Programming Refresher

Conditional and Loop Constructs

Learning Objectives

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

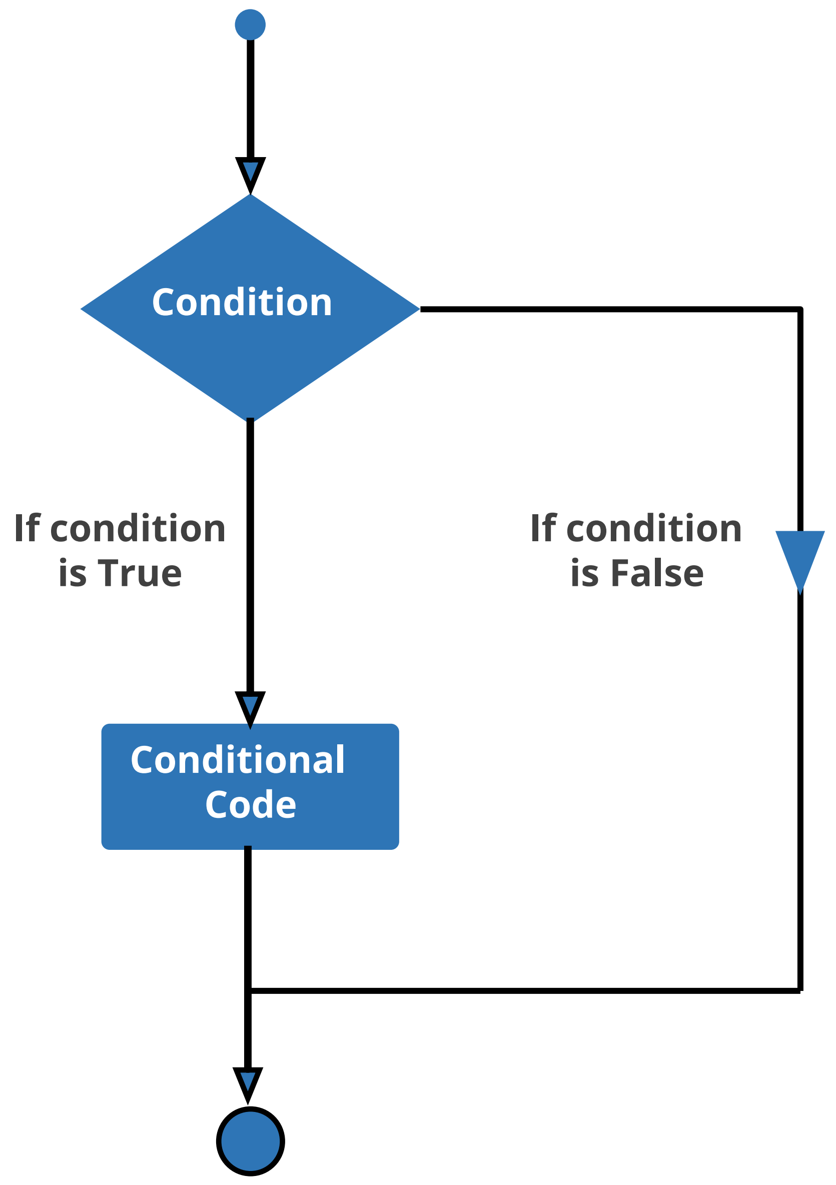
- Implement decision control structures in Python
- Learn different types of loops in Python
- Learn to implement Loop Control Statements.
- Describe and use the range function in Python

Decision Control Structures in Python

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.

