Programming Refresher

Conditional Statements and Loops

Learning Objectives

By the end of this lesson, you will be able to:

- Apply decision control structures to solve problems in Python
- Identify and use different types of loops to manage repetitive tasks in Python
- Implement and manage loop control statements effectively in Python programs to enhance code efficiency and control flow
- Utilize the range function to create sequences and control loops in Python

Business Scenario

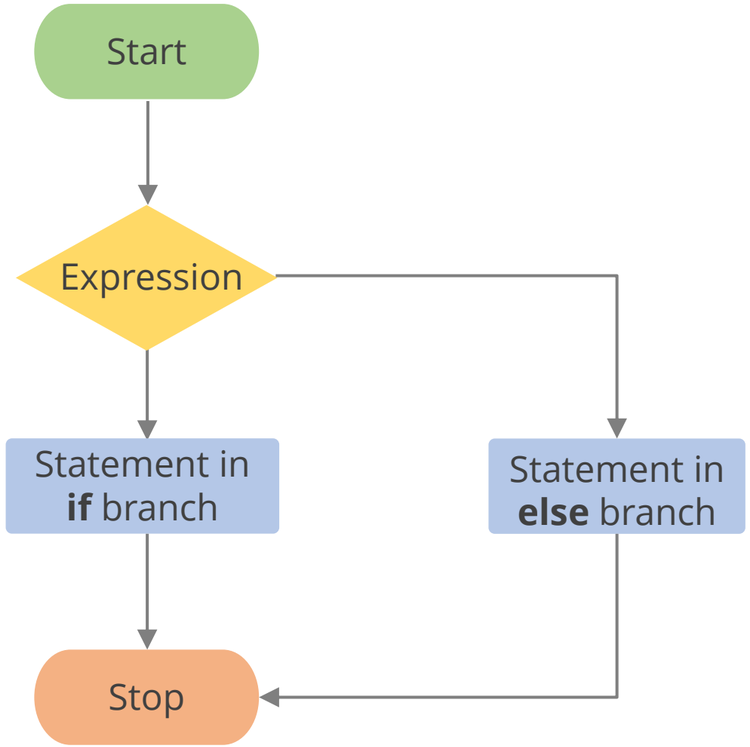
ABC is an e-commerce organization that lists online products based on purchases, reviews, and availability. Users filter these products based on multiple factors. The products are listed only if they match certain criteria; otherwise, an error is displayed.

The organization needs to develop this module using control structures, implement range functions to filter based on ranges, and use looping control statements.

