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

The two types of Boolean data types are: True and False.

We write them as below:

**True**: It represents the logical value of true.

**False**: It represents the logical value of false.

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

Thera are three different types of Boolean operators:

1. **and** operator: It is denoted by the keyword "and". It returns True if both operands are True; otherwise, it returns False.

2. **or** operator: It is denoted by the keyword "or". It returns True if at least one of the operands is True; otherwise, it returns False.

3. **not** operator: It is denoted by the keyword "not". It takes a single operand and negates its logical value. If the operand is True, the not operator returns False, and if the operand is False, it 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 ).

Here are the truth tables for each Boolean operator:

1. AND operator:

| Operand 1 | Operand 2 | Result |

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

| True | True | True |

| True | False | False |

| False | True | False |

| False | False | False |

2. OR operator:

| Operand 1 | Operand 2 | Result |

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

| True | True | True |

| True | False | True |

| False | True | True |

| False | False | False |

3. NOT operator:

| Operand | Result |

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

| True | False |

| False | True |

These truth tables represent all possible combinations of Boolean values for each operator and their corresponding evaluations.

4. What are the values of the following expressions?

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

not (5 > 4) : **False**

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

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

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

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

5. What are the six comparison operators?

The six comparison operators are as follows:

1. Equal to (==): This operator checks if two operands are equal and returns True if they are, and False otherwise.

2. Not equal to (!=): This operator checks if two operands are not equal and returns True if they are not, and False otherwise.

3. Greater than (>): This operator checks if the left operand is greater than the right operand and returns True if it is, and False otherwise.

4. Less than (<): This operator checks if the left operand is less than the right operand and returns True if it is, and False otherwise.

5. Greater than or equal to (>=): This operator checks if the left operand is greater than or equal to the right operand and returns True if it is, and False otherwise.

6. Less than or equal to (<=): This operator checks if the left operand is less than or equal to the right operand and returns True if it is, and False otherwise.

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 (==) is used for comparison and checks if two values are equal. It returns True if the values are equal, and False otherwise. On the other hand, the assignment operator (=) is used to assign a value to a variable.

**Condition usage:**

If you want to assign a value to a variable, you would use the assignment operator (=). For example: X = 5, y = 15

If you want to compare the values of two variables, you would use the equal to operator (==). For example: x == y (In this case, the condition x == y checks if the values of x and y are equal. It compares the values and returns True or False based on the comparison)

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')

In the given code, the indentation is missing, but I'll assume the standard indentation of four spaces for each block. Based on that assumption, the three blocks in the code can be identified as follows:

Block 1:

```python

spam = 0

if spam == 10:

print('eggs')

```This block consists of an `if` statement that checks if the variable `spam` is equal to 10. If it is, the code inside the block (printing 'eggs') will execute.

Block 2:

```python

if spam > 5:

print('bacon')

else:

print('ham')

```

This block also starts with an `if` statement. If the variable `spam` is greater than 5, the code inside the block (printing 'bacon') will execute. Otherwise, the code inside the `else` block (printing 'ham') will execute.

Block 3:

```python

print('spam')

print('spam')

```

This block consists of two consecutive `print` statements that will always execute regardless of the conditions in the previous blocks.

Keep in mind that the code provided is missing proper indentation, which is crucial in Python to define blocks of code. Indentation determines the grouping of statements in Python, so it's essential to ensure the code is correctly indented to reflect the intended block structure.

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 = 1

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?

I am using Windows and hence, I will press: **Ctrl + C** to interrupt its execution

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

In programming, "break" and "continue" are both control flow statements used in loops (such as "for" or "while") to alter the program's behaviour. However, they have distinct purposes and effects:

**Break:**

The "break" statement is used to exit the current loop immediately when a certain condition is met, regardless of whether the loop has completed all iterations or not.

When the "break" statement is encountered, the program flow jumps out of the loop entirely, and execution continues with the next statement after the loop.

**Continue:**

The "continue" statement is used to skip the rest of the current iteration and move to the next iteration of the loop.

When the "continue" statement is encountered, the program flow jumps to the next iteration, bypassing any remaining statements within the loop's body for that iteration.

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 differences between `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are as follows:

1. `range(10)`:

- This expression generates a sequence of numbers starting from 0 (default start value) up to, but not including, the specified end value, which is 10 in this case.

- It uses a step value of 1 (default step value), meaning it increments by 1 for each iteration.

- So, `range(10)` generates the sequence [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

2. `range(0, 10)`:

- This expression generates a sequence of numbers starting from the specified start value, which is 0 in this case, up to, but not including, the specified end value, which is 10.

- It also uses a step value of 1 (default step value).

- So, `range(0, 10)` generates the same sequence as `range(10)`: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

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

- This expression generates a sequence of numbers starting from the specified start value, which is 0, up to, but not including, the specified end value, which is 10.

- It explicitly specifies a step value of 1.

- Since 1 is the default step value, specifying it explicitly does not change the behaviour of the range.

- So, `range(0, 10, 1)` generates the same sequence as `range(0, 10)`: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

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.

**# Print 1 to 10 using 'for' loop:**

for i in range(1, 11):

print(i)

**# Print 1 to 10 using 'while' loop**

i =1

while True:

if i <= 10:

print(i)

i += 1

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

Spam.bacon()