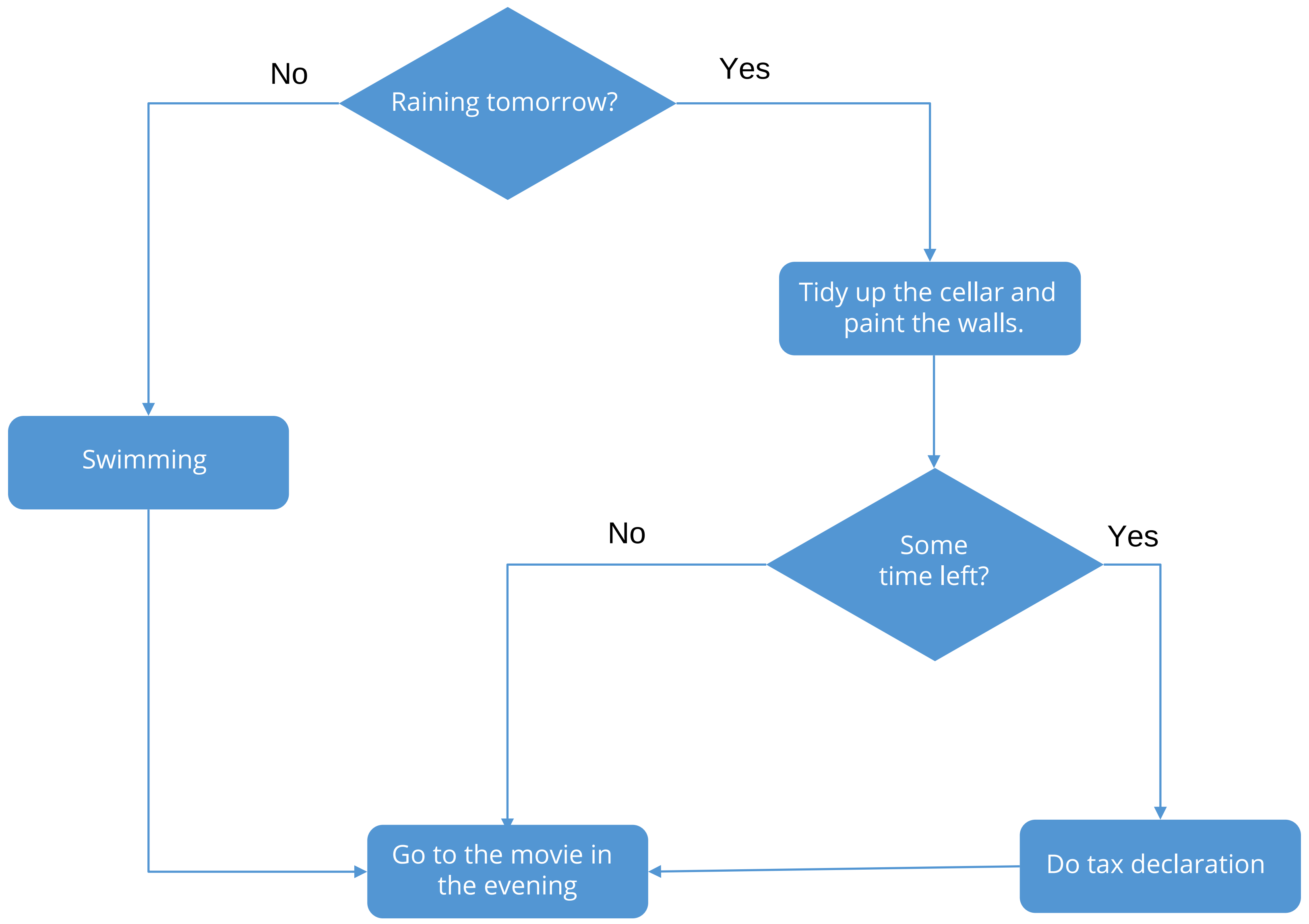


Decision Control Structures: Scenario

- Based on the conditions, a user decides on his activity throughout the day.
- If it rains in the next day morning, the user will tidy up the cellar and paint the walls; if time is left, the user will do the tax declaration and then go to a movie in the evening with friends.
- Otherwise, the user will go swimming and then go to a movie in the evening with friends.

