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

The Boolean data type represents logical values that can have one of two possible states: True or False. These two values are the only valid values for the Boolean data type.

In Python, you write these Boolean values as True and False. It's important to note that the capitalization is significant, as Python treats True and False as keywords.

Here are examples of how you would use and assign Boolean values in Python:

is\_raining = True

is\_sunny = False

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

Logical AND (and): The logical AND operator returns True if both operands are True, and False otherwise. It evaluates to True only when both conditions are satisfied. In Python, the logical AND operator is represented by the keyword and

Logical OR (or): The logical OR operator returns True if at least one of the operands is True, and False if both operands are False. It evaluates to True when at least one condition is satisfied. In Python, the logical OR operator is represented by the keyword or

Logical NOT (not): The logical NOT operator reverses the Boolean value of an operand. If the operand is True, the operator returns False, and if the operand is False, the operator returns True. In Python, the logical NOT operator is represented by the keyword not

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

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)

1. False
2. False
3. True
4. False
5. False
6. True
7. What are the six comparison operators?

Equal to (==): This operator checks if two values are equal and returns True if they are, and False otherwise. For example: 5 == 5 would evaluate to True.

Not equal to (!=): This operator checks if two values are not equal and returns True if they are not, and False if they are equal. For example: 5 != 3 would evaluate to True.

Greater than (>): This operator checks if the value on the left is greater than the value on the right and returns True if it is, and False otherwise. For example: 5 > 3 would evaluate to True.

Less than (<): This operator checks if the value on the left is less than the value on the right and returns True if it is, and False otherwise. For example: 3 < 5 would evaluate to True.

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

Less than or equal to (<=): This operator checks if the value on the left is less than or equal to the value on the right and returns True if it is, and False otherwise. For example: 3 <= 5 would evaluate to True

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

To differentiate between the equal to (==) operator and the assignment (=) operator, consider the following:

1. Equal to (==) operator:
   * 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 if they are not.
   * For example: x == 5 compares the value of x to the number 5 and returns True if they are equal.
2. Assignment (=) operator:
   * The assignment operator (=) is used to assign a value to a variable.
   * It assigns the value on the right side of the operator to the variable on the left side.
   * For example: x = 5 assigns the value 5 to the variable x.

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')

The identified blocks are:

1. Block 1:
   * It consists of the line print('eggs').
   * This block is executed if the condition spam == 10 is evaluated to True.
2. Block 2:
   * It consists of the line print('bacon').
   * This block is executed if the condition spam > 5 is evaluated to True.
3. Block 3:
   * It consists of the line print('ham').
   * This block is executed if the condition spam > 5 is evaluated to False
4. 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?

1. **Ctrl + C**: Pressing the Ctrl key and the C key simultaneously interrupts the program's execution in most command-line environments. This key combination sends an interrupt signal to the program, causing it to terminate.
2. **Ctrl + Break**: On some systems or IDEs, pressing the Ctrl key and the Break key (usually located in the same area as the Pause key) can also be used to interrupt the program's execution.

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

break statement:

* When encountered within a loop (such as for or while), the break statement immediately terminates the loop's execution.
* It causes the program to exit the loop and continue executing the code that follows the loop.
* Once a break statement is encountered, the program does not go back to the beginning of the loop or evaluate any remaining iterations.
* break is typically used when a certain condition is met, and you want to exit the loop prematurely.

continue statement:

* When encountered within a loop, the continue statement skips the rest of the loop body for the current iteration and moves on to the next iteration.
* It does not terminate the loop but rather jumps back to the beginning of the loop for the next iteration.
* The program skips any remaining statements in the loop body and proceeds with the next iteration.
* continue is typically used when you want to skip certain iterations based on a specific condition.

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

range(10):

* This creates a sequence that starts from 0 (the default start value) and ends at 9 (10 - 1).
* The step value is implicitly set to 1 (the default step value).

range(0, 10):

* This creates a sequence that starts from 0 (the provided start value) and ends at 9 (10 - 1).
* The step value is implicitly set to 1 (the default step value).
* The start value is inclusive, meaning it is included in the sequence, while the end value is exclusive, meaning it is not included in the sequence.

range(0, 10, 1):

* This creates a sequence that starts from 0 (the provided start value), ends at 9 (10 - 1), and increments by 1 (the provided step value).
* The start value is inclusive, while the end value is exclusive.
* In this case, explicitly specifying a step value of 1 does not change the behavior because it is the default step value.

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.

Here are two short programs that print the numbers 1 to 10 using a for loop and a while loop respectively:

For for loop

for num in range(1, 11):

print(num)

for while

num = 1

while num <= 10:

print(num)

num += 1

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

The import statement imports the module spam. To call the function bacon() from the spam module, you use the syntax spam.bacon(), where spam is the name of the module and bacon() is the name of the function.

This syntax ensures that you are calling the bacon() function from within the spam module and avoids any naming conflicts with other functions or variables in your code.

import spam

spam.bacon()