Decision Control Structures in Python

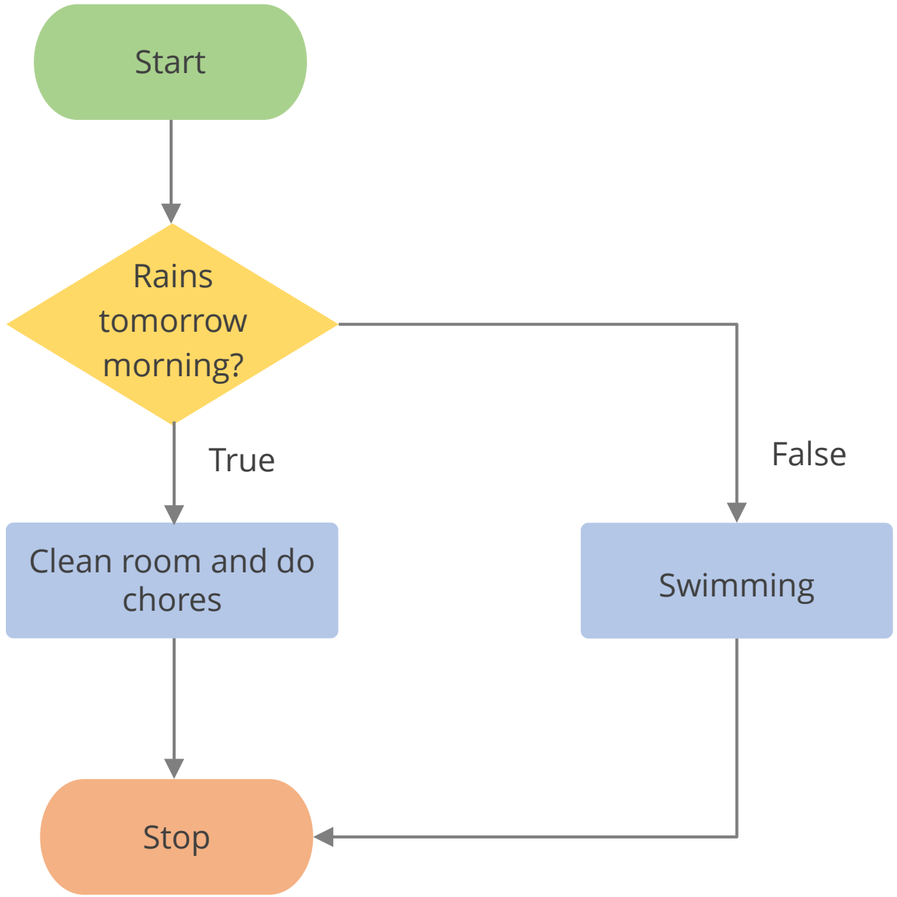
Decision Control Structures: Decision-Making

Decision-making is the process of making choices or performing tasks based on conditions. Decision control structures evaluate variables or expressions that return either True or False.



Decision Control Structures: Scenario

- Based on the weather, I can choose my activities for the day.
- **IF** it rains tomorrow morning, I will clean my room and do chores.
- **ELSE**, I will go swimming.



Decision Control Structures

Python offers four types of decision control structures, they are:

- If statement
- If-else statement
- Nested-if statement
- If-elif-else statement

If Statement

Python uses the if statement to change the program's control flow.
The indentation marks the block of code.

Syntax

```
if condition:
    statement
    statement
    .....
```

- A colon indicates the beginning of the block.
- The block is usually indented by four spaces.
- Each statement within the block must have the same indentation.

If Statement: Example

Example

```
inp = input("Nationality ? ")
```

```
Nationality ? French
```

```
if inp == "French":
    print("Préférez vous parler francais?")
```

```
Préférez vous parler francais?
```

If-Else Statement

The if-else statement evaluates the condition and executes the *if* block only when the test condition is True. Otherwise, it executes the *else* block.

Syntax

```
if condition:
    statement 1
    statement 2
else:
    statement 3
    statement 4
```

The else statement is optional in the if-else construct.

If-Else Statement: Example

Example

```
num = int(input('Enter a number : '))
```

```
Enter a number : -45
```

```
if num > 0:
    print(num, 'is positive number.')
else :
    print(num, 'is negative number.')
```

```
-45 is negative number.
```

If-Elif-Else Statement

The if-elif-else statement allows checking multiple conditions. If the if condition is False, it checks the next elif condition, and so on.

Syntax

```
if condition 1:
    statement
elif condition 2:
    statement
elif condition 3:
    statement
else:
    statement
```

Only one block among the if-elif-else blocks is executed. If all conditions are False, the else block is executed.

If-Elif-Else Statement: Example

Example

```
marks = int(input('Enter Marks : '))
if marks >= 90 :
    print('Grade A')
elif marks >= 70 :
    print('Grade B')
elif marks >= 55:
    print('Grade C')
elif marks >= 35:
    print('Grade D')
else :
    print('Grade F')
```

```
Enter Marks :  56
Grade C
```

Nested-If

Python allows an *if* statement inside another *if* statement.

Syntax

```
if (condition 1):
    statement
    # Executes when condition 1 is True
    if (condition 2):
        # Executes when condition 2 is also True
    # inner if Block ends here
# outer if Block ends here
```

- This format is called nesting.
- Indentation defines the level of nesting.

