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

**Ans:** *The Boolean data type in Python has two values:*

*1. “True”: This represents a true or affirmative value.*

*2. “False”: This represents a false or negative value.*

*In Python, these Boolean values are written with an initial capital letter: `True` and `False`.*

*Here are some examples of how to use Boolean values in Python:*

*“”” #python*

*x = True*

*y = False*

*if x:*

*print ("x is True")*

*if not y:*

*print ("y is False")”””*

*In this code, `x` is assigned the value `True`, and `y` is assigned the value `False`. The `if` statements then check the truthiness of these Boolean values and print messages accordingly.*

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

**Ans:** *There are three different types of Boolean operators in Python:*

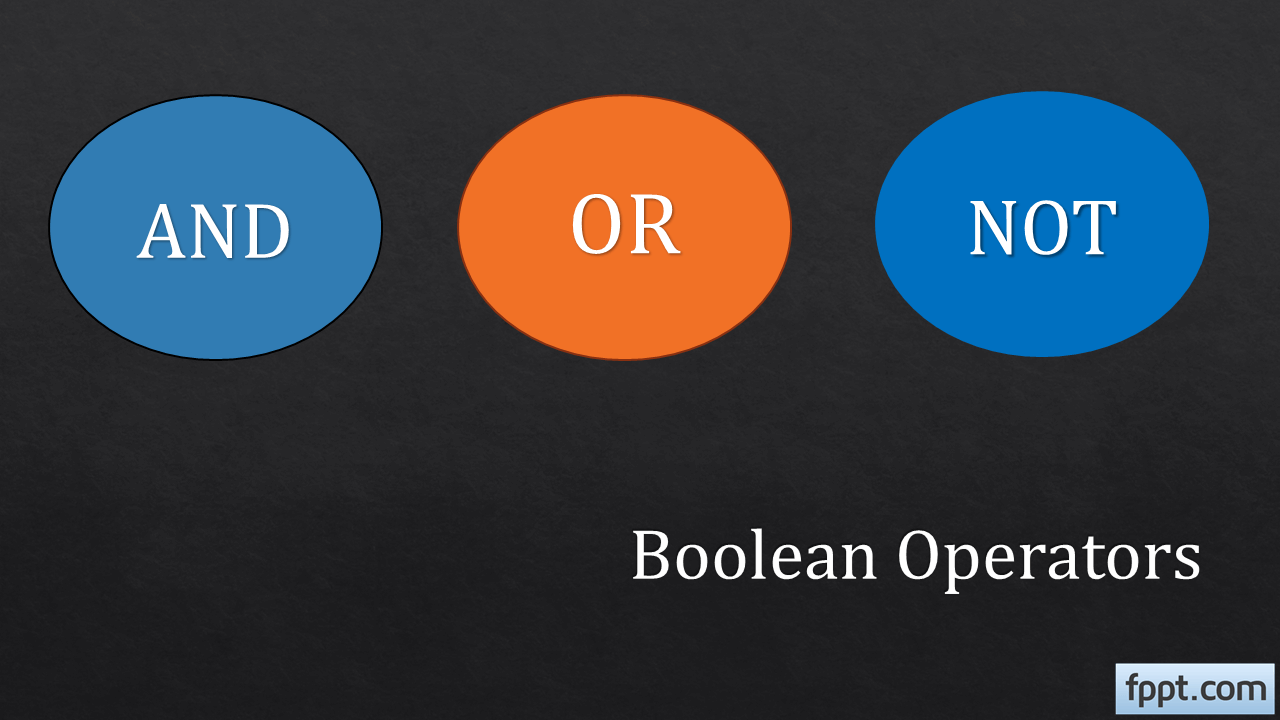
*1. AND Operator (“and”): The “and” operator returns “True” if both of the operands are “True”. Otherwise, it returns `False. `*

*2. OR Operator (“or”): The “or” operator returns `True` if at least one of the operands is “True”. It returns `False` if both operands are `False`.*

*3. NOT Operator (“not”): The “not” operator is a unary operator that returns the opposite Boolean value of its operand. If the operand is `True`, `not` makes it `False`, and if the operand is `False`, `not` makes it `True`.*

*These Boolean operators are commonly used in conditional statements, such as “if” statements, to control the flow of a program based on the evaluation of Boolean expressions.*

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

**Ans:** 

*The truth tables for each of the three Boolean operators: and, or, and not. These tables show every possible combination of Boolean values for the operator and the result of their evaluation.*

