Conditional Statements in Python

What are the if-else conditions? But before we jump into the nitty-gritty of programming, let's take a moment to understand where we encounter if-else conditions in our everyday lives.

Have you ever been faced with a choice and had to decide what to do? Well, that's when if-else conditions come into play. Imagine you wake up in the morning and look outside. If the sun is shining, you might decide to wear your favorite sunglasses and head out for a stroll.

If the sun is shining, you might decide to wear your favorite sunglasses.

But if it's raining, you'd probably grab an umbrella to stay dry. In both cases, you're making a decision based on a condition.

It's raining, you'd probably grab an umbrella to stay dry

Similarly, in programming, we use if-else conditions to make decisions based on certain conditions. It's like giving instructions to the computer, telling it what to do based on different situations. In Python, we use if-else statements to express these conditions. Think of it like a recipe with specific instructions. We begin with an "if" statement, followed by a condition written inside parentheses. If that condition is true, it means something is happening, and we execute the code inside the "if" block. If the condition is not true, then something else is happening, and we execute the code inside the "else" block. It's like having different sets of instructions for different situations. Just like in real life, where we have different outcomes depending on the conditions, we can create different paths in our program using if-else conditions. It allows us to handle various scenarios and make our programs more intelligent and flexible. So, in this chapter, we'll explore how to use if-else conditions in Python to create decision-making logic in our code. Get ready to make your programs smarter and more responsive!

Conditional statements in Python can be further classified into the following types:

1. **If Statement**
2. **If-Else Statement**
3. **If-Elif-Else Statement**
4. **Nested If Statement**

1. **If Statement: A simple if statement checks a single condition. If the condition is true, the code block under the if statement is executed. Otherwise, it is skipped.**

But what exactly is a code block? Think of a code block as a set of instructions that are grouped together. In Python, we use indentation, which is the whitespace at the beginning of a line, to define a code block. Indentation means adding spaces or tabs at the beginning of each line within the code block. It helps Python understand which lines of code belong together. Let me show you an example:

```
if condition:
    # Code block starts here

    statement_1

    # Code block ends here
```

In this example, we have an 'if' statement with a condition. If the condition is true, the code block starts right after the colon and is indented. Here, we have a statement, represented by statement_1. This statement is executed only if the condition is true. Remember, the indentation is crucial to indicate that this statement is a part of the code block. So, indentation helps us define the boundaries of a code block. It's like placing the instructions within a box, telling Python which lines of code belong to the if statement. The colon (:) before the indented block serves as a signpost, signaling the beginning of the code block associated with the if statement. Make sure to keep consistent indentation throughout your code to avoid any errors.

Now that we understand how an if statement works and the importance of indentation, let's understand if statement with an easy example:

```
age =int(input( “Enter age:  ”))
if age >= 18:
```

print("You are eligible to vote.")

In this example, we want to check if a person is eligible to vote. We start by asking the user to enter their age using the input function. The entered value is stored in the variable age after converting it to an integer using int(). Next, we have the if statement. It checks whether the age variable is greater than or equal to 18. If the condition is true, which means the entered age is 18 or above, the code block under the if statement is executed. If the condition is false, the code block is skipped entirely, and nothing is printed.

```
age = int(input("Enter age:"))
if age>=18:
    print("You are eligible to vote.")
```

Output:

```
Enter age:
```

Entering 32

```
Enter age:32
You are eligible to vote.
```

Let's assume the person's age is 32. When we execute the code with an age of 32, The result is displayed on the screen i.e. “You are eligible to vote.”

Now, let's consider a different scenario. Assume the person's age is 14. When we execute the code with an age of 14,

age =int(input(“Enter age: ”))
if age >= 18:

print("You are eligible to vote.")(input age 14)

Output:

```
Enter age:14

In [11]:
```

The screen displays no output since the condition age >= 18 is false.

The **if-else statement** . This statement gives us two options to choose from based on a condition: one option if the condition is true and another option if it is false. Let me explain this with a code snippet:

```
if condition:
    # Code block for the true condition
    statement_1
else:
    # Code block for the false condition
    statement_2
```

In this example, we have an if-else statement. The condition is checked, and if it evaluates to true, the code block under the "if" branch executes. Here, we have a statement represented by statement_1. This statement is executed only if the condition is true.

If the condition is false, the code block under the "else" branch executes. In this case, we have a statement represented by statement_2. This statement is executed only if the condition is false.