Nested-If: Example

Example

```
num = 15
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

```
Positive number
```

True or False

Python evaluates the following objects as False:

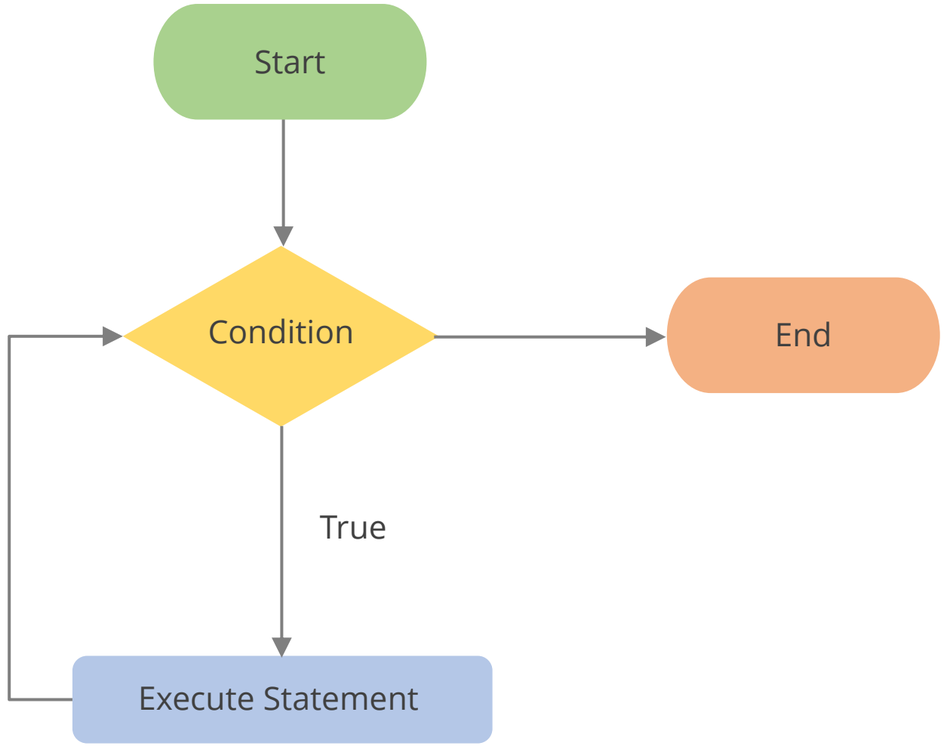
- Numerical zero values
- Boolean value False
- Empty strings
- Empty list, tuples, and dictionaries
- None

All other values are considered True in Python.

Loops

Loops

A loop statement allows the repeated execution of a statement or group of statements.



Types of Loops

The following types of loops handle looping requirements:

Count controlled loop

Condition controlled loop

Collection controlled loop

Count Controlled Loop

It is a method of repeating a loop a predetermined number of times.

Syntax

```
for <num> in <range>:
   body of loop
```

Condition Controlled Loop

A loop is repeated until a given condition changes from True to False or False to True, depending on the type of loop.

Syntax

