue` are two keywords used in loops (such as `for` and `while` loops) to control the flow of the loop execution. They serve different purposes:

**Break:**

The `break` statement is used to exit the loop prematurely. When a `break` statement is encountered within a loop, the loop's execution is immediately terminated, and the program continues with the code following the loop.

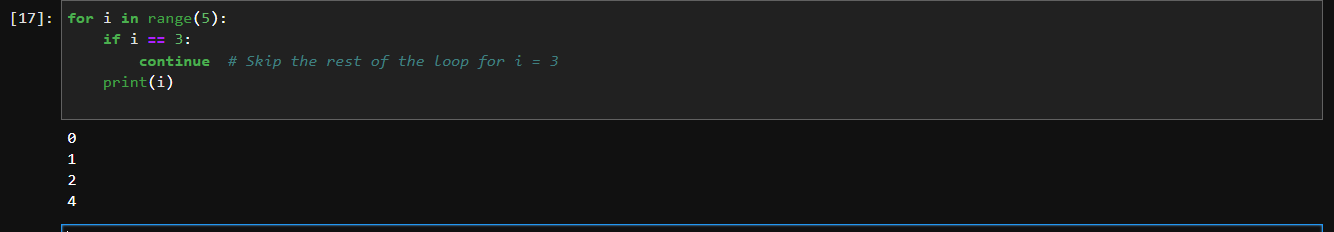
Example:



**Continue:**

The `continue` statement is used to skip the current iteration of the loop and move on to the next iteration. It essentially skips the rest of the loop's body for the current iteration and proceeds with the next iteration.

Example:



In summary, `break` is used to completely exit the loop, while `continue` is used to skip the current iteration and move to the next one. Both statements can help control the flow of the loop based on certain conditions.

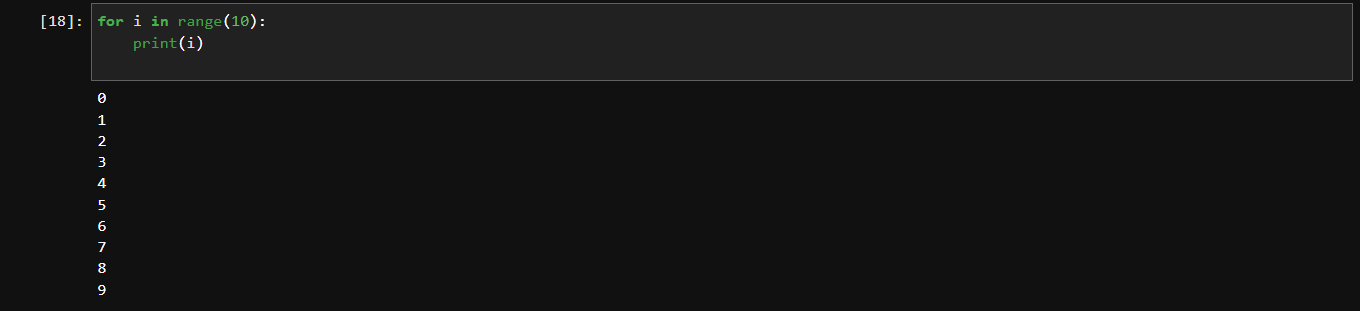
**11. In a for loop, what is the difference between range(10), range(0, 10), and range(0, 10, 1)?**

In a `for` loop, the three expressions `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are used to generate a sequence of numbers that the loop iterates over. However, they are essentially equivalent and produce the same result in this context. Here's the breakdown:

* **`range(10)`:**

This expression generates a sequence of numbers from 0 to 9. The starting value is omitted, so the default starting value of 0 is assumed. The loop will iterate through the numbers 0 to 9 (10 numbers in total).

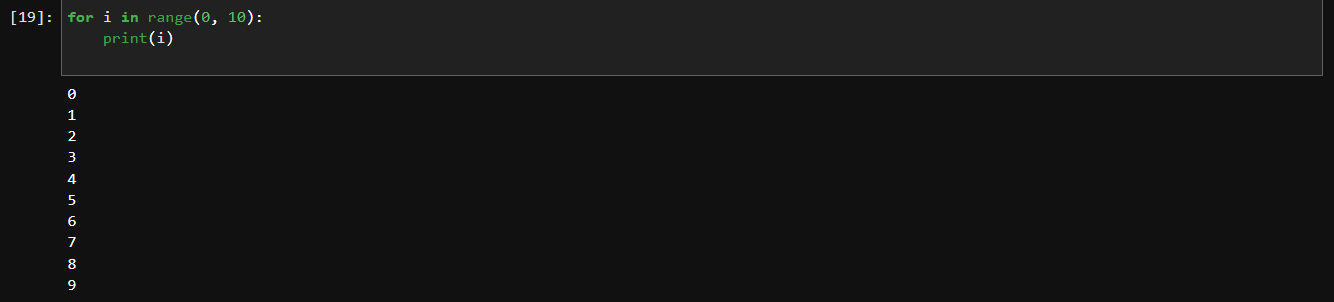
Example:



* **`range(0, 10)`:**

This expression explicitly specifies both the starting and ending values of the range. The loop will iterate through the numbers 0 to 9, just like in the previous example.

