**Module 6. Loops**

**Learning Objectives**

* Understand the basic syntax and structure of for and while loops in Python.
* Learn to use the break and continue statements to control loop execution.
* Practice iterating over different sequences such as lists, tuples, strings, and ranges.
* Master the use of the else clause with loops for post-loop execution.
* Identify and handle infinite loops and their use cases effectively.

Loops in Python are tools that let you repeat a set of instructions multiple times. Think of them as doing something over and over again until a condition is met. Loops are needed to automate repetitive tasks. Instead of writing the same code multiple times, you can use a loop to do it for you. This makes your code cleaner, more efficient, and easier to maintain.

There are two main types of loops in Python: ***for*** loops and ***while*** loops.

1. **The *for* loops**
   1. **Overview**

A ***for*** loops is used to iterate over a sequence (like a list, tuple, dictionary, set, or string) and execute a block of code for each item in the sequence. This is akin to using an iterator in other object-oriented languages.

When you use a for loop in Python, it steps through each item in the sequence one by one. The syntax is straightforward:

|  |
| --- |
| # Syntax of a for loop  for item in sequence:  # Do something with item |

Here's a practical example:

|  |
| --- |
| # Example: Print each item in a list  fruits = ["apple", "banana", "cherry"]  for fruit in fruits:  print(fruit) |

In this example, the loop goes through each item in the fruits list and prints it out.

**How It Differs from Other Languages**

In many other languages, the for loop is often used with a counter, like this:

|  |
| --- |
| // Example in Java  for (int i = 0; i < fruits.length; i++) {  System.out.println(fruits[i]);  } |

In Python, the ***for*** loop abstracts away the counter and directly iterates over the items, making the code cleaner and more readable.

**Why Use *for* Loops?**

* **Efficiency:** You can process each item in a sequence without manually managing a counter.
* **Readability:** Code is simpler and easier to understand.
* **Versatility:** Works with any iterable, including lists, tuples, sets, dictionaries, and strings.

**For Loops in Python Without Indexing Variables**

In many programming languages, you might need to initialize an index variable to loop through elements. However, Python’s ***for*** loop abstracts this, making iteration over sequences more straightforward and readable.

**Looping Through a String**

Strings in Python are iterable objects, meaning you can loop through each character in the string using a for loop.

Example: Loop Through the Letters in the Word "Hello"

|  |
| --- |
| # Loop through each character in the string "Hello"  for x in "Hello":  print(x) |

(Explanation)

* **Iterable String**: The string "Hello" is an iterable object, meaning it can be traversed one character at a time.
* **For Loop**: The for loop iterates over each character in the string.
* **Print Statement**: Inside the loop, the print(x) statement outputs each character individually.
* **Practical Use Cases:**
  + **Processing Characters**: Useful for tasks like character counting, finding vowels/consonants, or any character-specific processing.
  + **String Manipulation**: Can be used to create new strings based on certain conditions.
  1. **The *break* Statement**

The ***break*** statement allows you to exit the loop immediately when a specified condition is met, regardless of where you are in the loop.

Example: Exiting the Loop When x is "blue"

Let's look at an example where we have a list of colors. We want to print each color, but stop the loop as soon as we encounter "blue".

|  |
| --- |
| colors = ["red", "blue", "green"]  for x in colors:  print(x)  if x == "blue":  break |

(Explanation)

* **List of Colors:** We have a list colors containing "red", "blue", and "green".
* **For Loop:** The loop iterates over each item in the list.
* **Print Statement:** Each color is printed.
* **Condition:** When the loop encounters "blue", the if condition if x == "blue": becomes true.
* **Break Statement:** The break statement is executed, which exits the loop immediately.
  1. **The *continue* Statement**

The ***continue*** statement allows you to skip the rest of the code inside the current iteration of the loop and move to the next iteration.

