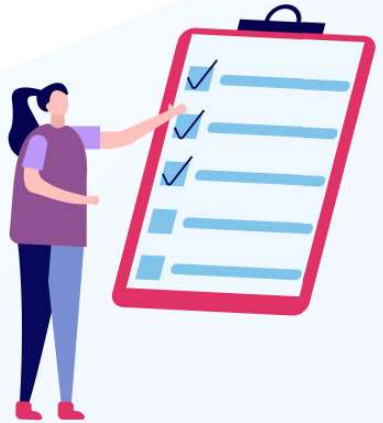
Conditional Statements and Loops

Learning Objectives

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

- Implement decision control structures in Python
- Learn different types of loops in Python
- Learn to implement loop control statements
- Describe and use the range function in Python



Business Scenario

ABC is an e-commerce organization that deals with the listing of online products based on their purchase, reviews, and availability. The users filter these products based on multiple factors. The products are listed only if it matches a few criteria, else an error is displayed.

The organization is supposed to develop this module with the help of control structure, implement range functions to filter based on the ranges, and implement looping control statements.

In this lesson, we will explore the following:

- Decision control structures in Python
- Loops, loop control structures, and loop-else statements in Python

Decision Control Structures in Python

Discussion

Decision Control Statements

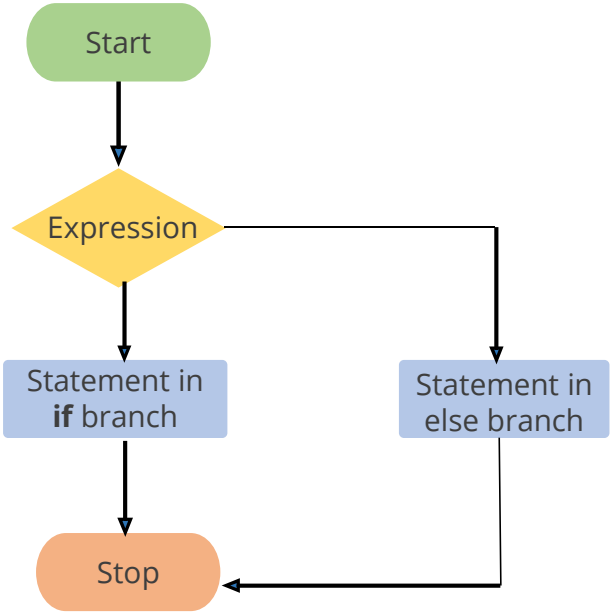
How can decisions be made in Python?

- What is decision-making in Python?
- What are the types of decision-making statements and why they are needed?

Decision Making

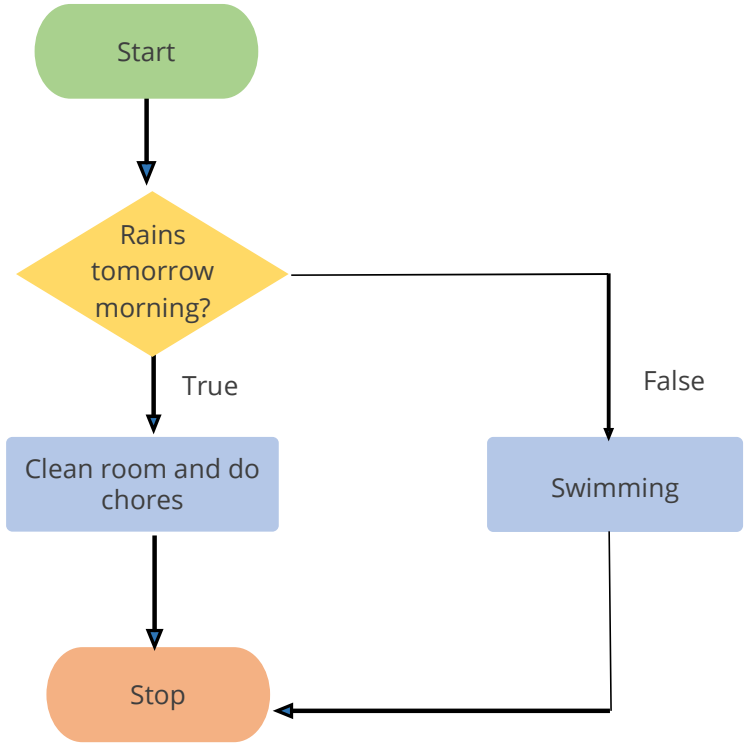
Decision-making is the process of making choices or performing tasks based on conditions.