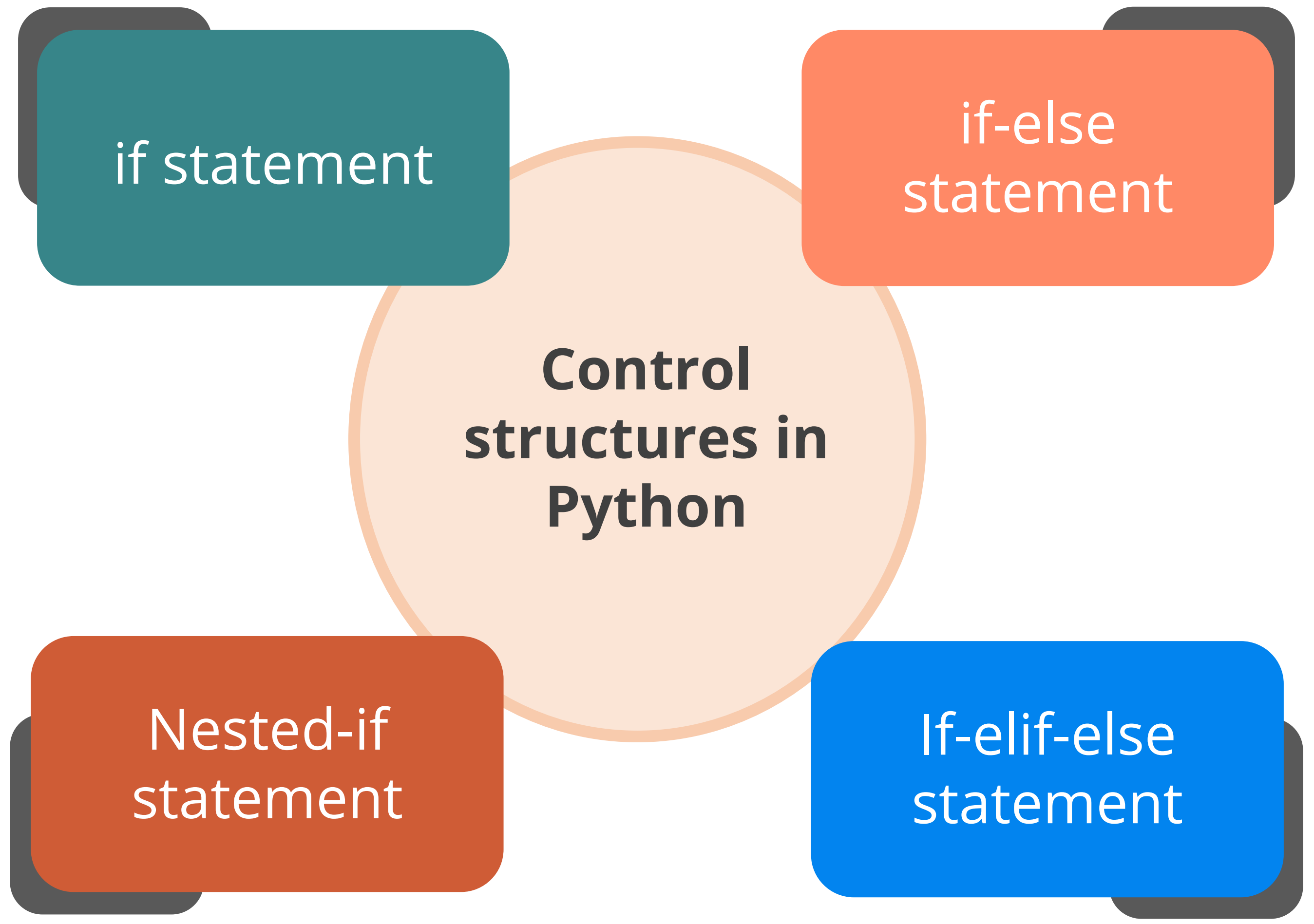
Decision Control Structures: Flowchart

The scenario is explained in the flowchart below:


No
Raining tomorrow?
Yes
Tidy up the cellar and paint the walls.
Swimming
No
Some time left?
Yes
Go to the movie in the evening
Do tax declaration

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

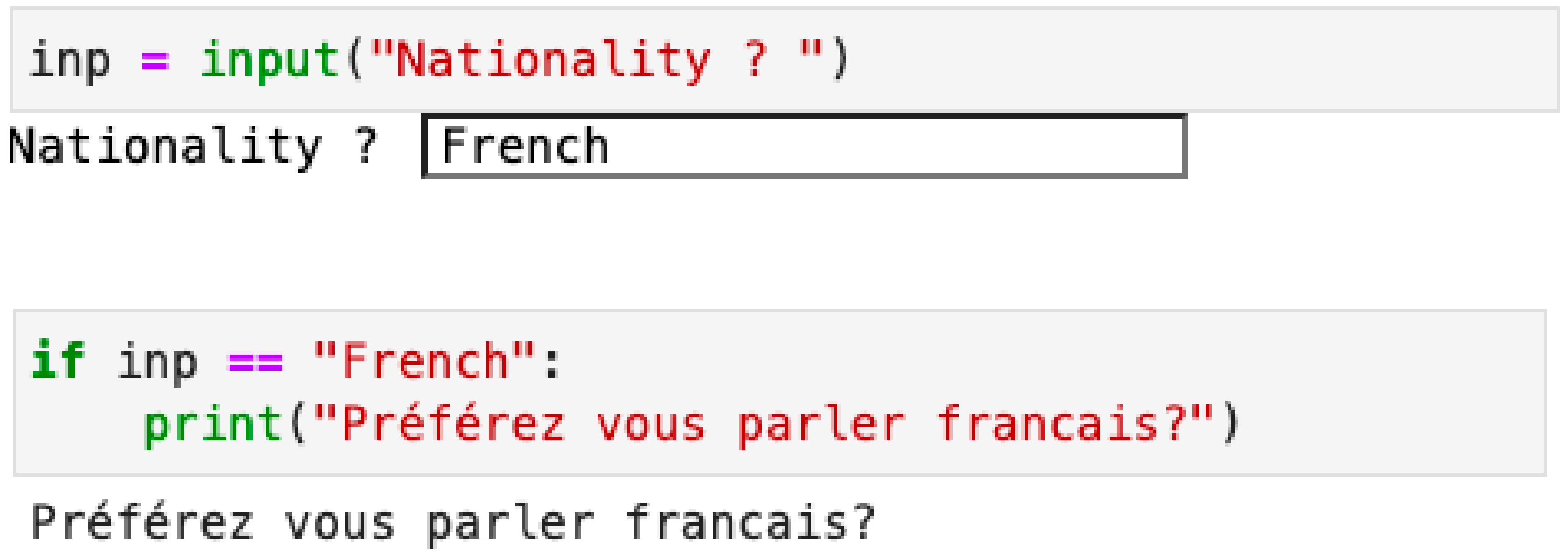
```
if condition:
    statement
    statement
    …….
```