So, with the if-else statement, we have the flexibility to handle different scenarios. Depending on the condition's result, our program can take different paths, executing specific instructions accordingly.

Let's understand if statement with an easy example:

```
age = int(input("Enter age: "))
if age >= 18:
    print("You are eligible to vote.")
else:
    print("Not eligible to vote.")
```

```
age = int(input("Enter age: "))

if age >= 18:
    print("You are eligible to vote.")
else:
    print("Not eligible to vote.")
```

[Execute the code and enter age:32]

```
Enter age: 32
You are eligible to vote.
```

Let's assume the person's age is 32. When we execute the code with an age of 32

'You are eligible to vote.' Is displayed on the screen because the entered age, 32, is greater than or equal to 18.

let's consider a different scenario and Assume that the person's age is 10. When we execute the code with the value of age as 10,

age = int(input("Enter age: "))
if age >= 18:

print("You are eligible to vote.")

else:

print("Not eligible to vote.")

(enter age 10)

Output:

```
Enter age: 10
Not eligible to vote.
```

The screen displays the output "Not eligible to vote" since the condition age >= 18 is false and so the else block got executed.

The **if-elif-else statement**. The if-elif-else statement is useful when we have multiple conditions to check. It allows us to provide different actions for different conditions. It's Python's way of indicating, "If the conditions before this one were not satisfied, then consider checking this condition. Let's understand this with a code snippet:

```
if condition_1:
    # Code block for condition_1 being true
    statement_1
elif condition_2:
    # Code block for condition_2 being true
    statement_2
else:
    # Code block when no conditions are true
    statement_3
```

In this example, we have an if-elif-else statement. Python checks the first condition, condition_1, and if it's true, it executes the code block under the "if" branch. The statement statement_1 is executed.

But what if condition_1 is not true? Python then moves to the next condition, condition_2, and checks if it's true. If condition_2 is true, it executes the code block under the "elif" branch. Here, the statement statement_2 is executed.

If neither condition_1 nor condition_2 is true, Python executes the code block under the "else" branch. In this case, the statement statement_3 is executed.

With the if-elif-else statement, we can efficiently handle multiple scenarios and ensure that the correct actions are taken based on the conditions met.

Consider this example that evaluates a student's performance based on their marks. We'll run three scenarios to demonstrate how if-elif-else works.

Here's the code:

```
marks = int(input("Enter marks: "))
if marks >= 90:
    print("Excellent!")
elif marks >= 70:
```

print("Good job!")

else:

print("Keep up the hard work!")

```
marks = int(input("Enter marks: "))

if marks >= 90:
    print("Excellent!")
elif marks >= 70:
    print("Good job!")
else:
    print("Keep up the hard work!")
```

Output:

```
Enter marks: 95
Excellent!
```

Let's assume the student's marks are 95. When we execute the code with marks as 95, As you can see, the program prints "Excellent!" because the entered marks, 95, are greater than or equal to 90. Now, let's consider a different scenario. Assume the student's marks are 80. When we execute the code with marks as 80, the screen displays the output "Good job!" since the condition marks >= 90 is false, but marks >= 70 is true. Now let's assume the student's marks are 60. When we execute the code with marks as 60, the screen displays the output "Keep up the hard work!" since both conditions marks >= 90 and marks >= 70 are false.

By running these three scenarios, The program evaluates a student's performance based on their marks. It provides feedback using different messages depending on the range in which the marks fall.

The if-elif-else statement is a powerful tool that allows our programs to make decisions based on multiple conditions. It enables us to produce various outcomes depending on the specific conditions met.

Nested if statements: These are if statements within other if statements, allowing us to create more complex decision-making structures.

Let's consider nested if statements as a series of gift boxes within gift boxes. Imagine you have a big gift box, and when you open it, you find another smaller gift box inside. This pattern continues with multiple layers of gift boxes, with each layer representing a different set of conditions or choices. In this scenario, each layer of the gift boxes represents a specific decision point, helping you narrow down your choices until you find the most suitable gift for your friend. Similarly, in programming, nested if statements allow you to create complex decision trees, with each level of nesting providing more specific conditions and actions based on the initial condition. Let’s understand this with a code snippet:

```
if condition_1:
    if condition_2:
        # Code block when both condition_1 and condition_2 are true
        statement_1
    else:
        # Code block when condition_1 is true, but condition_2 is false
        statement_2
else:
    # Code block when condition_1 is false
    statement_3
```

In this code snippet, we have nested if statements. First, Python checks condition_1. If it's true, it moves to the inner if statement and checks condition_2. If both condition_1 and condition_2 are true, Python executes the code block under the inner "if" branch, running statement_1.

