**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 represent binary logic, where "True" represents a logical true or "on" state, and "False" represents a logical false or "off" state.

In most programming languages, the syntax for writing these values is to use the keywords "True" and "False" (without quotation marks) with the first letter capitalized, such as:

True

False

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

The three different types of Boolean operators are:

AND operator: This operator returns "True" only if both operands are true. Otherwise, it returns "False". In many programming languages, the AND operator is represented by the symbol "&&" or the keyword "and".

OR operator: This operator returns "True" if at least one of the operands is true. It returns "False" only if both operands are false. In many programming languages, the OR operator is represented by the symbol "||" or the keyword "or".

NOT operator: This operator returns the opposite of the operand's value. If the operand is "True", it returns "False", and if the operand is "False", it returns "True". In many programming languages, the NOT operator is represented by the exclamation mark "!" or the keyword "not".

**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 ).**

Here are the truth tables for the three Boolean operators:

1. AND operator:

| Operand 1 | Operand 2 | Result |

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

| True | True | True |

| True | False | False |

| False | True | False |

| False | False | False |

2. OR operator:

| Operand 1 | Operand 2 | Result |

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

| True | True | True |

| True | False | True |

| False | True | True |

| False | False | False |

3. NOT operator:

| Operand | Result |

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

| True | False |

| False | True |

In each of these truth tables, the "Operand 1" and "Operand 2" columns represent the two Boolean values that are being evaluated by the operator, and the "Result" column represents the resulting Boolean value. Each row of the truth table shows a specific combination of input values and the resulting output value that is produced by the Boolean operator.

**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)**

Here are the values of the given expressions:

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

Value: False

Explanation: The expression "5 > 4" is True, but "3 == 5" is False. Therefore, the "and" operator evaluates to False.

2. not (5 > 4)

Value: False

Explanation: The expression "5 > 4" is True, but the "not" operator negates this value, resulting in False.

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

Value: True

Explanation: The expression "5 > 4" is True, and "3 == 5" is False. Therefore, the "or" operator evaluates to True.

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

Value: False

Explanation: The expression "(5 > 4) or (3 == 5)" is True because "5 > 4" is True. The "not" operator negates this value, resulting in False.

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

Value: False

Explanation: The expression "True == False" is False. Therefore, the entire expression evaluates to False.

6. (not False) or (not True)

Value: True

Explanation: The expression "not False" is True, and "not True" is False. Therefore, the "or" operator evaluates to True.

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

The six comparison operators in programming are:

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 or expressions and return a Boolean value of true or false based on the comparison result. For example, the expression "3 < 5" would return true because 3 is less than 5, while the expression "10 != 10" would return false because 10 is equal to 10.

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

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.

For example, if we have two variables `a` and `b`, and we want to check if `a` is equal to `b`, we would use the equal to operator like this:

```

if (a == b) {

// do something if a is equal to b

}

```

In this case, the expression `a == b` will return true if the values of `a` and `b` are the same, and false otherwise.

On the other hand, if we want to assign a value to a variable, we would use the assignment operator like this:

```

a = 5;

```

In this case, the value 5 is assigned to the variable `a`.

A condition where we might use the equal to operator is when we want to check if a user input matches a predefined value. For example:

```

var answer = "apple";

var userInput = prompt("What is your favorite fruit?");

if (userInput == answer) {

console.log("Correct!");

} else {

console.log("Incorrect.");

}

```

In this example, we are comparing the user's input to a predefined value (`answer`) using the equal to operator, and providing feedback based on the result.

A condition where we might use the assignment operator is when we want to update the value of a variable based on some calculation or input. For example:

```

var x = 5;

var y = 10;

x = x + y;

```

In this example, we are updating the value of `x` by adding the value of `y` to it using the assignment operator. The new value of `x` will be 15.

**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')**

Based on the indentation, there are three blocks in this code:

Block 1:

```

spam = 0

if spam == 10:

print('eggs')

```

This block assigns the value 0 to the variable `spam`, and then checks if `spam` is equal to 10. Since `spam` is not equal to 10, the `print('eggs')` statement is not executed.

