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

Sol: Two values of Boolean data type are **True** and **False**

most of the time it use for condition evaluation and deciding the control flow.

Ex1:

```
while True:
    # Some statements/ expressions
```

Ex2:

```
If False:
    # Some statements/ expressions
else:
    # some statements/ expressions
```

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

Sol: The three different types of Boolean operators are:

1. AND Operator: The AND operator is used to combine two or more conditions or expressions. It returns true only if all the conditions or expressions being evaluated are true.

2. OR Operator: The OR operator is used to combine two or more conditions or expressions. It returns true if at least one of the conditions or expressions being evaluated is true.

3. NOT Operator: The NOT operator is used to negate or reverse the logical state of a condition or expression. It returns true if the condition or expression is false, and false if the condition or expression is true.

These Boolean operators are commonly used in programming and logic to evaluate conditions, perform comparisons, and control the flow of a program based on the resulting Boolean values.

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).

Sol:

1. AND Operator:
   We get the result True if and only if all operands are True.

Operand1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

2. OR Operator:
   We get the result True if any one the operand True

Operand1	Operand 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

3. NOT Operator:
   We get the opposite Boolean value as result

Operand	Result
True	False
False	True

These truth tables outline the possible combinations of Boolean values for each operator and the resulting evaluation.

4. What are the values of the following expressions?

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

Sol: False

not (5 > 4)

Sol: False

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

Sol: True

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

Sol: False

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

Sol: False

(not False) or (not True)

Sol: True

5. What are the six comparison operators?

Sol: The six comparison operators are:

1. Equal to (==): This operator checks if the operands on both sides are equal and returns true if they are, and false otherwise.

2. Not equal to (!=): This operator checks if the operands on both sides are not equal and returns true if they are not, and false if they are equal.

3. Greater than (>): This operator checks if the operand on the left side is greater than the operand on the right side and returns true if it is, and false otherwise.

4. Less than (<): This operator checks if the operand on the left side is less than the operand on the right side and returns true if it is, and false otherwise.

5. Greater than or equal to (>=): This operator checks if the operand on the left side is greater than or equal to the operand on the right side and returns true if it is, and false otherwise.

6. Less than or equal to (<=): This operator checks if the operand on the left side is less than or equal to the operand on the right side and returns true if it is, and false otherwise.

These comparison operators are commonly used in programming and logic to compare values and make decisions 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.

Sol: The equal to operator (==) and assignment operator (=) serve different purposes and have distinct functionalities. Here's how you can differentiate between them:

1. Equal to operator (==):

The equal to operator is used to compare the equality of two values. It checks if the values on both sides of the operator are equal and returns a Boolean value (true or false) based on the result of the comparison.

Ex:

```
x = 5

y = 7

if x == y:

    print("x is equal to y")

else:

    print("x is not equal to y")
```

In this example, the equal to operator (==) is used to compare the values of variables `x` and `y`. If they are equal, it will print "x is equal to y"; otherwise, it will print "x is not equal to y".

2. Assignment operator (=):

The assignment operator is used to assign a value to a variable. It takes the value on the right side and assigns it to the variable on the left side.

Ex:

```
x = 5

y = x + 3

print(y)
```

In this example, the assignment operator (=) is used to assign the value of `x + 3` to the variable `y`. The value of `x + 3` (which is 8) is assigned to `y`, and when we print `y`, it will output 8.

It's important to note that the assignment operator (=) is used to assign values to variables, while the equal to operator (==) is used for comparing the equality of two values.

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')

Sol: *Block 1*

```
spam = 0
if spam == 10:
    print('eggs')
```

Block 2

```
if spam > 5:
    print('bacon')
else:
    print('ham')
```

Block 3

```
print('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.

Sol:

```
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?

Sol: "Ctrl" + "C"

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

Sol: The "break" and "continue" statements are used in control flow within loops in programming languages like Python. Here's how you can differentiate between them:

1. "break" statement:

The "break" statement is used to exit the current loop entirely. When encountered, it immediately terminates the loop and the program execution continues with the next statement after the loop.

Ex:

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

In this example, the loop will iterate from 1 to 5. However, when the value of `i` becomes 3, the "break" statement is encountered, and the loop is terminated. The output will be:

```
1
2
```

2. "continue" statement:

The "continue" statement is used to skip the remaining code inside the loop for the current iteration and move on to the next iteration of the loop. It doesn't terminate the loop; instead, it jumps back to the beginning of the loop to start the next iteration.

Ex:

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

In this example, when the value of `i` becomes 3, the "continue" statement is encountered. It skips the remaining code for that iteration and moves to the next iteration. Therefore, the number 3 will not be printed. The output will be:

```
1
2
4
5
```

In summary, the "break" statement is used to exit the loop entirely, while the "continue" statement is used to skip the remaining code inside the loop for the current iteration and move on to the next iteration.

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

Sol:

In a for loop, the expressions `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` have slightly different ways of specifying the range of values. However, in practice, they will produce the same outcome in terms of iteration. Here's an explanation of the differences:

1. `range(10)`:

This expression defines a range starting from 0 (default) up to, but not including, the value specified (10 in this case). It generates a sequence of numbers from 0 to 9.

Example usage in a for loop:

```
for i in range(10):
    print(i)
```

Output:

0

1

2

3

4

5

6

7

8

9

2. `range(0, 10)`:

This expression explicitly specifies the start and end values of the range. It starts from 0 (inclusive) and goes up to, but not including, the end value (10 in this case). It generates the same sequence of numbers as `range(10)`.

Example usage in a for loop:

```
for i in range(0, 10):
    print(i)
```

Output:

0

1

2

3

4

5

6

7

8

9

3. `range(0, 10, 1)`:

This expression specifies the start value, end value, and step value of the range. It starts from 0 (inclusive), goes up to, but not including, the end value (10 in this case), and increments by the step value (1 in this case). Since the step value is 1, it generates the same sequence of numbers as the previous examples.

Example usage in a for loop:

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

Output:

0

1

2

3

4

5

6

7

8

9

In summary, all three expressions will result in the same sequence of numbers in a for loop, generating values from 0 to 9. The difference lies in the explicitness of start, end, and step values in the `range()` function.

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:

Using for loop:

```
for num in range(1,11):
        print(num)
```

Using while loop:

```
num = 1
while num >= 10:
        print(num)
        num+=1
```

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

Sol:

```
Import spam
Spam.bacon()
```