Example: Skipping "blue"

Let's look at an example where we have a list of colors, and we want to print each color except for "blue".

|  |
| --- |
| colors = ["red", "blue", "green"]  for x in colors:  if x == "blue":  continue  print(x) |

(Explanation)

* **List of Colors:** We have a list colors containing "red", "blue", and "green".
* **For Loop:** The loop iterates over each item in the list.
* **Condition:** When the loop encounters " blue ", the if condition if x == " blue ": becomes true.
* **Continue Statement:** The continue statement is executed, which skips the rest of the code in the current iteration and proceeds with the next iteration.
* **Print Statement:** The fruit is printed only if it is not " blue ".
  1. **The *range()* Function**

The ***range()*** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number. You can also specify a different starting point and increment.

Example: Using the *range()* Function

Let's look at an example where we use the ***range()*** function to loop through a set of code six times.

|  |
| --- |
| for x in range(5):  print(x) |

(Explanation)

* **Range Function:** ***range(5)*** generates a sequence of numbers from 0 to 4. It does not include 5.
* **For Loop:** The loop iterates over each number in the range.
* **Print Statement:** Each number in the sequence is printed.

**Customizing the *range()* Function**

You can customize the ***range()*** function by specifying the start, stop, and step values.

Example: Custom Start, Stop, and Step

|  |
| --- |
| # Start at 2, stop before 10, increment by 2  for x in range(2, 10, 2):  print(x) |

(Explanation)

* **Start Value**: The sequence starts at 2.
* **Stop Value**: The sequence stops before 10.
* **Step Value**: The sequence increments by 2.
  1. **The *else* in *for* Loop**

The ***else*** keyword in a ***for*** loop provides a way to execute a block of code once the loop has finished iterating over all items. This else block will not execute if the loop is terminated by a ***break*** statement.

(Example)

Let's look at an example where we print all numbers from 0 to 4 and then print a message when the loop has ended.

|  |
| --- |
| for x in range(5):  print(x)  else:  print("Finished!") |

(Explanation)

* **Range Function:** ***range(5)*** generates a sequence of numbers from 0 to 4.
* **For Loop:** The loop iterates over each number in the range.
* **Print Statement:** Each number is printed.
* **Else Block:** After the loop completes all iterations, the else block is executed, printing "Finished!".
  1. **Nested Loops**

Nested loops involve having one loop inside another. The inner loop will execute all its iterations for each iteration of the outer loop.

(Example)

Let's look at an example where we have a list of colors and a list of sizes. We want to print each color with every size.

|  |
| --- |
| colors = ["red", "blue", "green"]  sizes = ["small", "medium", "large"]  for color in colors:  for size in sizes:  print(color, size) |

(Explanation)

* **Outer Loop**: Iterates over each color in the colors list.
* **Inner Loop**: For each color, iterates over each size in the sizes list.
* **Print Statement**: Prints the current color and size.

1. **The *while* loops**
   1. **Overview**

The ***while*** loops repeat a block of code as long as a condition is true. It's like saying "keep doing this until this condition is no longer true.". It's particularly useful when the number of iterations is not known beforehand.

Let's look at an example where we print numbers from 1 to 5 using a while loop. We will use an indexing variable i, which we initialize to 1 and increment in each iteration.

|  |
| --- |
| # Initialize the indexing variable  i = 1  # Start the while loop with a condition  while i <= 5:  print(i)  i += 1 # Increment the indexing variable |

In this example, the loop prints numbers from 1 to 5. It keeps running as long as the number is less than or equal to 5. After each print, the number is increased by 1.

(Explanation)

* **Initialize Indexing Variable**: ***i*** is set to 1 before the loop starts.
* **While Loop Condition**: The loop continues as long as ***i*** is less than or equal to 5.
* **Print Statement**: Each iteration prints the current value of ***i***.
* **Increment**: ***i*** is incremented by 1 after each iteration.
  1. **The *break* Statement**

The ***break*** statement allows you to exit the loop immediately when a specified condition is met, regardless of the loop's condition.