* *AND Operator (“and”):*

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Output** |
| *True* | *True* | *True* |
| *True* | *False* | *False* |
| *False* | *True* | *False* |
| *False* | *False* | *False* |

* *OR Operator (“or”):*

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Output** |
| *True* | *True* | *True* |
| *True* | *False* | *True* |
| *False* | *True* | *True* |
| *False* | *False* | *False* |

* *NOT Operator (“not”):*

|  |  |
| --- | --- |
| **Operand 1** | **Output** |
| *True* | *False* |
| *False* | *True* |

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

1. **(5 > 4) and (3 == 5)**
2. **not (5 > 4)**
3. **(5 > 4) or (3 == 5)**
4. **not ((5 > 4) or (3 == 5))**
5. **(True and True) and (True == False)**
6. **(not False) or (not True)**

**Ans***: Let's evaluate the values of the given expressions step by step:*

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

* *`5 > 4` is `True`.*
* *`3 == 5` is `False`.*
* *The `and` operator returns `True` if both operands are `True`, otherwise, it returns `False`.*
* *So, this expression evaluates to `True and False`, which is `False`.*

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

* + *`5 > 4` is `True`.*
  + *The `not` operator negates the Boolean value, so it becomes `not True`, which is `False`.*

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

* + *`5 > 4` is `True`.*
  + *`3 == 5` is `False`.*
  + *The `or` operator returns `True` if at least one operand is `True`, so this expression evaluates to `True or False`, which is `True`.*

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

* + *We already evaluated `(5 > 4) or (3 == 5)` as `True`.*
  + *The `not` operator negates the result, so it becomes `not True`, which is `False`.*

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

* + *`True and True` is `True`.*
  + *`True == False` is `False`.*
  + *The `and` operator returns `True` if both operands are `True`, so this expression evaluates to `True and False`, which is `False`.*

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

* + *`Not False` is `True`.*
  + *`Not True` is `False`.*
  + *The `or` operator returns `True` if at least one operand is `True`, so this expression evaluates to `True or False`, which is `True`.*

*So, the values of the given expressions are as follows:*

*1. `False`*

*2. `False`*

*3. `True`*

*4. `False`*

*5. `False`*

*6. `True`*

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

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

*1. Equal to (“==”): Checks if two values are equal.*

*Example: “5 == 5” returns” True”.*

*2. Not equal to (“! =”): Checks if two values are not equal.*

*Example: “5!= 3” returns “True”.*

*3. Greater than (“>”): Check if the value on the left is greater than the value on the right. Example: “7 > 3” returns “True”.*

*4. Less than (“<”): Check if the value on the left is less than the value on the right. Example: `2 < 4` returns `True`.*

*5. Greater than or equal to (“>=”): Check if the value on the left is greater than or equal to the value on the right. Example: `5 >= 5` returns `True`.*

*6. Less than or equal to (“<=”): Check if the value on the left is less than or equal to the value on the right. Example: “3 <= 3” returns “True”.*

*These comparison operators are frequently used in conditional statements and expressions to make decisions or filter data based on the results of the comparisons.*

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

**Ans***: We can tell the difference between the equal to (`==`) operator and the assignment operator (`=`) in Python by their context and usage.*

*1. Equal to Operator (“==”):*

* *The equal to operator (`==`) is used for comparison.*
* *It checks if two values are equal and returns a Boolean result (`True` or `False`).*
* *It is used in conditional statements and expressions to compare values.*

*Example:*

*“”” #python*

*x = 5*

*y = 3*

*if x == y:*

*print ("x is equal to y")”””*

*In this example, “x == y” compares the values of `x` and `y` to determine if they are equal.*

*2. Assignment Operator (“=”):*

* *The assignment operator (`=`) is used for variable assignment.*
* *It assigns a value to a variable.*
* *It does not compare values; instead, it assigns the value on the right side to the variable on the left side. Example:*

*```#python*

*x = 5 ```*

*In this example, `x = 5` assigns the value `5` to the variable `x`.*

*When to Use Each Operator:*

* *Use the equal to the operator (`==`) when you want to compare values to make decisions or perform conditional operations. For example, when you want to check if two variables have the same value before taking a certain action.*
* *Use the assignment operator (`=`) when you want to assign a value to a variable. This is commonly used to initialize variables or update their values as your program runs.*
* *It's important to be cautious about not confusing these operators, as mistaking `=` for `==` can lead to unintended consequences or errors 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')**