Decision control structures evaluate variables or expressions that return either True or False as an outcome.



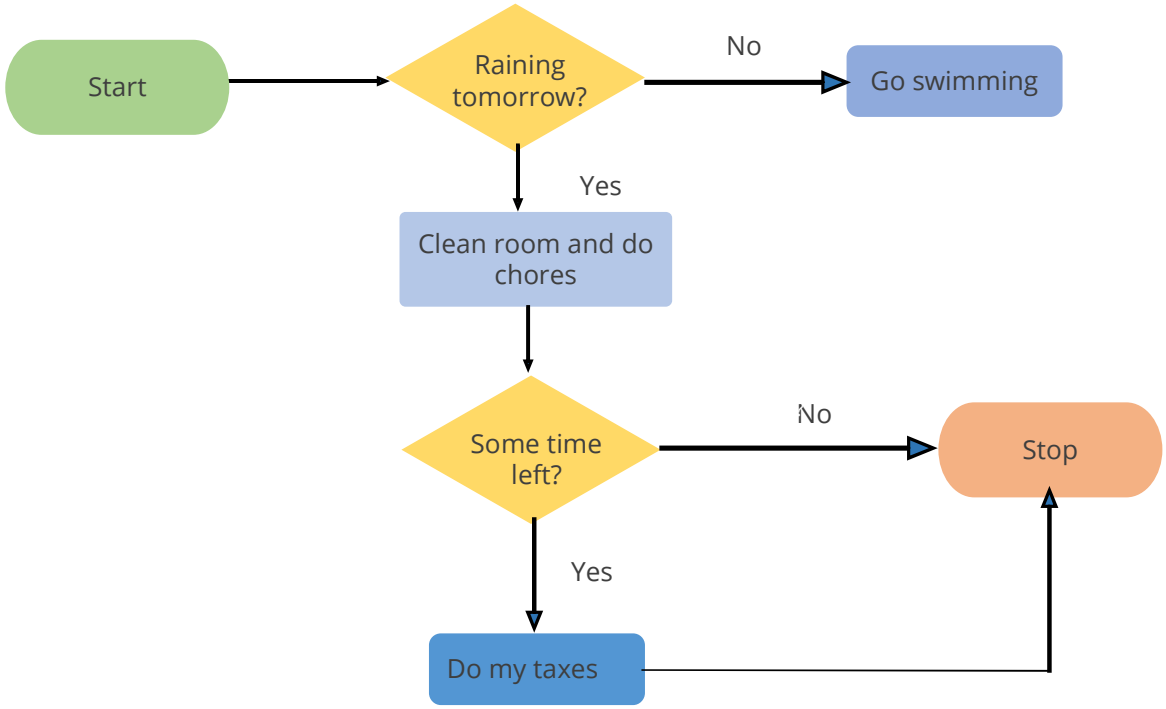
Decision Control Structures: Scenario

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



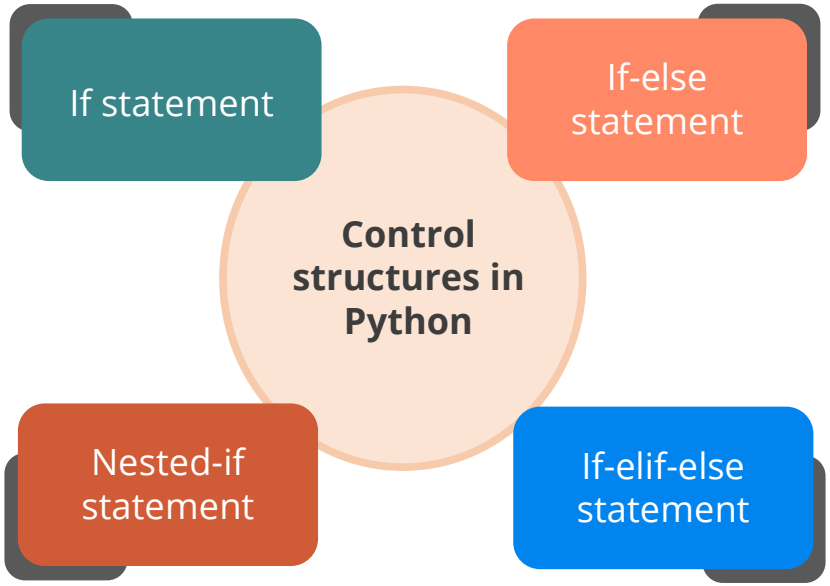
Nested if Statements

The scenario is explained in the following flowchart:



Decision Control Structures

Four types of decision control structures are available in Python:



If Statement

Python uses the if statement to change the flow of control in the program.
The indentation is used to mark the block of code.

