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

**Answer : Python boolean** type is one of the built-in data types provided by Python, which represents one of the two values i.e. True or False. Generally, it is used to represent the truth values of the expressions. For example, 1==1 is True whereas 2<1 is False.

**Example:** a = True

type(a)

b = False

type(b)

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

**Answer:** In Python, there are three main types of Boolean operators:

1. Logical AND (`and`):This operator returns `True` if both operands are `True`, otherwise, it returns `False`.

2. Logical OR (`or`): This operator returns `True` if at least one of the operands is `True`, and `False` if both operands are `False`.

3. Logical NOT (`not`): This operator is a unary operator that returns the opposite of the operand's truth value. If the operand is `True`, `not` returns `False`, and if the operand is `False`, `not` returns `True`.

These operators are often used to combine or modify boolean values in conditional statements and expressions.

**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 ).**

**Answer**: Certainly, here are the truth tables for each Boolean operator in Python:

**1. Logical AND (`and`):**

Operand 1 Operand 2 - Result

False False -False

False True -False

True False -False

True True -True

**2. Logical OR (`or`):**

Operand Operand - Result

False False - False

False True - True

True False -True

True True - True

**3. Logical NOT (`not`):**

Operand - Result

False - True

True - False

These truth tables illustrate the possible outcomes when applying each Boolean operator to different combinations of Boolean values.

**4. What are the values of the following expressions?**

Let's evaluate the values of the given expressions step by step:

1. (5 > 4) and (3 == 5) : (5 > 4) is True (3 == 5) is False

True and False is **False**

2. not (5 > 4) : (5 > 4) is True

not True **is False**

3. (5 > 4) or (3 == 5): (5 > 4) is True (3 == 5) is False

- True or False is **True**

4. not ((5 > 4) or (3 == 5)) : (5 > 4) is True (3 == 5) is False

- True or False is True

- not True is **False**

5. (True and True) and (True == False): True and True is True True == False is False

- True and False is **False**

6. (not False) or (not True) : not False is True not True is False

- True or False is **True**

**5. What are the six comparison operators?**

**Answer:**

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:`<=`

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

**Answer:**

**1. Equal To Operator (`==`):**

The equal to operator is used for comparison. It checks whether two values are equal or not. It returns `True` if the values are equal and `False` otherwise.

**2. Assignment Operator (`=`):**

The assignment operator is used to assign a value to a variable. It does not check for equality; rather, it assigns the value on the right-hand side to the variable on the left-hand side.

**Condition and Usage:**

num1 = 8

num2 = 12

if (num1 % 2 == 0 and num2 % 2 == 0):

print("Both numbers are even.")

else:

print("At least one of the numbers is not even.")

In this example, the condition using the equal to operator (`==`) checks if both `num1` and `num2` are even by checking if their remainders when divided by 2 are equal to 0. If both conditions are true, the program prints "Both numbers are even."

**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:**

**1. Block 1:**

if spam == 10:

print('eggs')

**2. Block 2:**

if spam > 5:

print('bacon')

**3. Block 3:**

else:

print('ham')

**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 = int(input("Enter a value for spam: ")) # You can replace this line with direct assignment

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**: press ctrl+C

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

- `break` is used to exit the loop entirely.

- `continue` is used to skip the current iteration and proceed to the next iteration of the loop.

**1. break:**

The `break` statement is used to immediately terminate the innermost loop (the loop that contains the `break` statement) and continue executing the code after the loop.

**Example:**

for i in range(5):

if i == 3:

break

print(i)

**2. continue:**

The `continue` statement is used to skip the remaining code inside the current iteration of a loop and move to the next iteration.

**Example:**

for i in range(5):

if i == 2:

continue

print(i)

**Both `break` and `continue` can help you control the behavior of loops and manage their execution based on specific conditions.**

**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 expressions `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are equivalent and will produce the same results. They generate a sequence of numbers from 0 to 9 (inclusive) with a step size of 1.

**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 a `for` loop to print the numbers from 1 to 10:**

for i in range(1, 11):

print(i)

**using a `while` loop to achieve the same result:**

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?**

import spam

spam.bacon()