- The colon symbol marks the beginning of the block of code.
- The block of code is indented, usually at four spaces.
- All the statements 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

It 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

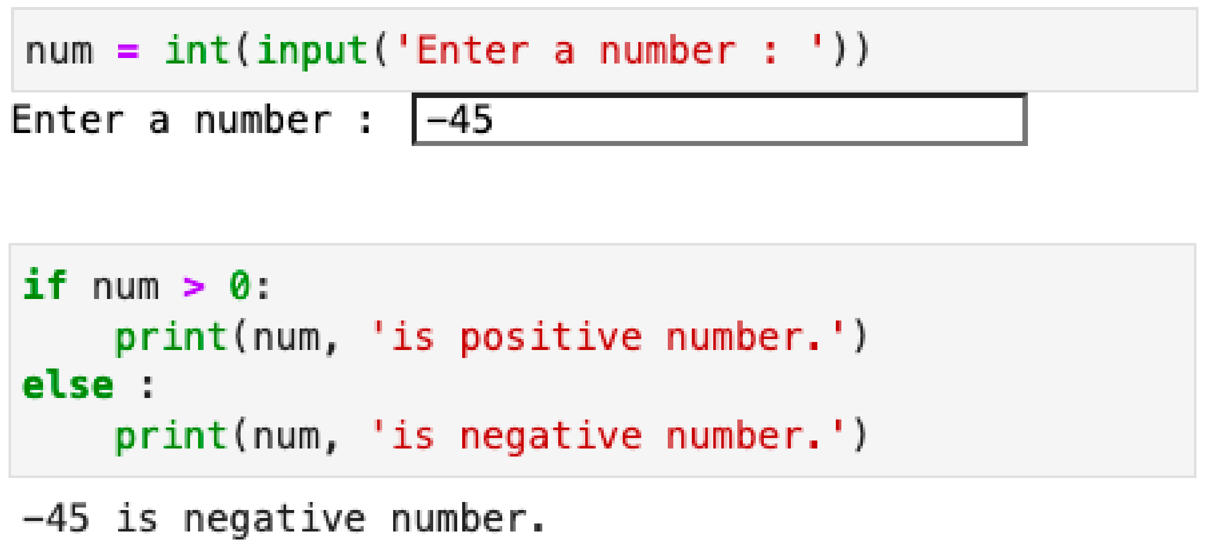
```
if condition :
   statement 1
   statement 2
else :
   statement 3
   statement 4
```