```
while <condition is true>:
   body of loop
```

Collection Controlled Loop

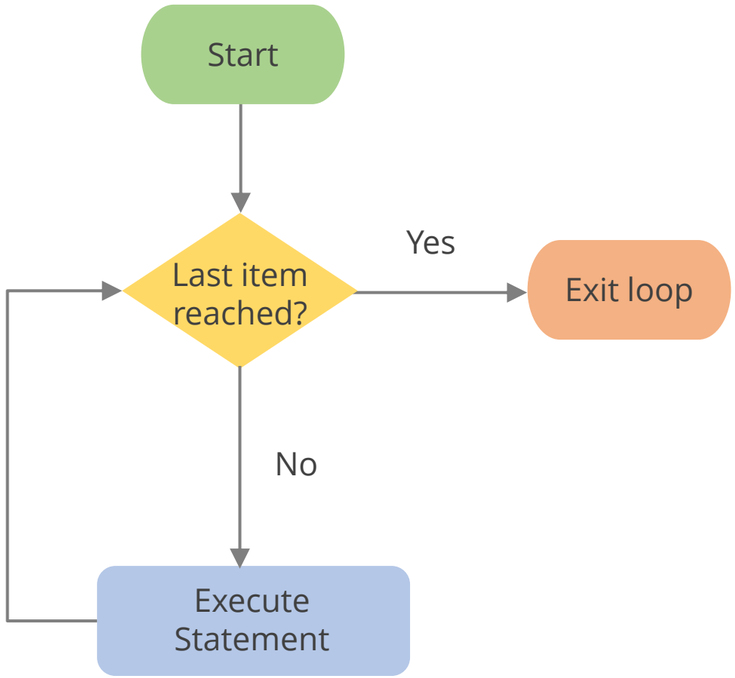
This special construct allows looping through the elements of a **collection**, which can be an array, list, or other ordered sequences.

Syntax

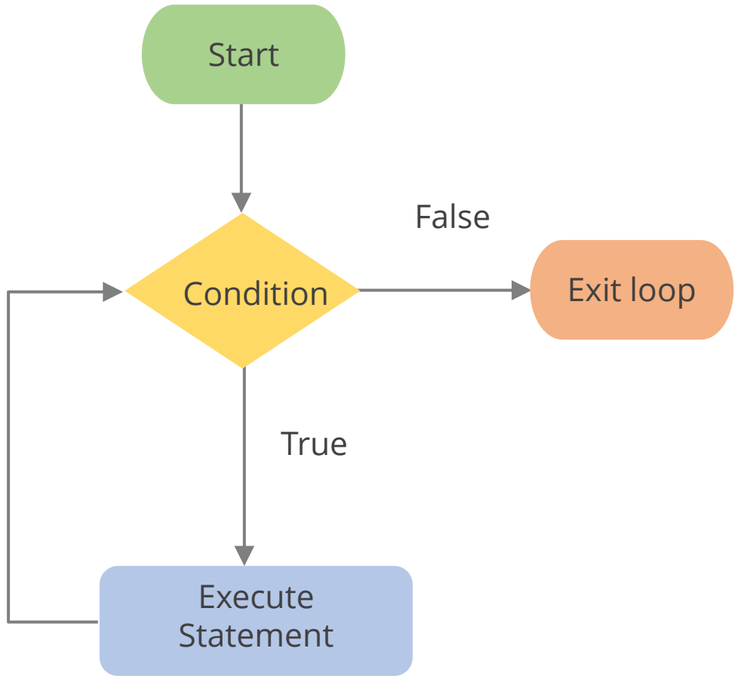
```
for <item> in <list>:
    body of loop
```

Loops in Python

Python supports **for** and **while** loop.



For loop



While loop

Loops in Python

for loop

The for loop iterates over a sequence list, tuple, string, or other objects. Its syntax is:

```
for a in iteration_object:
 Body
 of
 loop
```

while loop

The while loop iterates over a block of code if the test expression is true. Its syntax is:

```
while test_expression:
 Body
 of
 loop
```

Loops in Python: Example

for loop