Block 2:

```

if spam > 5:

print('bacon')

else:

print('ham')

```

This block checks if `spam` is greater than 5. Since `spam` is not greater than 5, the `print('ham')` statement is executed.

Block 3:

```

print('spam')

print('spam')

```

This block consists of two print statements that will always be executed, regardless of the value of `spam`. They will output the strings "spam" and "spam" respectively.

**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.**

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?**

If a program is stuck in an endless loop, the key combination that can be used to interrupt or stop the program depends on the programming environment or console that is being used.

In general, the following key combinations can be used to interrupt a program or stop an endless loop:

- Ctrl + C: This is a common key combination that can be used to interrupt a program on many platforms and environments. It sends a signal to the program to terminate the current operation and return control to the console.

- Ctrl + Break: This is another key combination that can be used to interrupt a program on some platforms and environments, such as Windows.

- Ctrl + \ (backslash): This key combination can be used to force a program to quit on some Unix-based systems.

- Alt + Ctrl + Delete: On Windows, this key combination opens the Task Manager, which can be used to force quit an unresponsive program.

It's important to note that stopping a program in an unexpected manner may cause data loss or other issues, so it's always best to try to avoid getting into an endless loop in the first place by writing robust code and testing it thoroughly.

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

Both `break` and `continue` are keywords in Python that can be used to control the flow of a loop.

The main difference between `break` and `continue` is that `break` is used to terminate the loop entirely, while `continue` is used to skip the current iteration of the loop and move on to the next iteration.

Here's an example to illustrate the difference:

```

# Using break

for i in range(1, 6):

if i == 3:

break

print(i)

# Output: 1

# 2

# Using continue

for i in range(1, 6):

if i == 3:

continue

print(i)

# Output: 1

# 2

# 4

# 5

```

In the first example, the `break` statement is used to terminate the loop when `i` equals 3. So only the numbers 1 and 2 are printed before the loop is terminated.

In the second example, the `continue` statement is used to skip over the iteration where `i` equals 3. So all the numbers from 1 to 5 are printed, except for 3.

So, in summary, `break` is used to completely terminate a loop when a certain condition is met, while `continue` is used to skip 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)?**

In a `for` loop, `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` all produce the same sequence of numbers from 0 to 9, but they differ in the way the parameters are specified.

`range(10)` specifies a range of numbers from 0 to 9, with a step size of 1. When `range()` is called with a single argument, it is interpreted as the `stop` parameter, and the `start` parameter is assumed to be 0, while the `step` parameter is assumed to be 1. So, `range(10)` is equivalent to `range(0, 10, 1)`.

`range(0, 10)` specifies a range of numbers from 0 to 9, with a step size of 1. Here, both the `start` and `stop` parameters are explicitly specified, while the `step` parameter is assumed to be 1 if it is not specified.

`range(0, 10, 1)` is also equivalent to `range(10)` and specifies the same range of numbers from 0 to 9, but with all three parameters explicitly specified.

In summary, `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are equivalent in terms of the sequence of numbers they produce in a `for` loop, but differ in the way their parameters are specified.

**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.**

Here's a short program that prints the numbers 1 to 10 using a `for` loop:

```

for i in range(1, 11):

print(i)

```

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

```

i = 1

while i <= 10:

print(i)

i += 1

```

In both cases, the loop starts at 1 and ends at 10, and the `print()` statement is used to print each number in the sequence. In the `for` loop, the `range()` function is used to generate the sequence of numbers, while in the `while` loop, the loop counter `i` is initialized to 1 and incremented by 1 on each iteration until it reaches 10.

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

If you have a function named `bacon()` inside a module named `spam`, you can call it after importing `spam` using dot notation as follows:

```python

import spam

spam.bacon()

```

Here, the `import` statement is used to import the `spam` module, and the function `bacon()` can be accessed using the dot notation `spam.bacon()`. This tells Python to look for the `bacon()` function inside the `spam` module.

Alternatively, you could import the `bacon()` function directly using the `from` keyword, like this:

```python

from spam import bacon

bacon()

```

Here, the `from` keyword is used to import the `bacon()` function directly from the `spam` module, so you can call it without using the `spam.` prefix.