Syntax

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

- A colon marks the beginning of the block of code.
- The block of code is indented, usually at four spaces.
- Each statement in the block of code should be at the same indentation.

If Statement: Example

The following example illustrates the use of the if statement.

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 body of *if* only when the test condition is True. Otherwise, the body of the *else* block will be executed.

Syntax

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

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

If-Else Statement: Example

The following example illustrates the use of the if-else statement.

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 for multiple conditions. If the condition for if is False, it checks the condition of the next elif block and so on.

Syntax

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

Only one block among the several if-elif-else blocks is executed according to the condition. If all the conditions are False, the body of else is executed.

If-Elif-Else Statement: Example

The following example illustrates the use of the if-elif-else statement.

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 in programming.
- The level of nesting is defined by using indentation.

Nested-If: Example

The following example illustrates the use of the nested-if statement.

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.

Decision Control Statements

How can decisions be made in Python?

- What is decision-making in Python?

Answer: Decision-making is the process of making choices or performing tasks based on conditions. Decision control structures and evaluate variables or expressions that return either *True* or *False* as an outcome.

- What are the types of decision-making statements and why are they needed?

Answer: The different decision-making statements are the If statement, If-else statement, If-elif-else statement, and nested-if statement.

Loops

Discussion

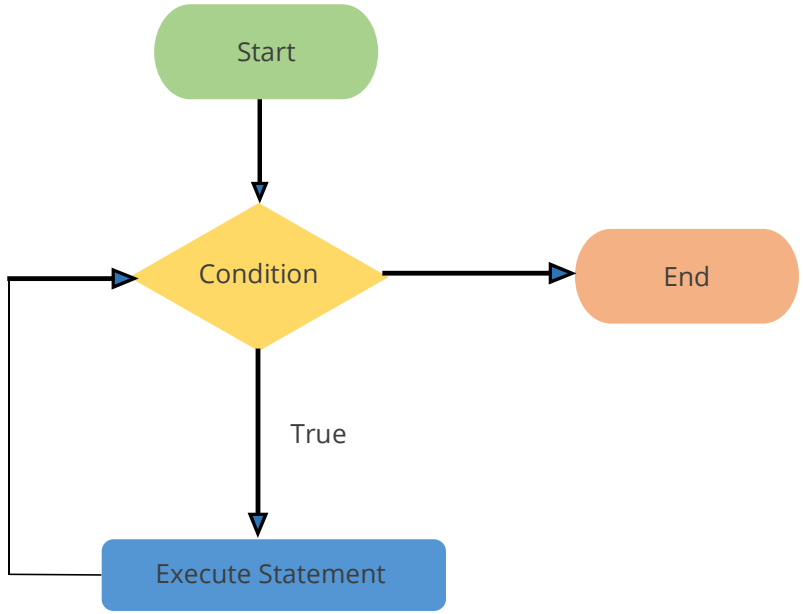
Loops

Are decision-making statements and loops the same?

- What are loops in Python?
- Identify the types of loops

Loops

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



Types of Loops

The following types of loops are used to handle looping requirements.

Count controlled loop

Condition controlled loop

Collection controlled loop

Count Controlled Loop

A method of repeating a loop a predetermined number of times

Syntax

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

Condition Controlled Loop

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

Syntax

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

Collection Controlled Loop

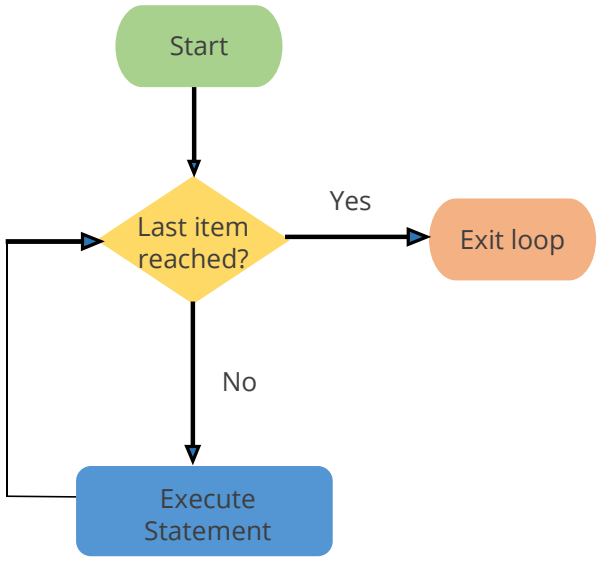
This is a special construct that allows looping through the elements of a "collection," which can be an array, list, or other ordered sequences.