```
string = 'Python'
for s in string :
    print(s)
```

```
P
y
t
h
o
n
```

while loop

```
counter = 0
while counter < 5 :
    print(counter)
    counter += 1
```

```
0
1
2
3
4
```

Nested Loops

A nested loop is a loop inside the body of the outer loop.

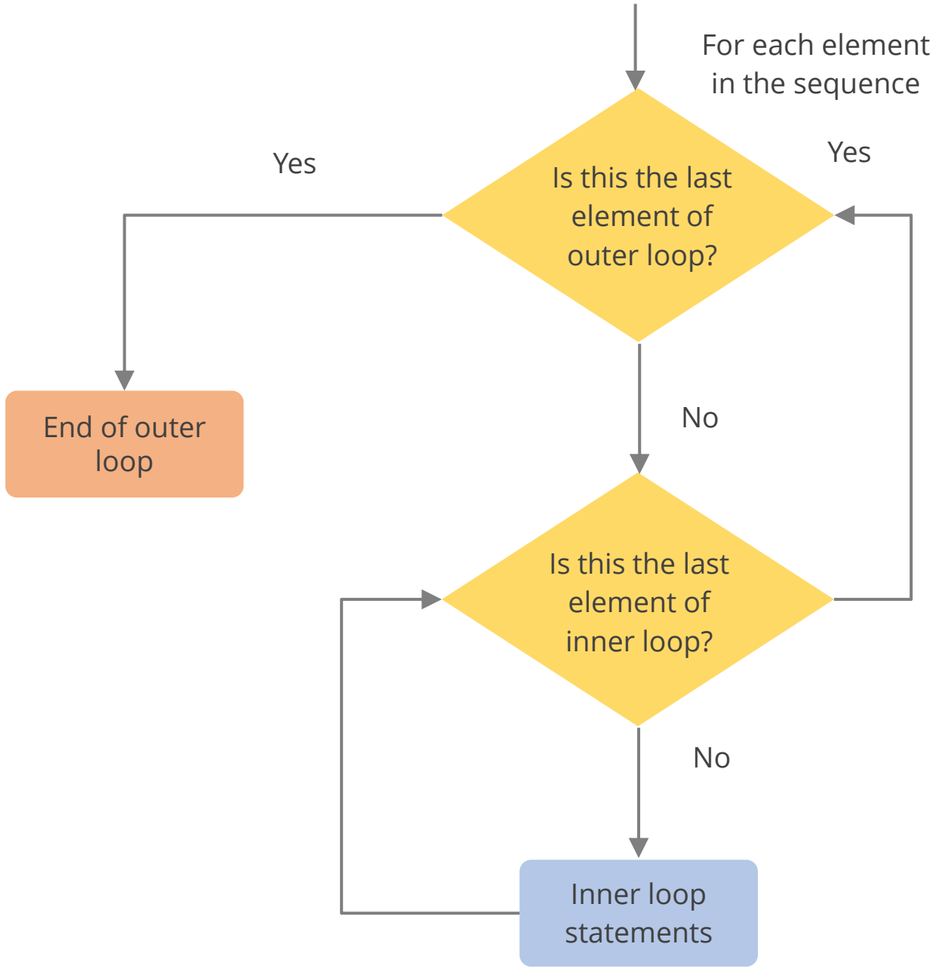
Syntax

```
# outer loop
for element in sequence :
    outer loop statements
    # inner loop
    for element in sequence :
        body of inner loop
    additional outer loop statements
```

The inner and outer loops can be of different or the same type.

Nested Loops: Flowchart

A nested loop is demonstrated in the following flowchart:



Nested Loops: Example

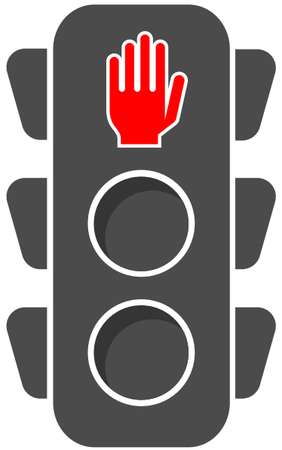
Here is the code to print the multiplication table from 2 to 10:

