**Python Control Statements**

Control statements are like instructions that tell Python how to navigate through the code. They help in making decisions and looping over tasks.

Control statements in Python are used to control the flow of execution in a program based on certain conditions. They allow you to make decisions, execute code selectively, and handle different scenarios dynamically.

The main ones are if/else for making choices, loops like for/while for repeating tasks, and keywords like break/continue to control the flow.

Let's dive in and explore these concepts to understand them better.

**Here are some commonly used control statements:**

### **1. if Statement:**

The `if` statement executes a block of code only if a specified condition is true. If the condition evaluates to false, the code block is not executed. It has the following

**syntax:**

```python

if condition:

# statement to be executed if the condition is true

```

### **2. if...else Statement:**

The `if...else` statement allows you to execute the instructions if one block of code condition is true, and another block runs when if the condition is false.

**It has the following syntax:**

```python

if condition:

# statement to be executed if condition is true

else:

# statement to be executed if condition is false

```

### **3. if...elif...else Statement:**

The `if...elif...else` statement allows you to check multiple conditions and execute different blocks of code based on which condition is true.

It has the following **syntax**:

```python

if condition1:

# code to be executed if condition1 is true

elif condition2:

# code to be executed if condition2 is true

else:

# code to be executed if all conditions are false

```

### **4. Nested if Statement:**

A nested `if` statement is an `if` statement that is placed inside another `if` statement. It allows you to check for additional conditions based on the result of the outer `if` statement.

It has the following **syntax**:

```python

if condition1:

if condition2:

# code to be executed if both condition1 and condition2 are true

```

### **5. Nested if...else Statement:**

A nested `if...else` statement is similar to a nested `if` statement, but it includes an `else` block for each `if` condition. It allows you to handle different scenarios based on multiple conditions. It has the following **syntax**:

```python

if condition1:

if condition2:

# code to be executed if both condition1 and condition2 are true

else:

# code to be executed if condition1 is true but condition2 is false

else:

# code to be executed if condition1 is false

```

**Example**:

```python

x = 10

if x > 0:

print("x is positive")

elif x < 0:

print("x is negative")

else:

print("x is zero")

```

This example demonstrates the usage of the **`if...elif...else`** statement to determine the sign of a number `x`. Depending on the value of `x`, it prints whether `x` is positive, negative, or zero.

Understanding and mastering control statements is crucial for writing effective and efficient Python programs, as they allow you to handle different scenarios and make your code more dynamic and adaptable.

## **Looping (For Loop, While Loop)**

## In Python are vital structures that enable the repetition of code execution, either for a specific count or until a certain condition is fulfilled.

**For Example**:

```python

**for** i in range(11):

print(i)

```

This snippet utilizes a for loop to iterate over numbers from 0 to 10 inclusively. With each iteration, the value of i is displayed. The loop runs until it hits 11, which is not part of the range.

**Another Example:**

```python

num = 1

**while** num <= 10:

print(num)

num = num + 1

```

In this code segment, a while loop is employed to print numbers ranging from 1 to 10. It starts at 1 and continues until it reaches 10, after which the loop ceases when the value of num exceeds 10.

## **Types of Control Statements in Python**

## **Break, Continue, and Pass**

These are control flow statements in Python that allow you to modify the behavior of loops and conditional statements.

**Break Statement:**

The break statement terminates the loop it is currently in, regardless of whether the loop condition is true or false.

**Example**:

```python

for i in range(10):

if i == 5:

break

print(i)

```

This code will print numbers from 0 to 4, then terminate the loop when i equals 5.

**Continue Statement:**

The continue statement skips the rest of the code inside the loop for the current iteration and proceeds to the next iteration of the loop.

**Example**:

```python

for i in range(10):

if i == 5:

continue

print(i)

```

This code will print numbers from 0 to 9, excluding 5, as the loop skips printing when i equals 5.

**Pass Statement:**

The pass statement is a null operation; nothing happens when it is executed. It is used as a placeholder when a statement is syntactically required but you have no need for any code to execute.

**Example**:

```python

for i in range(5):

if i == 3:

pass

else:

print(i)

```

This code will print numbers from 0 to 4, but when i equals 3, nothing will happen as the pass statement is executed.

These control flow statements provide flexibility and control over the flow of your Python code, allowing you to create more complex and efficient programs.

## Conclusion

Control statements in Python are used to control the flow of execution of a program. The three types of control statements are break, continue, and pass. These statements allow us to selectively execute specific parts of the code based on certain conditions, optimize performance, and handle errors. By using control statements in Python effectively, we can write more efficient and error-free code.