Syntax

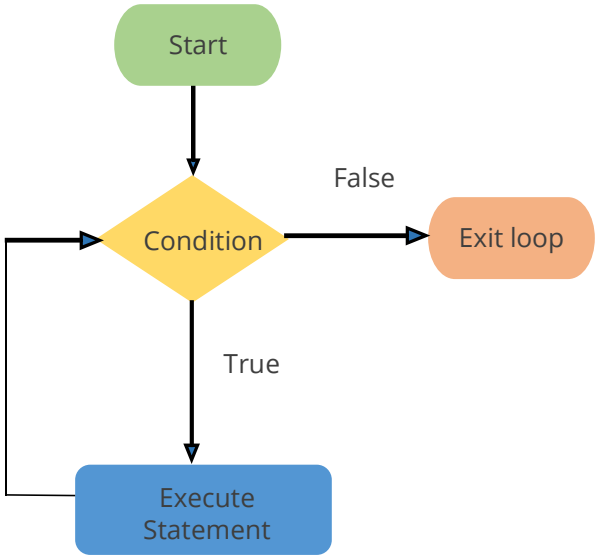
```
for <item> in <list>:
    body of loop
```

Loops in Python

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



For loop



While loop

Loops in Python

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

for loop

The for loop is used to iterate 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 is used to iterate over a block of code if the test expression is true. Its syntax is:

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

Loops in Python: Example

The following examples illustrate the use of **for** and **while** loops.

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.

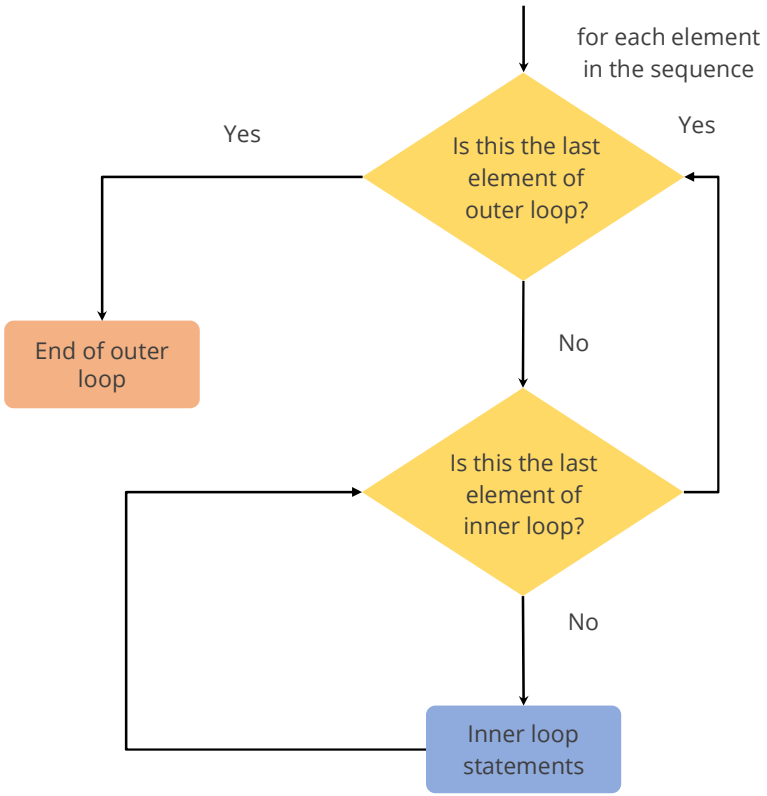
Syntax

```
# 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 the different or the same type.

Nested Loops: Flowchart

A nested loop is demonstrated in the following flowchart.



Nested Loops: Example

The following example illustrates the use of the nested loop.

Code to print 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

Are decision-making statements and loops the same?

- What are loops in Python?

Answer: A loop statement allows the execution of a statement or group of statements multiple times.

- Identify the types of loops

Answer: The types of loops include the count-controlled loop, the condition-controlled loop, and the collection-controlled loop.

Loops Control Statements

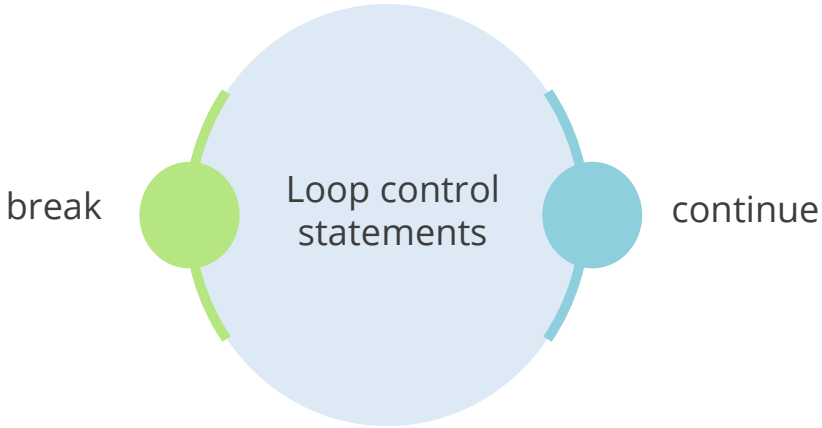
Discussion

Loop Control Statements

- What is the difference between break and continue?
- Give an example or scenario where break and continue can be used.

Loops Control Statements

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



Loops Control Statements: Break

Syntax