```
# outer loop
for i in range(2, 11):
    # nested loop
    # to iterate from 1 to 10
    for j in range(1, 11):
        # print multiplication
        print('{:2d} X {:2d} = {:2d}'.format(i,j, i*j))
    print('End of multiplication table of ', i, '\n')
```

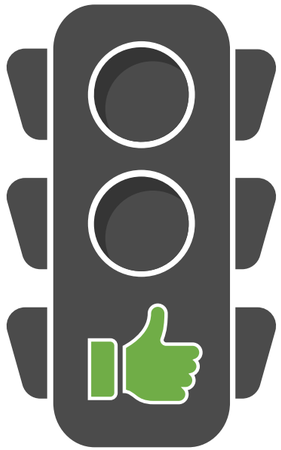
Loops Control Statements

Loops Control Statements

Loop control statements alter the flow of execution in loops. Python supports two such statements.



Break



Continue

Loops Control Statements: Break

Syntax

```
break
```

- The **break** statement exits the innermost enclosing for or **while** loop.
- It terminates the nearest enclosing loop and skips the optional **else** block.
- If a loop is terminated by a **break**, the loop variable retains its current value.

Break: Example

Example

```
# Use of break statement inside the loop

for i in "Hello string":
    if i == "l":
        break
    print(i)

print("End of Loop")
```

```
H
e
End of Loop
```

Loops Control Statements: Continue

Syntax

```
continue
```

- The **continue** statement skips the current iteration and proceeds with the next one.
- It does not terminate the loop but moves control to the next iteration.
- As a result, the optional **else** block of the loop still executes.

Continue: Example

Example

```
# Use of continue statement inside the loop

for i in "Hello string":
    if i == "l":
        continue
    print(i)

