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. These values are used to represent a logical state, and can be used in conditional statements and other operations that require a binary decision.

In most programming languages, you can write these values by simply typing the words True and False (with an uppercase "T" and "F"). For example:

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

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

The three different types of Boolean operators are:

1. AND (represented by the symbol & or "&&" in programming) - This operator returns "true" if both operands are true, and returns "false" otherwise.
2. OR (represented by the symbol | or "||" in programming) - This operator returns "true" if either operand is true, and returns "false" if both operands are false.

NOT (represented by the symbol ! or "!" in programming) - This operator returns "true" if the operand is false, and returns "false" if the operand is true. The NOT operator is also known as the negation operator.

3. 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 ).

Sol: Here are the truth tables for the three basic Boolean operators: AND, OR, and NOT.

1. AND (**&**) operator:

| **p** | **q** | **p & q** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* The AND operator returns **True** only if both **p** and **q** are **True**.

1. OR (**\|**) operator:

| **p** | **q** | **p | q** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* The OR operator returns **True** if either **p** or **q** is **True**.

1. NOT (**!**) operator:

| **p** | **!p** |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

* The NOT operator returns the opposite of the Boolean value of **p**. If **p** is **True**, **!p** returns **False**, and vice versa.

4. What are the values of the following expressions?

(5 &gt; 4) and (3 == 5)

not (5 &gt; 4)

(5 &gt; 4) or (3 == 5)

not ((5 &gt; 4) or (3 == 5))

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

(not False) or (not True)

1. Sol: The value of the expression (5 > 4) and (3 == 5) is False because while the first part of the expression (5 > 4) is True, the second part (3 == 5) is False, and the "and" operator only evaluates to True if both operands are True.
2. The value of the expression not (5 > 4) is False because the negation of the expression (5 > 4), which is True, is False.
3. The value of the expression (5 > 4) or (3 == 5) is True because at least one of the operands is True (the first one, (5 > 4)). The "or" operator only evaluates to False if both operands are False.
4. The value of the expression not ((5 > 4) or (3 == 5)) is False because the expression (5 > 4) or (3 == 5) is True, and the negation of True is False.
5. The value of the expression (True and True) and (True == False) is False because while the first part of the expression (True and True) is True, the second part (True == False) is False, and the "and" operator only evaluates to True if both operands are True.
6. The value of the expression (not False) or (not True) is True because the first operand (not False) is True and the second operand (not True) is False, and the "or" operator only evaluates to False if both operands are False.
7. What are the six comparison operators?

Sol: The six comparison operators in Python are:

1. Less than: **<**
2. Less than or equal to: **<=**
3. Equal to: **==**
4. Not equal to: **!=**
5. Greater than or equal to: **>=**
6. Greater than: **>**

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

condition and when you would use one.

Sol:: The difference between the equal to (comparison) operator **==** and the assignment operator **=** is that the equal to operator compares two values and evaluates to True if they are equal, while the assignment operator assigns a value to a variable.

Here's an example of a condition and when you would use each operator:

If you want to check if a variable has the same value as a certain constant, you would use the equal to operator:

7. Identify the three blocks in this code:

spam = 0

if spam == 10:

print(&#39;eggs&#39;)

if spam &gt; 5:

print(&#39;bacon&#39;)

else:

print(&#39;ham&#39;)

print(&#39;spam&#39;)

print(&#39;spam&#39;)

sol: The three blocks in this code are:

1. The first block is the assignment of **0** to the variable **spam**.
2. The second block is the **if** statement that checks if **spam** is equal to **10**. If the condition is True, the message **'eggs'** would be printed, but since **spam** is equal to **0**, the block is skipped.
3. The third block is the **if**-**else** statement that checks if **spam** is greater than **5**. Since **spam** is not greater than **5**, the message **'ham'** is printed.

The remaining three **print** statements each print the message **'spam'**.

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.

Sol:

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?

Ssol:

If a program is stuck in an endless loop, you can usually stop it by pressing the **CTRL** + **C** keys on your keyboard. This combination is called a keyboard interrupt and it is a way to stop the program execution in many operating systems.

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

Sol:

**break** and **continue** are two important control flow statements in programming languages, such as Python, used to control the flow of execution in loops. The main difference between the two is as follows:

* **break**: **break** is used to exit a loop completely when a certain condition is met. When the **break** statement is executed, the loop terminates and the program continues with the next statement after the loop.
* **continue**: **continue** is used to skip the current iteration of a loop and move on to the next iteration. When the **continue** statement is executed, the program skips the rest of the statements in the current iteration and continues with the next iteration.

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

Sol: The **range** function is used in Python to generate a sequence of numbers. In a **for** loop, the **range** function can be used to specify the number of iterations for the loop.

The difference between **range(10)**, **range(0, 10)**, and **range(0, 10, 1)** is as follows:

* **range(10)**: This generates a sequence of numbers starting from **0** up to but not including **10**. This is equivalent to **range(0, 10)**.
* **range(0, 10)**: This generates a sequence of numbers starting from **0** up to but not including **10**. The first argument **0** is the starting number and the second argument **10** is the ending number.
* **range(0, 10, 1)**: This generates a sequence of numbers starting from **0** up to but not including **10** with a step of **1**. The first argument **0** is the starting number, the second argument **10** is the ending number, and the third argument **1** is the step.

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.

Sol:

Here is the program that prints the numbers 1 to 10 using a for loop:

for i in range(1, 11):

print(i)

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

i = 1

while 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?

Sol:

You would call the **bacon()** function from the **spam** module like this:

import spam

spam.bacon()