```
break
```

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

Break: Example

The following example illustrates the use of a **break** statement.

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 continues with the next iteration.
- It does not terminate the loop; it just moves the control to the next iteration.
- Since the loop does not terminate unexpectedly, it executes the optional **else** block of the loop.

Continue: Example

The following example illustrates the use of the continue statement.

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 Control Statements

- What is the difference between break and continue?

Answer: The break statement breaks the innermost enclosing of for or while loop, whereas the continue statement skips the current iteration and continues with the next.

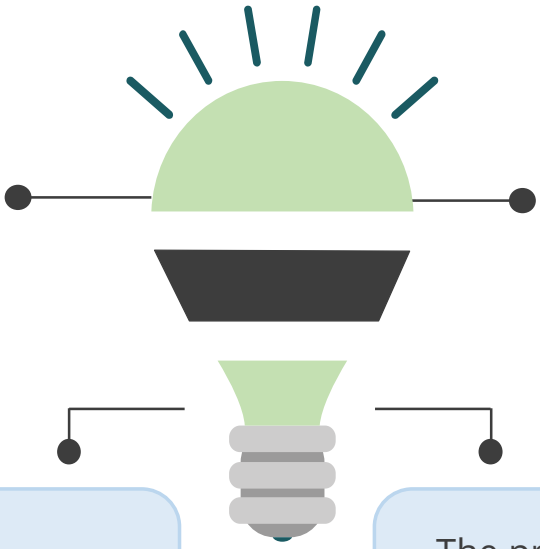
- Give an example or scenario to define where each of them can be used.

Answer: In case the optional else has to be used, the continue statement can be used as it executes the optional else block of the loop, whereas, in another case, the break statement can be used.

Loop Else Statements

Loop Else Statement

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



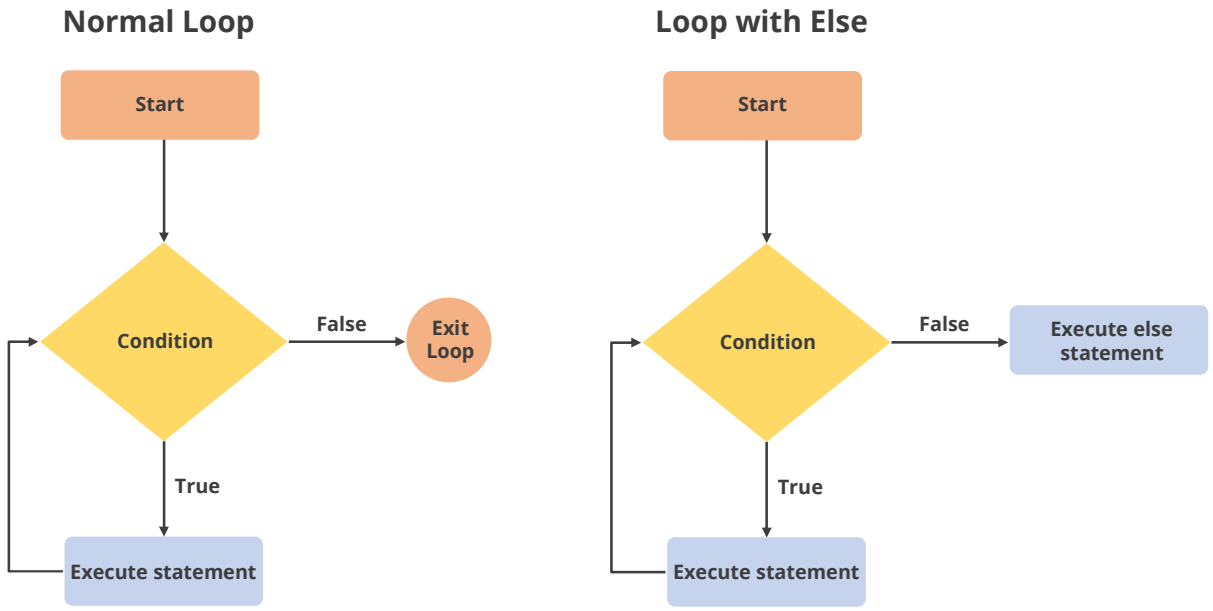
The statements of the **else** block are executed after all the iterations are completed.

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 is not executed when the loop is terminated because of a **break** statement.



For Else Statement: Example

The following example illustrates the use of a **for else** statement.

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

The following example illustrates the use of a **while else** statement.

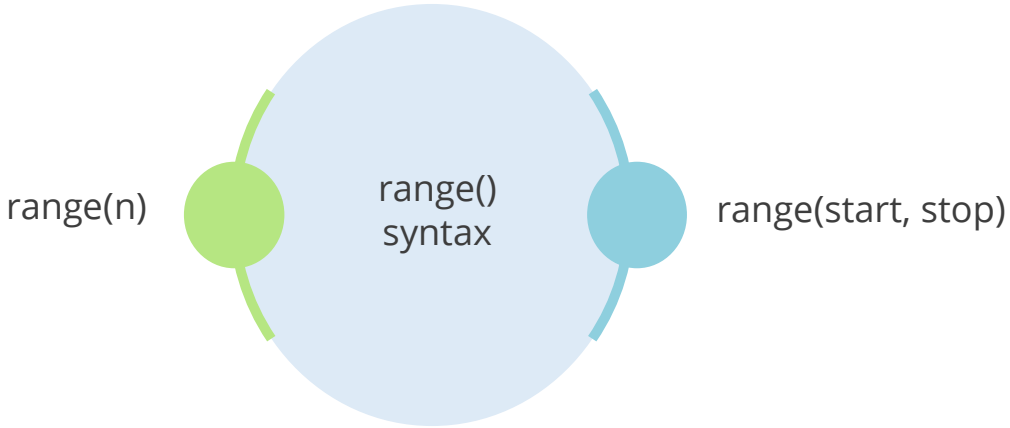
Example