Example: Exiting the Loop When ***a*** is 5

Let's look at an example where we print numbers from 1 to 9, but we want to exit the loop when ***a*** equals 2.

|  |
| --- |
| a = 1  while a < 10:  print(a)  if a == 5:  break  a += 1 |

(Explanation)

* **Initialize Indexing Variable**: ***a*** is set to 1 before the loop starts.
* **While Loop Condition**: The loop continues as long as ***a*** is less than 10.
* **Print Statement**: Each iteration prints the current value of ***a***.
* **Break Condition**: When ***a*** equals 5, the ***if*** condition becomes *true* and the ***break*** statement is executed.
* **Increment**: ***a*** is incremented by 1 after each iteration, except when the loop is broken.
  1. **The *continue* Statement**

The ***continue*** statement skips the rest of the code in the current iteration and moves to the next iteration of the loop.

Example: Skipping an Iteration When ***a*** is 5

Let's look at an example where we print numbers from 1 to 10, but we want to skip printing the number 5.

|  |
| --- |
| a = 0  while a < 11:  a += 1  if a == 5:  continue  print(a) |

(Explanation)

* **Initialize Indexing Variable**: ***a*** is set to 0 before the loop starts.
* **While Loop Condition**: The loop continues as long as ***a*** is less than 11.
* **Increment:** ***a*** is incremented by 1 at the beginning of each iteration.
* **Continue Condition:** When ***a*** equals 5, the ***if*** condition becomes true, and the continue statement is executed. This skips the print statement for that iteration and proceeds to the next iteration.
* **Print Statement**: Prints the current value of ***a***, except when ***a*** equals 5.
  1. **The *else* in *while* Loop**

The ***else*** statement is used to run a block of code once the ***while*** loop condition becomes false. The code inside the ***else*** block will only execute after the loop has completed all iterations without being interrupted by a ***break*** statement.

(Example)

Let's look at an example where we print numbers from 0 to 4 and then print a message once the loop condition is false.

|  |
| --- |
| x = 0  y = 5  while x < y:  print(x)  x += 1  else:  print("x is no longer less than y") |

(Explanation)

* **Initialize Indexing Variable**s: ***x*** is set to 0 and ***y*** is set to 5 before the loop starts.
* **While Loop Condition**: The loop continues as long as ***x*** is less than ***y***.
* **Print Statement**: Each iteration prints the current value of ***x***.
* **Increment**: ***x*** is incremented by 1 after each iteration.
* **Else Block**: When the loop condition ***x < y*** becomes false, the else block is executed, printing "***x is no longer less than y***".
  1. **The *infinite* Loop**

An infinite loop in Python is a loop that continues to run indefinitely because the terminating condition is never met. This can happen in a ***while*** loop if the loop's condition always remains true and there is no mechanism inside the loop to change it. Infinite loops can be useful in certain situations but are often unintended and can cause a program to become unresponsive.

(Example 1)

|  |
| --- |
| while True:  print("This loop will run forever.") |

(Explanation)

* The condition ***True*** is always true, so the loop never stops.
* The ***print*** statement will execute repeatedly without end.

(Example 2)

|  |
| --- |
| a = 0  while a < 10:  print(a)  # Missing increment of a |

(Explanation)

* The loop condition ***a*** ***< 10*** starts as true because ***a*** is initialized to ***0***.
* The value of ***a*** is never changed inside the loop, so the condition ***a < 10*** remains true indefinitely.
* The loop will keep printing ***0*** forever.
  1. **Various Use Cases of While Loop**

A ***while*** loop can terminate based on a string condition. In this case, the loop continues until it encounters an empty string, which causes the loop condition to evaluate to ***False***.

