**Q1.What are the two values of the Boolean data type? How do you write them?**

Ans - The two values of the Boolean data type in Python are `True` and `False`. They represent the two possible states of logical truth in Python.

In Python, you write the Boolean value `True` as `True` (with an uppercase 'T') and the Boolean value `False` as `False` (with an uppercase 'F'). These values are keywords in Python and are not enclosed in quotes.

**Q2. What are the three different types of Boolean operators?**

Ans - In Python, there are three different types of Boolean operators used for logical operations:

1. Logical AND (`and`): The `and` operator returns `True` if both operands are `True`, and `False` otherwise. It evaluates to `False` if at least one of the operands is `False`. Here's an

2. Logical OR (`or`): The `or` operator returns `True` if at least one of the operands is `True`, and `False` if both operands are `False`. It evaluates to `True` if any one of the operands is `True`. Here's an example:

3. Logical NOT (`not`): The `not` operator is a unary operator that negates the value of the operand. It returns `True` if the operand is `False`, and `False` if the operand is `True`. It essentially flips the Boolean value. Here's an example:

**Q3. Make a list of each Boolean operator&#39;s truth tables (i.e. every possible combination of Boolean**

**values for the operator and what it evaluate ).**

Certainly! Here are the truth tables for each Boolean operator:

1. Logical AND (`and`):

|  |  |  |
| --- | --- | --- |
| Operand 1 | Operand 2 | Result |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

1. Logical OR (`or`):

|  |  |  |
| --- | --- | --- |
| Operand 1 | Operand 2 | Result |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

1. Logical NOT (`not`):

|  |  |
| --- | --- |
| Operand | Result |
| True | False |
| False | True |

**Q4. What are the values of the following expressions?**

Ans -

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

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

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

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

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

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

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

Ans - In Python, there are six comparison operators used to compare values. These operators return a Boolean value (`True` or `False`) based on the comparison result. The six comparison operators in Python are:

1. Equal to (`==`): It checks if the values of two operands are equal.

2. Not equal to (`!=`): It checks if the values of two operands are not equal.

3. Greater than (`>`): It checks if the value of the left operand is greater than the value of the right operand.

4. Less than (`<`): It checks if the value of the left operand is less than the value of the right operand.

5. Greater than or equal to (`>=`): It checks if the value of the left operand is greater than or equal to the value of the right operand.

6. Less than or equal to (`<=`): It checks if the value of the left operand is less than or equal to the value of the right operand.

**Q6. How do you tell the difference between the equal to and assignment operators?Describe a**

**condition and when you would use one.**

Ans - In Python, the equal to (`==`) operator is used for comparison, while the assignment (`=`) operator is used for assigning values to variables.

The equal to (`==`) operator is used to compare two values and check if they are equal. It returns `True` if the values are equal and `False` otherwise. Here's an example:

```python

x = 5

y = 10

if x == y:

print("x is equal to y")

else:

print("x is not equal to y")

```

In this example, the `==` operator is used to compare the values of `x` and `y`. Since `x` and `y` have different values (`5` and `10`), the condition evaluates to `False`, and the output will be "x is not equal to y."

On the other hand, 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. Here's an example:

```python

x = 5

y = x

```

In this example, the assignment operator `=` is used to assign the value of `x` (which is `5`) to the variable `y`. After the assignment, `y` will have the same value as `x`.

To differentiate between the equal to (`==`) operator and the assignment (`=`) operator, remember that the equal to operator (`==`) is used for comparison and returns a Boolean value (`True` or `False`). The assignment operator (`=`) is used to assign values to variables.

You would use the equal to (`==`) operator when you want to compare values, such as in conditional statements or when checking equality between variables. On the other hand, you would use the assignment (`=`) operator when you want to assign a value to a variable or update its value.

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

The code snippet you provided is missing proper indentation, which is crucial in Python for defining blocks. However, based on the intended structure, I can identify the three blocks as follows:

Block 1:

```

if spam == 10:

print('eggs')

```

Block 2:

```

if spam > 5:

print('bacon')

```

Block 3:

```

else:

print('ham')

print('spam')

print('spam')

```

Note that the indentation level determines the boundaries of each block. In Python, indentation is typically done using four spaces or a tab character. Proper indentation helps define the structure and hierarchy of the code, especially within conditional statements and loops.

**Q.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 -

spam = 1

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

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

Ans - If your program is stuck in an endless loop and you want to interrupt or terminate it, you can typically press the following key combination:

On most systems: `Ctrl + C`

Pressing `Ctrl + C` sends an interrupt signal to the running program, causing it to terminate and return control back to the command line or the integrated development environment (IDE) where you executed the program.

**Q10. How can you tell the difference between break and continue?**

Ans - In Python, `break` and `continue` are two keywords used to alter the flow of control in loops. Here's how you can differentiate between them:

1. `break`: The `break` statement is used to exit the current loop entirely. When encountered, the `break` statement immediately terminates the loop and continues with the execution of the next statement outside the loop. It is often used to prematurely exit a loop based on certain conditions. Here's an example:

```

for i in range(1, 6):

if i == 3:

break

print(i)

```

Output:

```

1

2

```

In this example, the loop iterates over the range from 1 to 5. When `i` becomes 3, the `break` statement is encountered, and the loop is terminated. As a result, only the values 1 and 2 are printed.

2. `continue`: The `continue` statement is used to skip the remaining code within the current iteration of the loop and move on to the next iteration. When encountered, the `continue` statement skips the rest of the code in the loop body and proceeds with the next iteration, without executing any further statements in the current iteration. Here's an example:

```

for i in range(1, 6):

if i == 3:

continue

print(i)

```

Output:

```

1

2

4

5

```

In this example, when `i` becomes 3, the `continue` statement is encountered. As a result, the `print(i)` statement is skipped, and the loop moves on to the next iteration. Hence, the value 3 is not printed.

**Q.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 `range()` function is commonly used to generate a sequence of numbers. Let's examine the differences between `range(10)`, `range(0, 10)`, and `range(0, 10, 1)`:

1. `range(10)`: This form of `range()` specifies the stop value of the sequence, starting from zero. In this case, it generates numbers from 0 to 9 (10 numbers in total). The start value is implicitly set to 0, and the step value is implicitly set to 1.
2. `range(0, 10)`: This form of `range()` specifies both the start and stop values of the sequence. It generates numbers starting from the given start value (0) up to, but not including, the given stop value (10). The step value is again implicitly set to 1.

3. `range(0, 10, 1)`: This form of `range()` specifies the start, stop, and step values of the sequence. It generates numbers starting from the given start value (0), incrementing by the given step value (1), up to, but not including, the given stop value (10).

**Q.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 -

for loop -

for i in range(1, 11):

print(i)

while loop -

i = 1

while i <= 10:

print(i)

i += 1

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

**importing spam?**

Ans -

import spam

spam.bacon()