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

ANS :- There are two types of Boolean data type in python which are true and false usually we write them as True and False in the expression to check a particular condition.

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

ANS :- In Python, there are three different types of Boolean operators: "and," "or," and "not." These operators are used to perform logical operations on Boolean values or 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 ).

ANS :- Given below are the truth tables for each Boolean operator:

1. \*\*and\*\* (logical conjunction):

| Operand 1 | Operand 2 | Result |

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

| False | False | False |

| False | True | False |

| True | False | False |

| True | True | True |

2. \*\*or\*\* (logical disjunction):

| Operand 1 | Operand 2 | Result |

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

| False | False | False |

| False | True | True |

| True | False | True |

| True | True | True |

3. \*\*not\*\* (logical negation):

| Operand | Result |

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

| False | True |

| True | False |

These truth tables illustrate the outcome of each Boolean operator when applied to different combinations of Boolean values (True or False).

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)

ANS :- The given expressions solved one by one are as follows :

1. `(5 > 4) and (3 == 5)`

- `(5 > 4)` is True.

- `(3 == 5)` is False.

- Therefore, the expression evaluates to `True and False`, which is False.

2. `not (5 > 4)`

- `(5 > 4)` is True.

- Applying the `not` operator, the expression becomes `not True`, which is False.

3. `(5 > 4) or (3 == 5)`

- `(5 > 4)` is True.

- `(3 == 5)` is False.

- Therefore, the expression evaluates to `True or False`, which is True.

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

- `(5 > 4) or (3 == 5)` is True.

- Applying the `not` operator, the expression becomes `not True`, which is False.

5. `(True and True) and (True == False)`

- `(True and True)` is True.

- `(True == False)` is False.

- Therefore, the expression evaluates to `True and False`, which is False.

6. `(not False) or (not True)`

- `not False` is True.

- `not True` is False.

- Therefore, the expression evaluates to `True or False`, which is True.

The values of the given expressions are:

1. False

2. False

3. True

4. False

5. False

6. True

5. What are the six comparison operators?

ANS :- The six comparison operators are:

1. Equal to (==): Compares if two values are equal and returns True if they are, False otherwise.

2. Not equal to (!=): Checks if two values are not equal and returns True if they are not equal, False otherwise.

3. Greater than (>): Determines if the left operand is greater than the right operand. It returns True if it is, False otherwise.

4. Less than (<): Checks if the left operand is less than the right operand. It returns True if it is, False otherwise.

5. Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right operand. It returns True if it is, False otherwise.

6. Less than or equal to (<=): Checks if the left operand is less than or equal to the right operand. It returns True if it is, False otherwise.

These comparison operators are commonly used to compare values and create conditions in programming languages to control the flow of the program based on the results of these comparisons.

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

ANS :- The equal to operator (`==`) is used for comparison, whereas the assignment operator (`=`) is used for assigning a value to a variable.

To differentiate between the equal to and assignment operators, consider the following:

1. Equal to Operator (`==`):

- The equal to operator compares two values for equality.

- It returns `True` if the values are equal and `False` if they are not equal.

- Example: `x == y` compares the values of `x` and `y` and returns `True` if they are equal.

2. Assignment Operator (`=`):

- The assignment operator assigns a value to a variable.

- It assigns the value on the right-hand side to the variable on the left-hand side.

- Example: `x = 5` assigns the value `5` to the variable `x`.

Here's a condition where you would use each operator:

Condition Example: Suppose you want to check if a given number is even.

num = 6

if num % 2 == 0: # Here, the '==' operator is used to compare num % 2 with 0.

print("The number is even.")

else:

print("The number is odd."

In this condition, the equal to operator (`==`) is used to compare the result of `num % 2` with `0`. If the condition evaluates to `True`, it means the number is even, and the corresponding message is printed.

The assignment operator (`=`) is not used in this condition. Instead, the value `6` is assigned to the variable `num` beforehand using the assignment operator.

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')

ANS :- The code provided does not have proper indentation, making it difficult to identify the blocks. However, based on the given code snippet, we can infer the intended blocks by assuming the proper indentation. Here are the three identified blocks:

1. Block 1:

```python

spam = 0

if spam == 10:

print('eggs')

```

This block consists of an `if` statement that checks if the variable `spam` is equal to 10. If the condition is true, it would execute the statement `print('eggs')`. However, since `spam` is assigned a value of 0, the condition evaluates to false, and the `print('eggs')` statement will not be executed.

2. Block 2:

```python

if spam > 5:

print('bacon')

else:

print('ham')

```

This block also includes an `if` statement. It checks if the variable `spam` is greater than 5. If the condition is true, it executes the statement `print('bacon')`. However, since `spam` has a value of 0, the condition evaluates to false. Therefore, the `else` block is executed, and it prints 'ham'.

3. Block 3:

```python

print('spam')

print('spam')

```

This block is not inside any conditional statement and consists of two `print` statements that will be executed regardless of the conditions. It will print '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.

ANS :- spam = int(input(“Please enter the value : “))

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?

ANS :- If your program is stuck in an endless loop and you need to interrupt or terminate it, you can typically press the following key combination:

- \*\*Ctrl + C\*\* (or \*\*Command + C\*\* on macOS): This key combination sends an interrupt signal to the running program, forcing it to terminate. It is a common way to stop the execution of a program that is stuck or running indefinitely.

By pressing Ctrl + C (or Command + C), the program should be interrupted, and you will regain control of the terminal or command prompt. However, please note that this key combination might vary depending on the operating system and the specific development environment you are using.

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

ANS :- The `break` and `continue` statements are used in control flow structures, such as loops, to alter the program's flow to differentiate between `break` and `continue`:

1. break:

- The `break` statement is used to exit or terminate a loop prematurely.

- When the `break` statement is encountered within a loop (e.g., `for` or `while`), the loop is immediately terminated, and the program execution continues with the statement following the loop.

- It helps in exiting the loop early based on certain conditions.

2. continue:

- The `continue` statement is used to skip the remaining code within a loop iteration and move to the next iteration.

- When the `continue` statement is encountered within a loop, the program jumps to the next iteration without executing the remaining statements within the loop for that particular iteration.

- It helps in bypassing specific iterations and proceeding to the next iteration.

To summarize, `break` is used to terminate the loop entirely, while `continue` is used to skip the remaining code within the current iteration and move on to the next iteration of the loop.

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

ANS :- Here's a summary of the difference between `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` in a `for` loop:

- `range(10)`: Generates a sequence of numbers from 0 to 9 (10 numbers in total).

- `range(0, 10)`: Generates a sequence of numbers from 0 to 9 (10 numbers in total), explicitly specifying the start and end values.

- `range(0, 10, 1)`: Generates a sequence of numbers from 0 to 9 (10 numbers in total), explicitly specifying the start, end, and step values. The step value of 1 means that each subsequent number in the sequence is incremented by 1.

All three forms of `range()` would allow you to iterate over the same set of numbers in a `for` loop, starting from 0 and ending at 9.

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.

ANS :- FOR LOOP :

for in range(1,11):

print(i)

WHILE LOOP :

i = 1

while ( i < 11):

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?

ANS :- To call the `bacon()` function from the `spam` module after importing it, you would use the following syntax:

```python

import spam

spam.bacon()

```

In this example, the `spam` module is imported using the `import` statement. After importing, you can access the `bacon()` function using the module name (`spam`) followed by a dot (`.`) and the function name (`bacon()`). This syntax allows you to call the `bacon()` function from the `spam` module.