```
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 function **range** 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 a function such as list and tuples.

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 integer numbers 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 additional optional step argument that specifies the increment of the sequence.

Syntax

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

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

Step in Range: Example

Positive step value creates a forward sequence.

Example

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

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

Negative step value creates a reverse sequence.

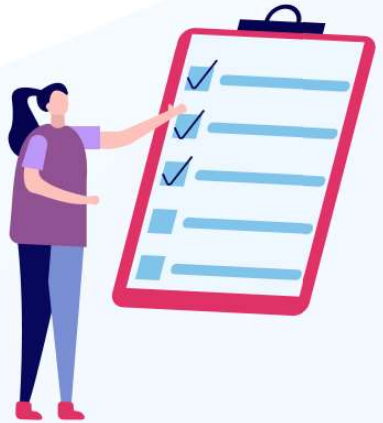
Example

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

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

Key Takeaways

- If-else conditional constructs are used to control the program's flow.
- For and while loops are used to repeatedly execute statements.
- The break and continue statements are used to skip some statements inside the loop or terminate the loop immediately without checking the condition.
- Python supports the else clause with for and while loops.
- Range function in Python is used to generate a range of numbers and it also works with for loop.



Knowledge Check

Knowledge Check 1

Which of the following is not used as a loop in Python?

A. for loop

B. while loop

C. do-while loop

D. None of the above

Knowledge Check 1

Which of the following is not used as a loop in Python?

A. for loop

B. while loop

C. do-while loop

D. None of the above

The correct answer is **C**

Python does not use the do-while as a loop.

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.