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

**Answer:** The Boolean data type represents two values: **True** and **False**. In Python, these values are written as **True** and **False**, respectively.

We can write Boolean values using the keywords **True** and **False**.

For example:

**x = True**

**y = False**

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

**Answer:** There are 3 boolean operators as follows:

1. **and**: This operator returns **True** if both operands are **True**, and **False**
2. **or**: This operator returns **True** if at least one of the operands is **True**, and **False**
3. **not**: This operator returns the opposite of the operand. If the operand is **True**, it returns **False**, and vice versa.

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 evaluates).

**Answer**: Here are the truth tables for the three Boolean operators:

1. **and**:

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

1. **or**:

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

**3.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)

**Answer:** Here are the values of the given expressions:

1. **(5 > 4) and (3 == 5)** evaluates to **False** because the second operand **(3 == 5)** is **False**.
2. **not (5 > 4)** evaluates to **False** because **5 > 4** is **True** and **not True** is **False**.
3. **(5 > 4) or (3 == 5)** evaluates to **True** because the first operand **(5 > 4)** is **True**.
4. **not ((5 > 4) or (3 == 5))** evaluates to **False** because the inner expression **(5 > 4) or (3 == 5)** evaluates to **True**, and **not True** is **False**.
5. **(True and True) and (True == False)** evaluates to **False** because the second operand **(True == False)** is **False**.
6. **(not False) or (not True)** evaluates to **True** because the first operand **(not False)** is **True**.

5.What are the six comparison operators?

**Answer:** Here are the six comparison operators in Python:

1. **==** (equal to)
2. **!=** (not equal to)
3. **>** (greater than)
4. **<** (less than)
5. **>=** (greater than or equal to)
6. **<=** (less than or equal to)

These operators are used to compare two values and evaluate to either **True** or **False** depending on the relationship between the values. For example, **5 > 3** evaluates to **True**, while **5 < 3** evaluates to **False**.

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

**Answer**: The equal to operator **==** is used to compare two values for equality, while the assignment operator **=** is used to assign a value to a variable.

Here's an example to illustrate the difference between the two:

x = 5 # assigns the value 5 to the variable x

y = 7 # assigns the value 7 to the variable y

if x == y: #here it denotes comparison

print("x and y are equal")

else:

print("x and y are not equal")

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')

**Answer**:

**# Block 1: setting the initial value of spam to 0**

spam = 0

**# Block 2: conditional statement to check if spam is equal to 10**

if spam == 10:

print('eggs')

**# Block 3: conditional statement to check if spam is greater than 5**

if spam > 5:

print('bacon')

else:

print('ham')

**# Block 4: printing the value 'spam'**

print('spam')

**# Block 5: printing the value 'spam'**

print('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.

**Anwer: spam = 3**

**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?

**Answer**: If a program is stuck in an endless loop, you can press "Ctrl + C" on Windows or "Command + C" on a Mac to stop the program. This will send a keyboard interrupt signal to the program and force it to stop running.

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

**Answer**: **break** and **continue** are two keywords used in loops in Python.

**break** is used to exit a loop immediately, regardless of whether the loop has finished iterating over all items in the sequence or not. When **break** is executed inside a loop, the loop is terminated and the program execution continues with the next statement after the loop.

**continue**, on the other hand, is used to skip over a particular iteration of a loop. When **continue** is executed inside a loop, the current iteration is stopped, and the program execution continues with the next iteration of the loop.

In other words, **break** is used to exit the loop altogether, while **continue** is used to skip over specific iterations of the loop.

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

**Answer**: Differences between **range(10)**, **range(0, 10)**, and **range(0, 10, 1) are as follows**:

* **range(10)**: This generates a sequence of numbers from 0 up to (but not including) 10. The default **start** value is 0 and the default **step** value is 1, so this is equivalent to **range(0, 10, 1)**.
* **range(0, 10)**: This generates a sequence of numbers from 0 up to (but not including) 10. The **start** value is 0 and the **step** value is 1.
* **range(0, 10, 1)**: This generates a sequence of numbers from 0 up to (but not including) 10, with a **start** value of 0, an **end** value of 10, and a **step** value of 1. This is equivalent to **range(10)**.

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.

**Answer:**

**Using For Loop:**

for i in range(1, 11):

print(i)

**Using 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?

**Answer**: If the module **spam** containing the function **bacon()** is imported in the current program, you can call the function using the syntax **spam.bacon()** where **spam** is the module name and **bacon()** is the function name.E.g.

import spam

# calling bacon function from spam module

spam.bacon()