But what if condition_1 is true, but condition_2 is false? In that case, Python executes the code block under the inner "else" branch, running statement_2.

If condition_1 is false, Python goes straight to the outer "else" branch and executes the code block there, running statement_3.

As previously discussed indentation is crucial in Python to indicate block structures and maintain the program's logic. Here's how it works in the provided code:

The outer if statement has no indentation, and it checks condition_1.

The inner if-else statement has one level of indentation, indicating that it is inside the body of the outer if statement.

The statement inside the inner if statement (statement_1) have an additional level of indentation, showing they are part of the inner if statement's body.

The else block of the inner if-else statement (statement_2) is indented at the same level as the inner if, indicating it is associated with the inner if statement.

The else block of the outer if statement (statement_3) is at the same level of indentation as the outer if, showing it is associated with the outer if statement.

The correct indentation helps Python identify the blocks and ensures that the code executes logically. Improper indentation may lead to syntax errors or unexpected program behavior.

Let’s understand this with a simple example where it checks a student's age and their grade to provide personalized feedback based on their achievement. Let's break it down step by step.

```
age = 18
grade = "A"

if age >= 18:
    if grade == "A":
        print("Congratulations on your achievement!")
    else:
        print("Keep striving for excellence!")
else:
    print("You need to be at least 18 years old to qualify.")
```

In this example, we have two variables: age and grade. The age variable is set to 18, and the grade variable is set to "A."

The outer if statement checks if the student's age is greater than or equal to 18.

If the condition is true , which means the student's age is 18 or above, the code inside the outer if block is executed. Now, let's look at the inner if statement.

The inner if statement checks if the student's grade is "A." If both conditions are true, the code inside the inner if block is executed. If the student's age is 18 or above and they have an "A" grade, the program prints "Congratulations on your achievement!"

Output:

```
Congratulations on your achievement!
```

If the student's age is 18 or above but they don't have an "A" grade, the code inside the else block of the inner if is executed.

```
age = 18
grade = "B"

if age >= 18:
    if grade == "A":
        print("Congratulations on your achievement!")
    else:
        print("Keep striving for excellence!")
else:
    print("You need to be at least 18 years old to qualify.")
```

Output:

```
Keep striving for excellence!
```

The program prints "Keep striving for excellence!" when the student's age is 18 or above, but their grade is not "A."

Lastly, if the student's age is less than 18, the code inside the outer else block is executed, and the program prints "You need to be at least 18 years old to qualify.

```
age = 15
grade = "A"

if age >= 18:
    if grade == "A":
        print("Congratulations on your achievement!")
    else:
        print("Keep striving for excellence!")
else:
    print("You need to be at least 18 years old to qualify.")
```

Output:

```
You need to be at least 18 years old to qualify.
```

Let's see one more example:

```
# Get input from the user
age = int(input("Enter your age: "))

# Check age conditions using nested if statements

if age >= 16:
    print("You are eligible to apply for a learner's permit to drive.")

    # Nested if statement for additional condition

    if age >= 18:
        print("You are eligible for a full driver's license.")

    else:
        print("You can drive with a provisional license under supervision.")

else:
    print("Sorry, you must be at least 16 years old to apply for a learner's permit.")
```

In this example, the program checks if the user is at least 16 years old to be eligible to apply for a learner's permit to drive. The nested if statement then checks if the user is 18 or older, indicating eligibility for a full driver's license. If the outer if condition is not met, it prints a message indicating that the user is not eligible to apply for a learner's permit.

The output is:

```
Enter your age:
```

Let’s input the age as 24.

```
Enter your age: 24
You are eligible to apply for a learner's permit to drive.
You are eligible for a full driver's license.
```

You are eligible to apply for a learner's permit to drive.

You are eligible for a full driver's license.

This indicates that at the age of 24, the user is eligible for both a learner's permit and a full driver's license.

And that's how the nested if-else statement works! By using this structure, we can create more complex decision-making structures to provide personalized feedback based on multiple conditions.