**Ans:** *In Python, blocks of code are defined by their indentation levels. Typically, code blocks are indented with a consistent number of spaces or tabs. In this provided code, there are three blocks. I'll identify them below:*

*1. Block 1:*

*“”” if spam == 10:*

*print('eggs')*

*“””*

*This block is associated with the first `if` statement and contains the `print('eggs’) ` statement. It is executed if the condition `spam == 10` is `True`*

*2. Block 2:*

*```if spam > 5:*

*print('bacon')*

*else:*

*print('ham')*

*```*

*This block starts with the second `if` statement and includes both the `print('bacon’) ` statement and the `print('ham’) ` statement within an `if-else` structure. It is executed based on the condition `spam > 5`. If `spam > 5` is `True`, it executes `print('bacon’) `; otherwise, it executes `print('ham')`.*

*3. Block 3:*

*```print('spam')*

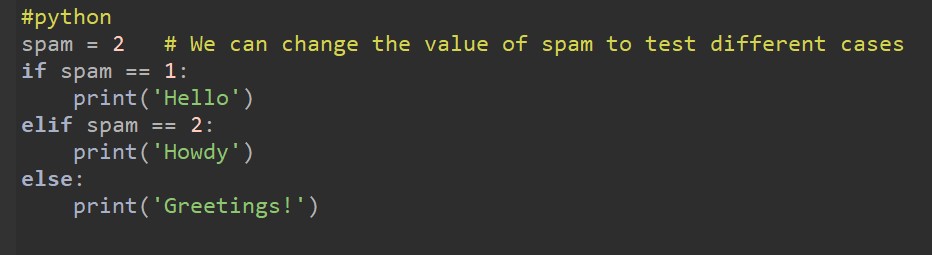
*print('spam') ```*

*This block consists of two `print` statements and is not governed by any `if` or `else` condition. It will always be executed regardless of the conditions in the previous blocks. These `print` statements are executed unconditionally.*

*It's important to note that in Python, indentation is crucial for code structure, and incorrect indentation can lead to syntax errors or unintended behaviour.*

**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**.

**Ans:** We *can achieve this using an `if- elif -else` structure in Python. Here's the code that accomplishes the task:*

**

*In this code:*

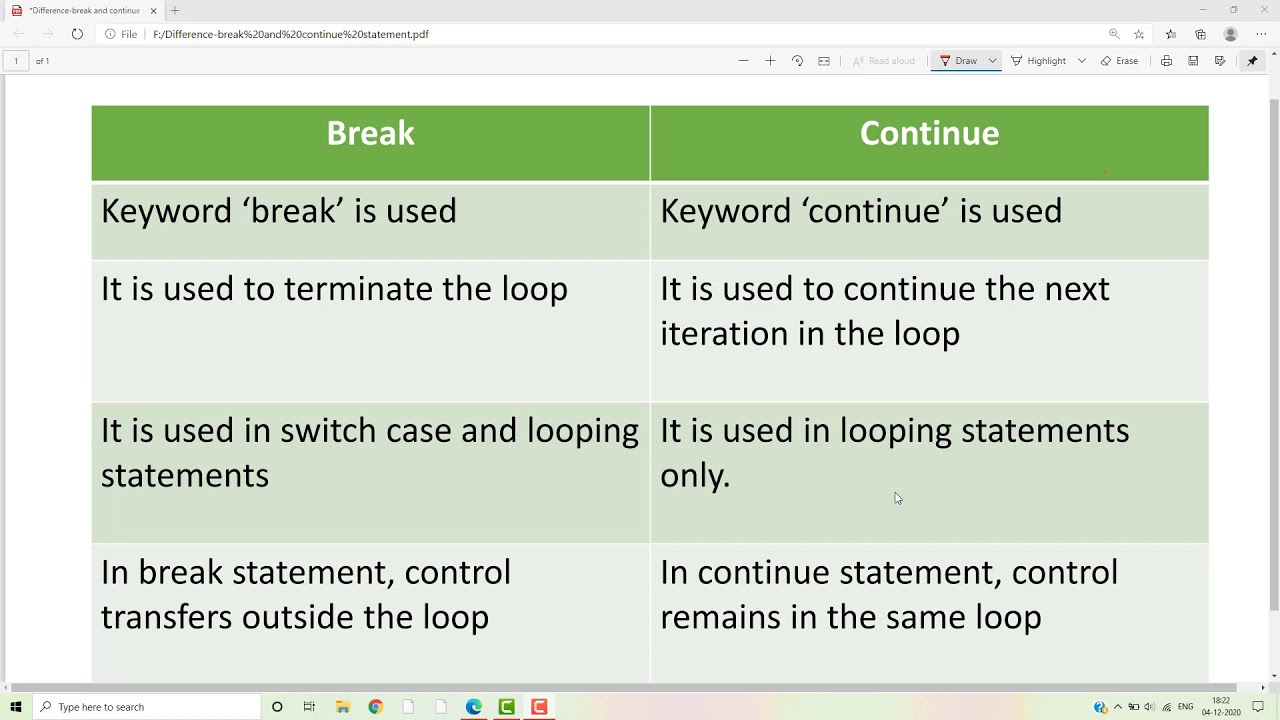
* + - *If `spam` is equal to `1`, it will print `'Hello'`.*
    - *If `spam` is equal to `2`, it will print `'Howdy'`.*
    - *If `spam` is any other value, it will print `'Greetings!'`.*

