Control Flow Statements In Python

CONTROL FLOW STATEMENT

Good day, students!

This is the first module in a new series on control flow statements. I hope you're ready to take your Python programming skills to the next level by learning new topics and coding.

Let’s get started with understanding the basic concept of control flow sta

A control statement is a statement that determines whether other statements will be executed.

An if statement decides whether to execute another statement, or decides which of two statements to execute.

A loop decides how many times to execute a statement.

Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

Conditional statements, transfer statements, and iterative statements are the three main types of control flow statements. These types are further classified into different categories. We will discuss these categories in detail later.

But it's important to understand the fundamental concept of decision making.

Decision making is the most important aspect of almost all programming languages.

As the name implies, decision making allows us to run a particular block of code for a particular decision. You can set up a condition and tell the compiler to execute a particular action if the condition is met. In case the condition is not met, you can instruct the compiler to execute a different block of code.

Here we begin with the first type of conditional statement called the if statement.

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

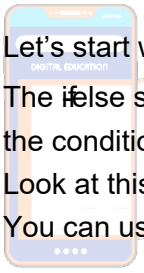
Let’s look at an example of if statement to understand better.

In this example you can check whether the number entered by the user is even or odd using user input. This can be done by checking if the number is perfectly divisible by 2. If the remainder of the modulus operation is zero then the number is even, otherwise the number entered by the user is odd.

Here the user entered 22, therefore the output is even since 22 divided 2 leaves zero reminder.
I hope this explanation is clear and you have understood the if statement.

The if statement

Welcome back to the module on control flow statements, in the previous module we discussed if statements. In one we will cover the if else statement.



Let's start with the definition of the if else statement.
The if else statement provides an else block combined with the if statement which is executed in the false case o the condition. If the condition is true, then the if block is executed. Otherwise, the else block is executed.
Look at this example.
You can use this to see if the user is eligible to vote by entering their age.

The if else statement will check the user input and execute the if block of code when the condition is True, and if condition is False, it will execute the else block of code.
In this case, the user input is 25, therefore the output is you are eligible to vote.

Let us move on to the third type of conditional statement called the elif statement.
The elif statement enables us to check multiple conditions and execute the specific block of statements dependi upon the true condition among them.

Consider the following example. This block of code is used to determine a student's performance by using marks as input.
Depending on the marks entered the elif statement checks multiple conditions one by one and if the condition fu then executes that code in which category the student falls.
Here the input is 62. Since the input does not fulfill the first condition the compiler will check the next condition.
Now the input marks of 62 is less than 80 but more than 60, therefore the output will be good.

The elif statement is useful when you need to check multiple conditions. With these we can make a tricky decision.

Loops

Hello there, learners. Welcome back to the control flow statements module. We'll go through loop statements and their many types in this module.

Let's start off with understanding what loops are.

The flow of the programs written in any programming language is sequential by default.

Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times. This is where loops come into use.

There are many advantages of using a loop in programming. Such as.

It provides code reusability.

By using loops, we do not need to write the same code again and again.

Using loops, we can traverse over the elements of data structures (array or linked lists).

This takes us to the for loop.

What is for loop?

The for loop in Python is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

But before that let me explain the range function to understand the for loop better.

The range function is used to generate a sequence of numbers.

The range function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

The syntax for range function is.

range (Start, Stop, Step)

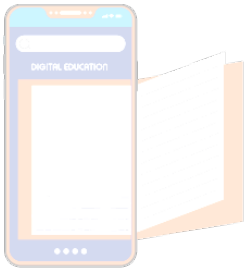
Let's look at an example of elif statement to better understand.

By using the user's limit as input, you can construct a range of natural numbers.
So, if the input is 6.
The output will then be a number ranging from 0 to 5. Please keep in mind that the sequence begins with zero a
default starting number and ends before the required number is entered.

Here are two more examples of the if else statements using range functions.
In this example, you can determine the start, stop and step.
The user input in this case for stop is 12. Therefore the range will have 1,4,7,10 as output since the step or the r
gap is 3.

Similarly in the second example, you can take input for the range to generate a series of float numbers with
the step of 0.1, starting from 1.
Here the user input is 1.5. Therefore the range will have 1.0, 1.1, 1.2, 1.3, and 1.4 as output.