print("End of Loop")
```

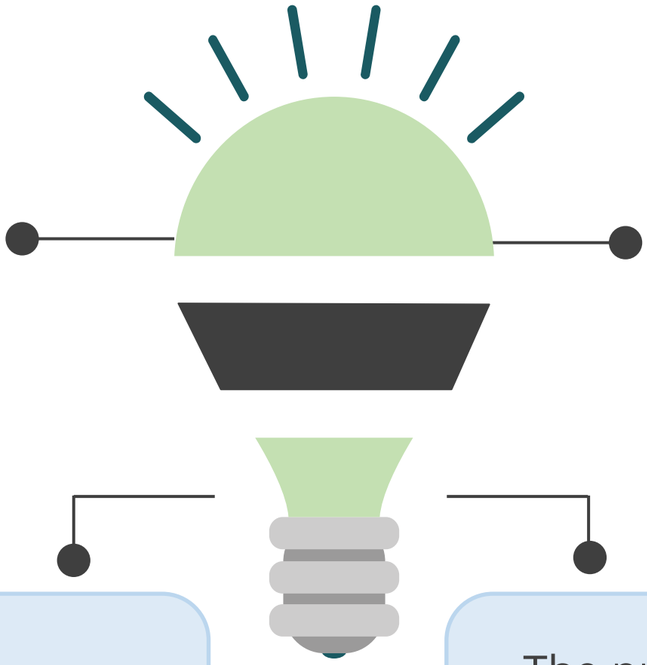
```
H
e
o

s
t
r
i
n
g
End of Loop
```

Loop Else Statements

Loop Else Statement

Python allows the else keyword to be used with both the for and while loops.



The statements of the **else** block are executed after all the iterations are completed.

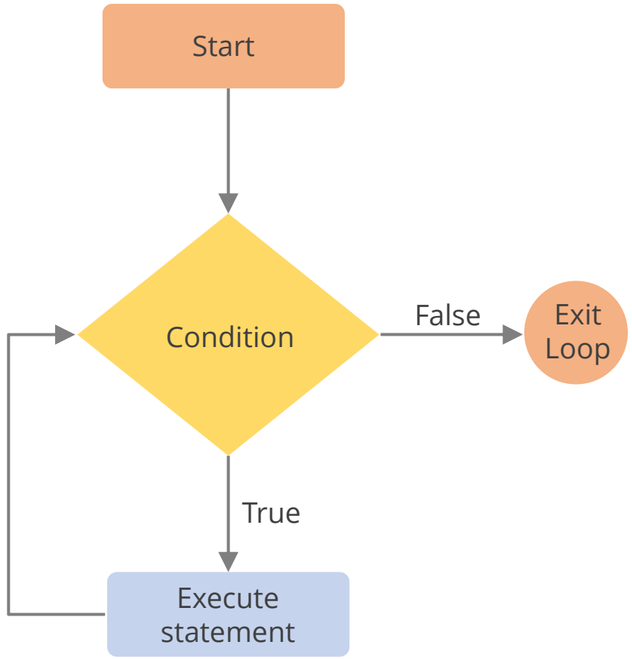
The **else** clause is defined after the body of the loop.

The program exits the loop only after the **else** block is executed.

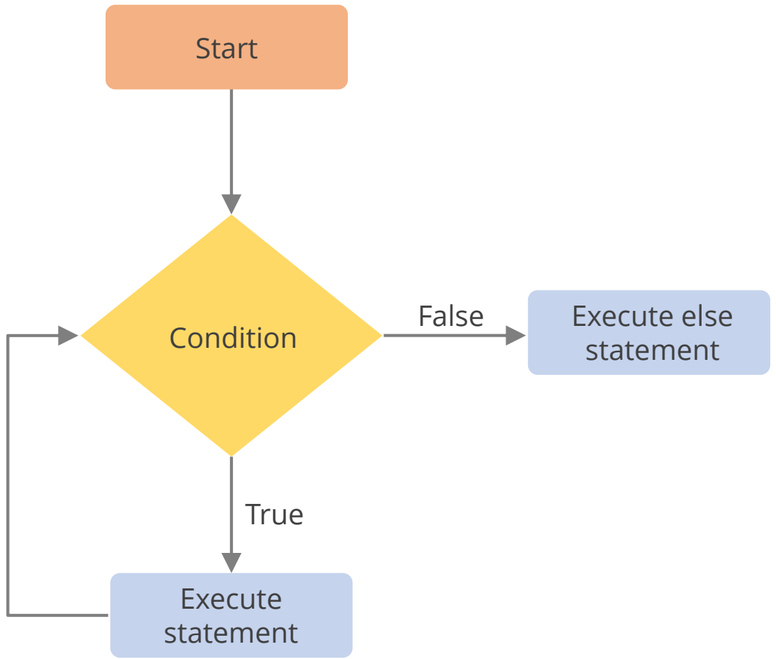
Loop Else Statement

The loop **else** statement does not execute if the loop terminates due to a **break** statement.

Normal Loop



Loop with Else



For Else Statement: Example

Example

```
numbers = [1, 2, 3, 4, 5, 6, 7]
for num in numbers:
    if num == 6:
        print("Number found!")
        break
else:
    print("Number not found!")
```

```
Number found!
```

While Else Statement: Example

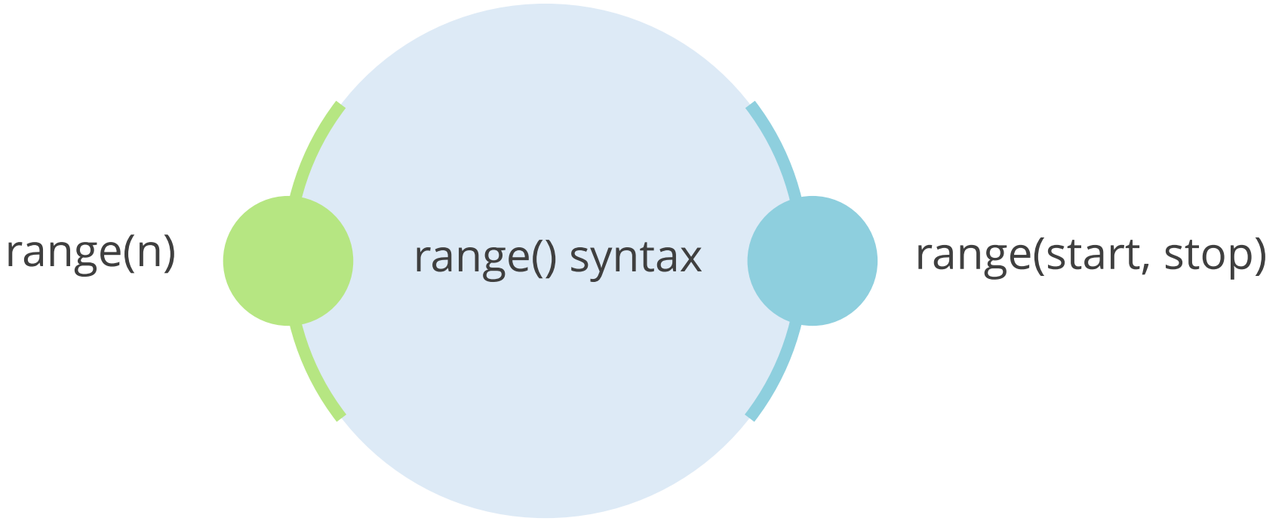
Example