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

**Ans:** **Ctrl + C:** *This keyboard shortcut is commonly used to interrupt the execution of a program in a terminal or command prompt. When you press Ctrl + C, it sends a signal to the running program, causing it to terminate.*

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

**Ans:**

****

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

**Ans:** *In a for loop in Python, range (10), range (0, 10), and range (0, 10, 1) are three different ways to generate a sequence of numbers that the loop will iterate over. However, they are functionally equivalent, and there is no significant difference between them. Let's break down each of these expressions:*

**1***.“***range** *(10)”: This expression creates a sequence of numbers starting from 0 (the default start value) and ending at 9 (10 - 1, the specified end value).*

*It generates numbers from 0 to 9 in increments of 1 (the default step value).*

*Essentially, it represents the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.*

***2****. “***range** *(0, 10)”:*

*This expression explicitly specifies the start value as 0 and the end value as 9.*

*Like range (10), it generates numbers from 0 to 9 in increments of 1 (the default step value).*

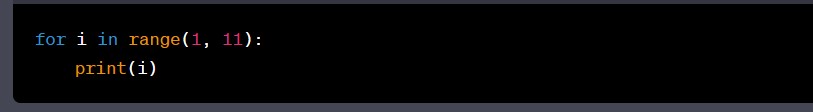
**3. range** *(0, 10, 1):*

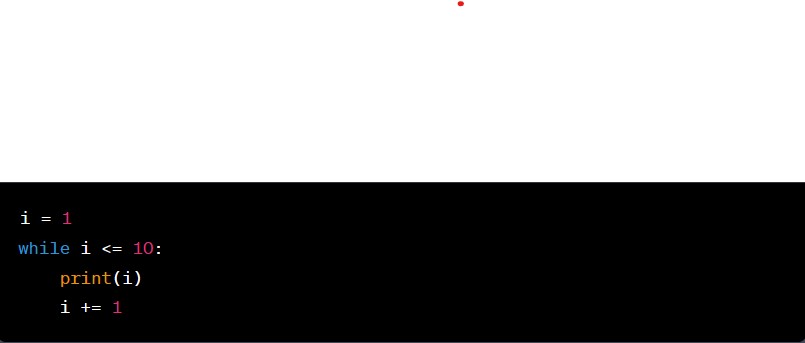
*This expression is similar to the range (0, 10) but explicitly specifies a step value of 1 (which is also the default step value).*

*It generates numbers from 0 to 9 in increments of 1.*

**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**.

*Ans:* *Here's a short program that prints the numbers 1 to 10 using a `for` loop:*

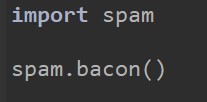
**

*And here's an equivalent program that prints the numbers 1 to 10 using a `while` loop:*

*Both of these programs will produce the same output, printing the numbers from 1 to 10. The `for` loop is often preferred when you have a definite range of values to iterate over, while the `while` loop is more flexible and can be used when you have a condition for termination that may not be based solely on a numeric range.*

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

**Ans:** *If you have a function named `bacon () ` inside a module named `spam`, you can call it after importing the `spam` module in the following way:*

**

*Here's what's happening:*

*1. You use the `import` statement to import the `spam` module into your Python script.*

*2. Once the `spam` module is imported, you can access its functions, classes, and variables using dot notation. In this case, you access the `bacon ()` function as `spam. bacon()`, where `spam` is the module name, and `bacon` is the function name.*

*This allows you to call the `bacon ()` function from the `spam` module in your script.*