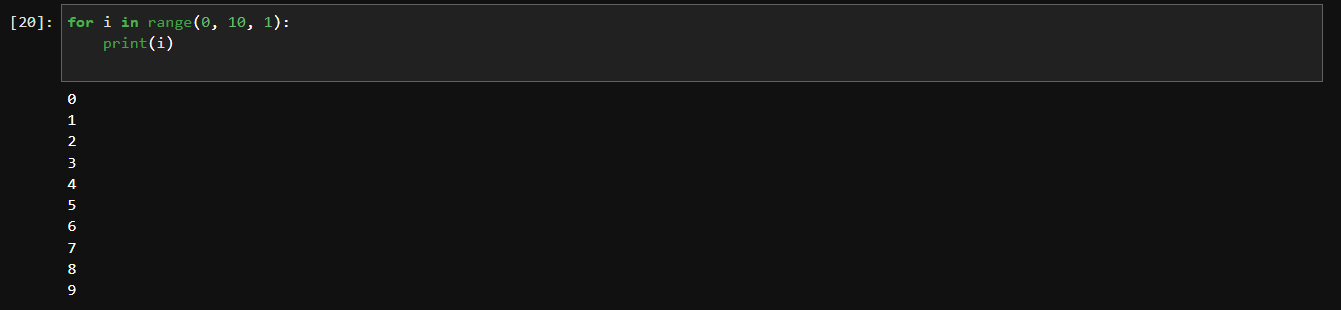
Example:



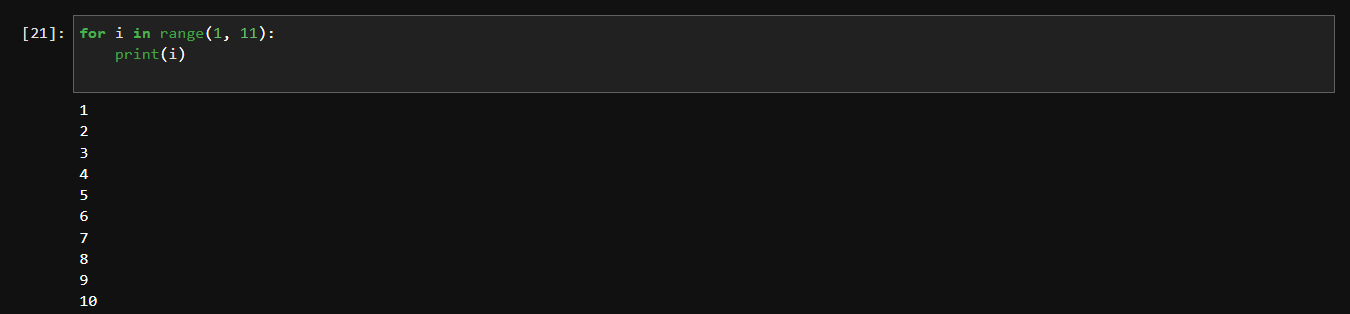
* **`range(0, 10, 1)`:**

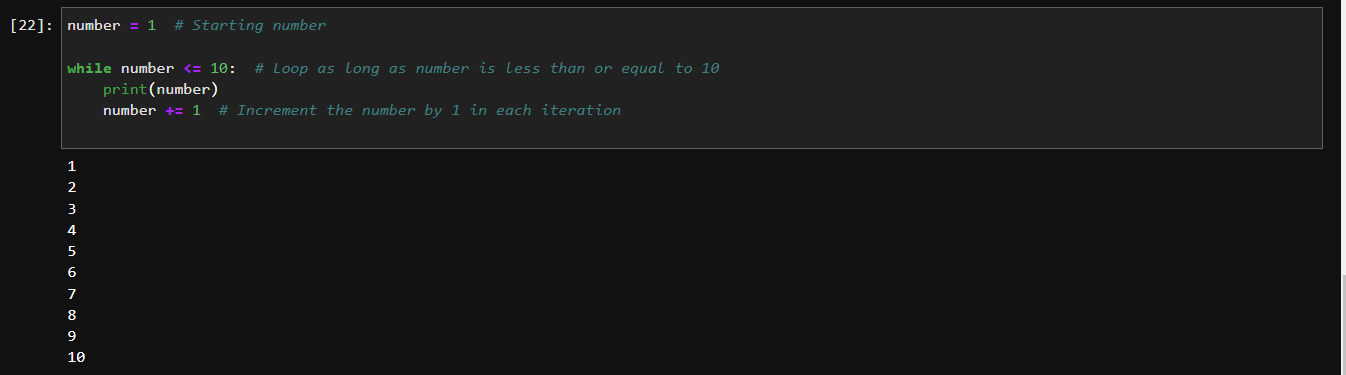
This expression adds an additional step or stride argument. The default step is 1, so this expression is effectively the same as the previous two. It iterates through the numbers 0 to 9 with a step of 1.

Example:



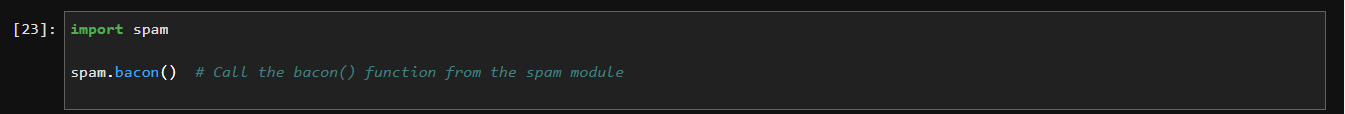
**12. Write a short program that prints the numbers 1 to 10 using a for loop. Then write an equivalent program that prints the numbers 1 to 10 using a while loop.**





**13. If you had a function named bacon() inside a module named spam, how would you call it after importing spam?**

If you have a function named `bacon()` inside a module named `spam`, and you want to call it after importing the `spam` module, you would use the following syntax:



Here, `spam` is the name of the module, and `bacon()` is the function within that module. When you use the syntax `spam.bacon()`, you're specifying that you want to call the`bacon()` function from the `spam` module.