```
count = 0
while count < 5:
    print("Count:", count)
    if count == 3:
        print("Count reached 3!")
        break
    count += 1
else:
    print("Loop completed!")
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count reached 3!
```

Range Function

The built-in **range** function can be used with loops to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.



The range object can be converted to a sequence collection using functions such as list and tuple.

Range(n) Function

It generates a sequence of n integer numbers starting from 0 and ending with (n-1).

Example

```
print(list(range(10)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Range(start, stop) Function

It generates a sequence of integers starting with the start value and ending with (stop-1).

Example

```
print(list(range(15, 25)))
```

```
[15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

Step in Range

The **range()** function has an optional step argument that specifies the increment of the sequence.

Syntax

```
range(start, stop, step)
```

The default increment is 1. The increment can be positive or negative, but not zero.

Step in Range: Example

A positive step value creates a forward sequence.

Example

```
print(list(range(2, 20, 2)))
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

A negative step value creates a reverse sequence.

Example

```
print(list(range(20, 2, -2)))
```

```
[20, 18, 16, 14, 12, 10, 8, 6, 4]
```

Assisted Practice: Conditional Statements and Loops

Duration: 10 mins

Objective: In this demonstration, you will learn to use the conditional statements and loops

Steps to perform:

Step 1: Demonstrate conditional statements using if, elif, and else

Step 2: Demonstrate loops using for and while

Note

Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.

ASSISTED PRACTICE

Key Takeaways

- If-else conditional constructs control the program's flow.
- For and while loops repeatedly execute statements.
- The break and continue statements skip certain statements inside the loop or terminate the loop immediately without checking the condition.
- Python supports the else clause with for and while loops.
- The range function in Python generates a range of numbers and works with the for loop.

Knowledge Check

Knowledge Check

1

Which of the following is used to terminate a loop early in Python?

A. break

B. end

C. stop

D. exit

Knowledge Check

1

Which of the following is used to terminate a loop early in Python?

A. break

B. end

C. stop

D. exit

The correct answer is **A**

In Python, the break statement is used to exit a loop early when a certain condition is met.

Knowledge Check

2

Which one of the following is a valid Python if statement?

A. if a>=2 :

B. if (a>=2)

C. if a>=22

D. if a=>22

Knowledge Check 2

Which one of the following is a valid Python if statement?

A. if a>=2 :

B. if (a>=2)

C. if a>=22

D. if a=>22

The correct answer is **A**

In Python, an if statement always ends with a colon.

Knowledge Check 3

Where can the continue statement be used?

A. while loop

B. for loop

C. do-while loop

D. Both A and B

Knowledge Check

3

Where can the continue statement be used?

A. while loop

B. for loop

C. do-while loop

D. Both A and B

The correct answer is **D**

The continue statement can be used in both while and for loops.

Thank You!