Introduction

Conditional statements are one of the most fundamental concepts in programming. They allow you to create code that executes different actions based on a certain condition. Without conditional statements, programming would be limited to executing the same instructions every time a program runs, which severely restricts the power and flexibility of the code.

In Python, conditional statements are used to control the flow of the program based on whether certain conditions are true or false. This allows you to create programs that make decisions based on user input, data values, or other factors, and execute different actions accordingly.

In the rest of this notebook, we'll dive into the syntax and usage of conditional statements in Python, including if, elif, and else statements.

Syntax of If Statements

The basic syntax of an if statement in Python is:

```
In [ ]: if condition:
            # execute this code block if the condition is True
```

Here's an example of how an if statement can be used to check whether a number is positive:

```
In [1]: number = 8
        if number > 0:

            print("The number is positive")
```

```
The number is positive
```

In this example, the condition number > 0 is evaluated to True, so the code block that follows the if statement is executed. That is why output of this program is:

The number is positive

In general, the condition in an if statement can be any expression that evaluates to a Boolean value (True or False). If the condition is true, the code block that follows the if statement is executed. If the condition is false, the code block is skipped and the program continues executing the next statement outside of the if block.

Here's an example program that uses an if statement to check whether a given number is positive or negative:

In [2]:
```python
# get input from user
number = int(input("Enter a number: "))

# check if number is positive
if number > 0:
    print("The number is positive")
```

```
Enter a number: 45
The number is positive
```

In this program, the user is prompted to enter a number using the input() function, which returns a string that is then converted to an integer using the int() function. The if statement then checks whether the number is greater than zero. If it is, the code block that follows the if statement is executed, which simply prints the message "The number is positive" to the console.

If the user enters a negative number, the code block inside the if statement is not executed and nothing is printed to the console. This shows how the if statement can be used to execute different code blocks based on whether a certain condition is true or false.

Else Statements

In Python, an else statement can be used in conjunction with an if statement to execute a different code block when the condition in the if statement is not met.

The basic syntax of an if-else statement is:

In [ ]:
```python
if condition:
    # execute this code block if the condition is True
else:
    # execute this code block if the condition is False
```

Here's an example of how an if-else statement can be used to check whether a number is positive or negative:

In [3]:
```python
# get input from user
number = int(input("Enter a number: "))

# check if number is positive or negative
if number > 0:
    print("The number is positive")
else:
    print("The number is negative or zero")
```

```
Enter a number: -20
The number is negative or zero
```

In this program, the user is prompted to enter a number using the input() function, which is converted to an integer using the int() function. The if statement then checks whether the number is greater than zero. If it is, the code block that follows the if statement is executed, which prints the message "The number is positive" to the console. If the number is not greater than zero, the else block is executed instead, which prints the message "The number is negative or zero" to the console.

This shows how the if-else statement can be used to execute different code blocks based on whether a certain condition is true or false.

Elif Statements

In Python, the elif statement is used in conjunction with the if statement to check multiple conditions. It allows you to specify multiple conditions and execute different code blocks based on which condition is true.

The basic syntax of an if-elif-else statement is:

In [ ]:
```python
if condition1:
    # execute this code block if condition1 is True
elif condition2:
    # execute this code block if condition1 is False and condition2 is True
elif condition3:
    # execute this code block if condition1 and condition2 are False and condition3
else:
    # execute this code block if all the conditions are False
```

Here's an example of how an if-elif-else statement can be used to check whether a number is positive, negative or zero:

In [4]:
```python
# get input from user
number = int(input("Enter a number: "))

# check if number is positive, negative or zero
if number > 0:
    print("The number is positive")
elif number < 0:
    print("The number is negative")
else:
    print("The number is zero")
```

```
Enter a number: 0
The number is zero
```

In this program, the user is prompted to enter a number using the input() function, which is converted to an integer using the int() function. The if statement checks whether the number is greater than zero. If it is, the code block that follows the if statement is executed, which prints the message "The number is positive" to the console. If the number is not greater than zero, the elif statement checks whether the number is less than zero. If it is, the code block that follows the elif statement is executed, which prints the message "The number is negative" to the console. If the number is not greater than zero and not less than zero, the else block is executed, which prints the message "The number is zero" to the console.

This shows how the if-elif-else statement can be used to check multiple conditions and execute different code blocks based on which condition is true.

Nested If Statements

In Python, a nested if statement is an if statement that is inside another if statement. This allows you to create more complex conditional statements by checking multiple conditions within a single code block.

The basic syntax of a nested if statement is:

In [ ]:
```
if condition1:
    # execute this code block if condition1 is True
    if condition2:
        # execute this code block if condition1 and condition2 are True
    else:
        # execute this code block if condition1 is True and condition2 is False
else:
    # execute this code block if condition1 is False
```

Here's an example of how a nested if statement can be used to determine if a person is eligible to vote:

In [5]:
```
# get input from user
age = int(input("Enter your age: "))
citizen = input("Are you a citizen of this country? (yes/no): ")

# check if person is eligible to vote
if age >= 18:
    if citizen == "yes":
        print("You are eligible to vote")
    else:
        print("Sorry, only citizens can vote")
else:
    print("Sorry, you are not old enough to vote")
```

```
Enter your age: 35
Are you a citizen of this country? (yes/no): yes
You are eligible to vote
```

In this program, the user is prompted to enter their age and citizenship status. The first if statement checks whether the person is 18 years or older. If they are, the nested if statement checks whether they are a citizen of the country. If they are, the message "You are eligible to vote" is printed to the console. If they are not a citizen, the message "Sorry, only citizens can vote" is printed. If the person is not 18 or older, the else statement is executed, which prints the message "Sorry, you are not old enough to vote".

This shows how nested if statements can be used to check multiple conditions and create more complex conditional statements.

Short-circuiting

In Python, short-circuiting is a technique used to simplify complex conditional statements by taking advantage of the fact that logical operators and and or only evaluate their operands until a certain condition is met.

For example, in an or statement, if the first operand evaluates to True, the second operand is not evaluated because the entire expression is already considered to be True. Similarly, in an and statement, if the first operand evaluates to False, the second operand is not evaluated because the entire expression is already considered to be False.

Here's an example of how short-circuiting can be used to check if a string is not empty before performing a certain action:

In [6]:
```
text = ""
if text and len(text) > 0:
    # perform some action
    print("The string is not empty")
else:
    print("The string is empty")
```

```
The string is empty
```

In this program, the if statement checks whether text is not empty and whether its length is greater than 0. However, because of short-circuiting, if text is empty, the second operand (len(text) > 0) is not evaluated because the entire expression is already considered to be False. This simplifies the conditional statement and makes the code more readable.

If the text variable is not empty, the second operand is evaluated and the code block within the if statement is executed, printing "The string is not empty". If text is empty, the else block is executed, printing "The string is empty".

Short-circuiting can be a powerful tool to simplify conditional statements and improve the efficiency of your code. However, it's important to use it judiciously and make sure that your code is still easy to understand and maintain.

Conclusion

In conclusion, conditional statements are a fundamental concept in Python programming that allow you to execute code blocks based on certain conditions. By using if statements, else statements, elif statements, nested if statements, and short-circuiting, you can create complex programs that are able to make decisions and take different actions based on a variety of inputs.

By using the examples and techniques covered in this notebook, you can start to write more powerful and efficient programs that are able to make decisions and take different actions based on a variety of inputs. So, start practicing and experimenting with conditional statements today, and take your Python programming skills to the next level!