(Example)

|  |
| --- |
| names = ["Monty", "", "Python"]  i = 0  while names[i]:  print(names[i])  i += 1 |

(Explanation)

* The ***while*** loop iterates through the ***names*** list starting from the first element.
* The loop stops when it encounters an empty string (""), as empty strings evaluate to ***False***.
* Only the first element, "***Monty***", is printed because the second element is an empty string, which causes the loop to terminate.

**Summary**

1. Loops in Python are tools that let you repeat a set of instructions multiple times, automating repetitive tasks.
2. The two main types of loops in Python are ***for*** loops and ***while*** loops.
3. A ***for*** loop iterates over a sequence (like a ***list***, ***tuple***, ***dictionary***, ***set***, or ***string***) and executes a block of code for each item.
4. For loops in Python abstract away the counter and directly iterate over items, making code cleaner and more readable.
5. Using a for loop in Python is efficient, readable, and versatile, working with any iterable.
6. Strings in Python are iterable objects, allowing you to loop through each character in the string using a for loop.
7. The ***break*** statement allows you to exit the loop immediately when a specified condition is met.
8. The ***continue*** statement allows you to skip the rest of the code inside the current iteration and move to the next iteration.
9. The ***range()*** function returns a sequence of numbers, which can be customized with start, stop, and step values for iteration.
10. The ***else*** keyword in a ***for*** loop executes a block of code once the loop has finished iterating over all items, unless interrupted by a break statement.
11. Nested loops involve having one loop inside another, allowing complex iterations such as printing each color with every size.
12. A ***while*** loop repeats a block of code as long as a condition is true, useful when the number of iterations is not known beforehand.
13. The ***break*** statement in a while loop exits the loop immediately when a specified condition is met.
14. The ***continue*** statement in a while loop skips the rest of the code in the current iteration and moves to the next iteration.
15. Infinite loops continue running indefinitely because the terminating condition is never met, which can be useful but often unintended.

**Programming Exercises**

1. **Population Growth**

Write a program that calculates the population of a city over a period of years. The program should ask the user for the initial population, the annual growth rate (as a percentage), and the number of years to simulate.

Use a loop to calculate and display the population at the end of each year.

1. **Savings Account Balance**

Write a program that calculates the balance of a savings account over a period of time. The program should ask the user for the initial balance, the annual interest rate, and the number of years.

Use a loop to display the balance at the end of each year, considering the interest compounding annually.

1. **Temperature Conversion**

Write a program that displays a table of Fahrenheit temperatures 32 through 212 and their Celsius equivalents.

The formula for converting a temperature from Fahrenheit to Celsius is:

1. **Multiplication Table**

Write a program that uses nested loops to display a multiplication table for the numbers 1 through 10. The outer loop should iterate over the numbers 1 through 10, and the inner loop should multiply the outer loop variable by each number from 1 to 10.

1. **Loan Amortization Schedule**

Write a program that calculates the amortization schedule for a loan. The program should ask the user for the loan amount, annual interest rate, and number of years for repayment.

Use a loop to display the remaining balance after each monthly payment.

1. **Factorial Calculation**

Write a program that calculates the factorial of a given number using a loop. The factorial of a non-negative integer ***n*** is the product of all positive integers less than or equal to ***n***.

1. **Countdown Timer**

Write a program that asks the user for a number of seconds and then counts down to zero, displaying the remaining seconds. The loop should pause for one second between each iteration.

1. **Prime Numbers**

Write a program that asks the user for a number and then uses a loop to determine whether the number is prime. A prime number is a number greater than 1 that has no positive divisors other than 1 and itself.

1. **Sales Tax Calculation**

Write a program that calculates the total price of items in a shopping cart, including sales tax. The program should ask the user to enter the price of each item and a sales tax rate.

Use a loop to calculate and display the total price with tax after all items have been entered.

1. **Interest Earned**

Write a program that asks the user for an initial investment amount and an annual interest rate. Use a loop to calculate and display the amount of interest earned each year for a specified number of years. At the end, display the total amount of interest earned.