I hope the concept of range is clear to you with these examp



Terating a List

Hello and welcome to this module on loops. You will learn about nested for loops and use of else statements wi
loop in this module.
Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n times for
every iteration of the outer loop.

Look at the example.

Here a range of asterisks will print as the nested loop.

The "inner loop" will be executed one time for each iteration of the "outer loop". It's up to the input from the user to end
the loop.

The user input in this case is 5.

As a result, the range in the nested loop is raised by one every iteration until it reaches five times.

In this case, the number is written according to the user input.

If the input is 5, the nested loop iterates 5 times, each time adding one.

Moving on to the next topic which is using else statements with for loop.

Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop wh executed only when all the iterations are exhausted.

For example if a range is defined in for statement, then the else keyword in the for loop specifies a block of code be executed when the loop is finished which in this case is 5.

So the output will not include 5 but by default will start from zero and print upto 4 followed by the message since t loop is exhausted.

Using else statement with while loop

Hello learners. Welcome back to this module on control flow statements. We will continue with loops and discus while loop and will talk about using else statements with while loop.

Let's start with the while loop.

The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

For example if the value of I is set to 1 and the while loop is set to check the result until it is less than or equal to 5.

Because the while loop statement executes the code block repeatedly while the condition is true, the output is printed from 1 to 5.

To help you learn the while loop, below are some examples.

This code generates odd numbers from 1 to ten.

The code will be run in a loop until all of the odd numbers have been generated, which will be at least 10.

Similarly, the while loop may be used to generate even integers from 0 to 10, which can be done by increasing t value of i by 2 each time until the loop condition equals 10.

Moving on to the next topic of the module which is using the else statement with a while loop.
Python enables us to use the else statement with the while loop also.
The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using a break statement, then the else block will not be executed and the statement present after else block will be executed.
Take a look at the following example. The value of i is set to 1 and the loop continues until the value of i falls below or equals 5. When the while loop has run its course, the else statement will be performed. As a result, the output will consist of a series of numbers ranging from 1 to 5, followed by the else statement message.

I hope the topic of while loop and use of else statements with it is clear.

Break statement

Hello learners. In this module we will discuss break, continue and pass statements with examples.

We will begin with a break statement. The break is a keyword in python which is used to bring the program control out of the loop.

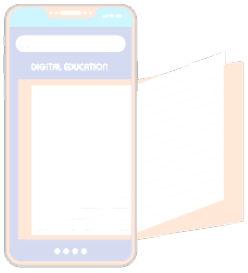
The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.
The list in this example contains a series of numbers ranging from 10 to 70.The if condition is used to see if number 50 is present in the list and then identify its position in the list. Instead of performing the remaining iterations, the break statement is used to end the loop as soon as the condition is met. When a break statement is encountered, the controller exits the loop immediately. According to the output, 50 was placed at the fifth position on the list.

Moving on to the continue statement. The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and starts with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

A range of 1 to 8 is used in this example. The condition is that the output should print skip if i is less than 6, and continue statement is used to skip the current iteration and continue with the following iteration till 8.

The last topic of the module is the pass statement. The pass statement is a null operation since nothing happens when it is executed. It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement in its place.
Here in the example the list contains vowels. Now the if statement will check if the list has a, if the condition is true then the pass statement will do nothing and print of the vowels.
It is useful in a situation where we are implementing new methods or also in exception handling.

I hope the concepts of break, continue, and pass statements are understood after the detailed explanation of examples.



vityarthi

Iterating a List

Welcome to the module on control flow statements. In this module you will learn about iterating a list, adding, assigning and removing elements from a list.
But first you must understand how you can iterate a list.
A list can be iterated by using a for in loop. A simple list containing four strings can be iterated as follows.
List = ["John", "David", "Ram", "Abdullah"]
By using for in loop you iterate through all the elements in a list.

You can add elements to a list as well.

Python provides an append() function by using which we can add an element to the list.

However, the append() method can only add the value to the end of the list.

In this example, the variable n will have the value of the number of elements in the list assigned by the user. Once that is determined, you can take the list of numbers with the list.append function from the user. Now can print the list with the print statement to see the user input.

Similarly you can remove elements from the list using the remove function such as in this example. The list already contains numbers, you can print and check the original list.

With the use of remove function, you remove a number from the list and print the updated list.

I hope the basic concept of iteration and adding or removing elements from a list has been understood.

In the following modules, we'll cover similar interesting topics.