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

The two values of the Boolean data type are "true" and "false". In programming, we usually write them in lowercase letters, as "true" and "false".Boolean values are often used in programming to represent the truth or falsity of a condition or statement. They are used in logical operations such as AND, OR, and NOT to determine the overall result of the operation based on the truth values of its operands.

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

1\_ AND operator: The AND operator is used to combine two or more Boolean expressions and returns true only if all expressions are true. The symbol used for the AND operator is "&&" (double ampersand).

2\_OR operator: The OR operator is used to combine two or more Boolean expressions and returns true if at least one expression is true. The symbol used for the OR operator is "||" (double vertical bars).

3\_ NOT operator: The NOT operator is used to reverse the Boolean value of an expression. If the expression is true, the NOT operator returns false, and if the expression is false, the NOT operator returns true. The symbol used for the NOT operator is "!" (exclamation mark).

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 ).

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)

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

Value: False

not (5 > 4)

Value: False

Ans : (5 > 4) or (3 == 5)

Value: True

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

Value: False

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

Value: False

(not False) or (not True)

Value: True

5. What are the six comparison operators?

The six comparison operators are:

Equal to (==): Returns True if the two operands are equal.

Not equal to (!=): Returns True if the two operands are not equal.

Greater than (>): Returns True if the left operand is greater than the right operand.

Less than (<): Returns True if the left operand is less than the right operand.

Greater than or equal to (>=): Returns True if the left operand is greater than or equal to the right operand.

Less than or equal to (<=): Returns True if the left operand is less than or equal to the right operand.

These operators are used to compare two values in a conditional statement and return a Boolean value (True or False) depending on the result of the comparison.

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 a comparison operator that checks whether two values are equal or not, whereas the assignment operator (=) is used to assign a value to a variable.The main difference between the two operators is that the equal to operator (==) is used in a conditional statement to compare two values and returns a Boolean value (True or False), while the assignment operator (=) is used to assign a value to a variable.

Here's an example of using the equal to operator:

x = 5

y = 7

if x == y:

print("x is equal to y")

else:

print("x is not equal to y")

In this example, we're comparing the values of x and y using the equal to operator (==) in the conditional statement. Since x is not equal to y, the output will be "x is not equal to y".

Here's an example of using the assignment operator:

x = 5

y = x

print(y)

In this example, we're using the assignment operator (=) to assign the value of x to y. When we print the value of y, it will be 5, which is the same value as x.So, to summarize, we use the equal to operator (==) to compare two values and the assignment operator (=) to assign a value to a variable.

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')

spam = 0

# Block 1: Conditional statement

if spam == 10:

print('eggs')

# Block 2: Conditional statement and else clause

if spam > 5:

print('bacon')

else:

print('ham')

# Block 3: Print statements

print('spam')

print('spam')

Block 1 is a conditional statement that checks whether the value of the variable spam is equal to 10. If it is, it prints the string "eggs".

Block 2 is also a conditional statement that checks whether the value of spam is greater than 5. If it is, it prints the string "bacon". If it's not greater than 5, it goes to the else clause and prints the string "ham".

Block 3 consists of two print statements that print the string "spam" twice.

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.

Here's the code that prints "Hello" if 1 is stored in spam, "Howdy" if 2 is stored in spam, and "Greetings!" if anything else is stored in spam:

spam = 3

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

In this code, we first assign a value of 3 to the variable spam. Then we use an if-elif-else statement to check the value of spam. If the value is 1, we print "Hello". If the value is 2, we print "Howdy". Otherwise, we print "Greetings!". Since the value of spam is 3 in this example, the output will be "Greetings!".

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

If the program is stuck in an endless loop, we can use the following keys to stop the program:

Ctrl + C: This is a common keyboard shortcut used to interrupt a running process in the command line interface. It sends a "Keyboard Interrupt" signal to the program, causing it to terminate.

Ctrl + Break: This is another keyboard shortcut that can be used to stop a running process in the command line interface. On some systems, it may be labeled as "Pause" or "Pause/Break".

Ctrl + Alt + Del: On Windows systems, this keyboard shortcut brings up the Task Manager, which can be used to force stop a running process.

It's important to note that forcibly stopping a running process can sometimes cause data loss or other issues, so it's recommended to only do this as a last resort. It's usually better to try and find the source of the endless loop and fix the issue in the code.

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

break and continue are two control flow statements in Python that are used to alter the normal flow of execution of a loop. The main difference between them is:

break is used to immediately terminate a loop (for loop, while loop or nested loop) when a certain condition is met. When break is encountered in a loop, the loop is exited and the control flow continues to the next statement after the loop.

continue is used to skip the current iteration of a loop and move to the next iteration, without executing the remaining code in the current iteration. When continue is encountered in a loop, the current iteration is skipped and the control flow jumps to the next iteration.

Here's an example to illustrate the difference:

# Using break in a for loop

for i in range(5):

if i == 3:

break

print(i)

# Output: 0 1 2

# Using continue in a for loop

for i in range(5):

if i == 3:

continue

print(i)

# Output: 0 1 2 4

In the first example, we use break in a for loop to terminate the loop when i is equal to 3. When i is equal to 3, the loop is exited and the output is 0 1 2, since the remaining iterations are not executed.

In the second example, we use continue in a for loop to skip the iteration when i is equal to 3. When i is equal to 3, the current iteration is skipped and the output is 0 1 2 4, since the remaining iterations are executed after skipping the current iteration.

So, to summarize, break is used to terminate a loop immediately when a certain condition is met, while continue is used to skip the current iteration of a loop and move to the next 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 range() function is used to generate a sequence of numbers. The range() function can take up to three arguments: start, stop, and step.

Here are the differences between range(10), range(0, 10), and range(0, 10, 1):

range(10): This generates a sequence of numbers from 0 up to (but not including) 10, with a default step size of 1. In other words, it is equivalent to range(0, 10, 1).

Example:

for i in range(10):

print(i)

# Output: 0 1 2 3 4 5 6 7 8 9

range(0, 10): This generates a sequence of numbers from 0 up to (but not including) 10, with a step size of 1.

Example:

for i in range(0, 10):

print(i)

# Output: 0 1 2 3 4 5 6 7 8 9

range(0, 10, 1): This generates a sequence of numbers from 0 up to (but not including) 10, with a step size of 1. This is equivalent to range(10).

Example:

for i in range(0, 10, 1):

print(i)

# Output: 0 1 2 3 4 5 6 7 8 9

In summary, the differences between range(10), range(0, 10), and range(0, 10, 1) are just in the way they are specified. All three generate the same sequence of numbers.

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

for i in range(1, 11):

print(i)

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

i = 1

while i <= 10:

print(i)

i += 1

Both programs produce the same output:

1

2

3

4

5

6

7

8

9

10

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

If a function named bacon() is inside a module named spam, we can call it after importing spam using the dot notation. Here's an example:

import spam

spam.bacon()

In the example , we import the spam module using the import statement, and then we call the bacon() function using the dot notation (spam.bacon()). This tells Python to look for the bacon() function inside the spam module.