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

It 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

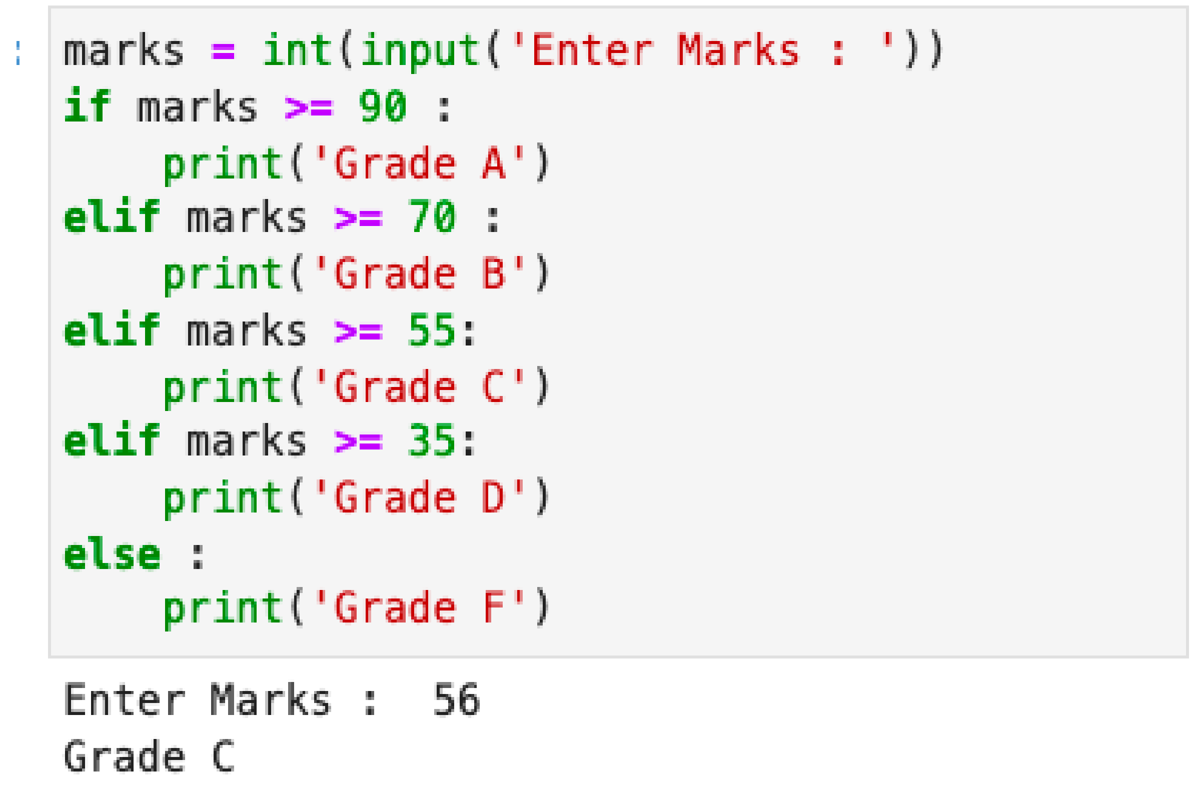
```
if condition 1 :
    statement
elif condition 2:
    statement
elif condition 3
    statement
else :
    statement
```

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

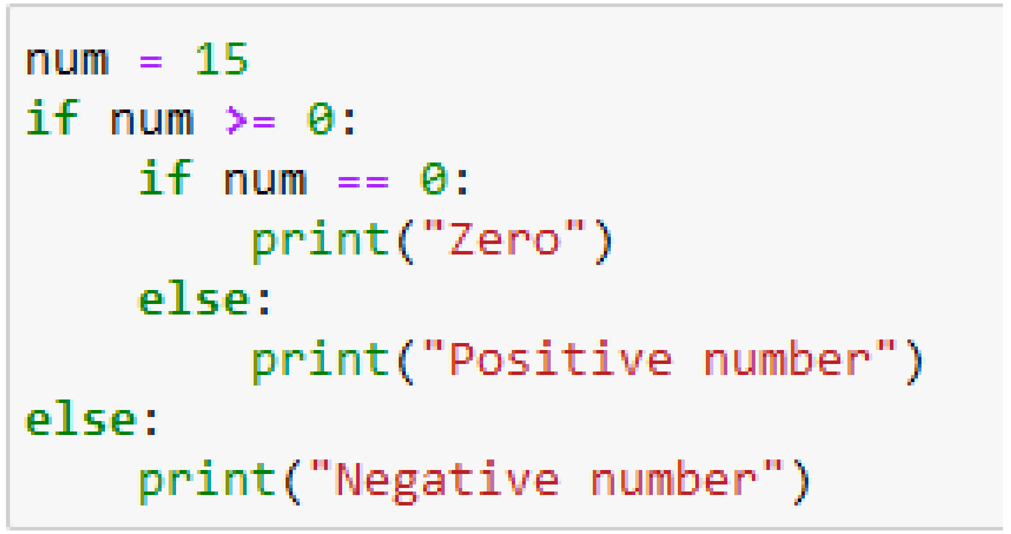
```
if (condition 1):
    statement
    # Executes when condition 1 is true
    if (condition 2):
        # Executes when condition 2 is also true
    # inner if Block is ends here
# outer if Block is ends here
```

- This is called nesting in programming.
- The level of nesting can be 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:

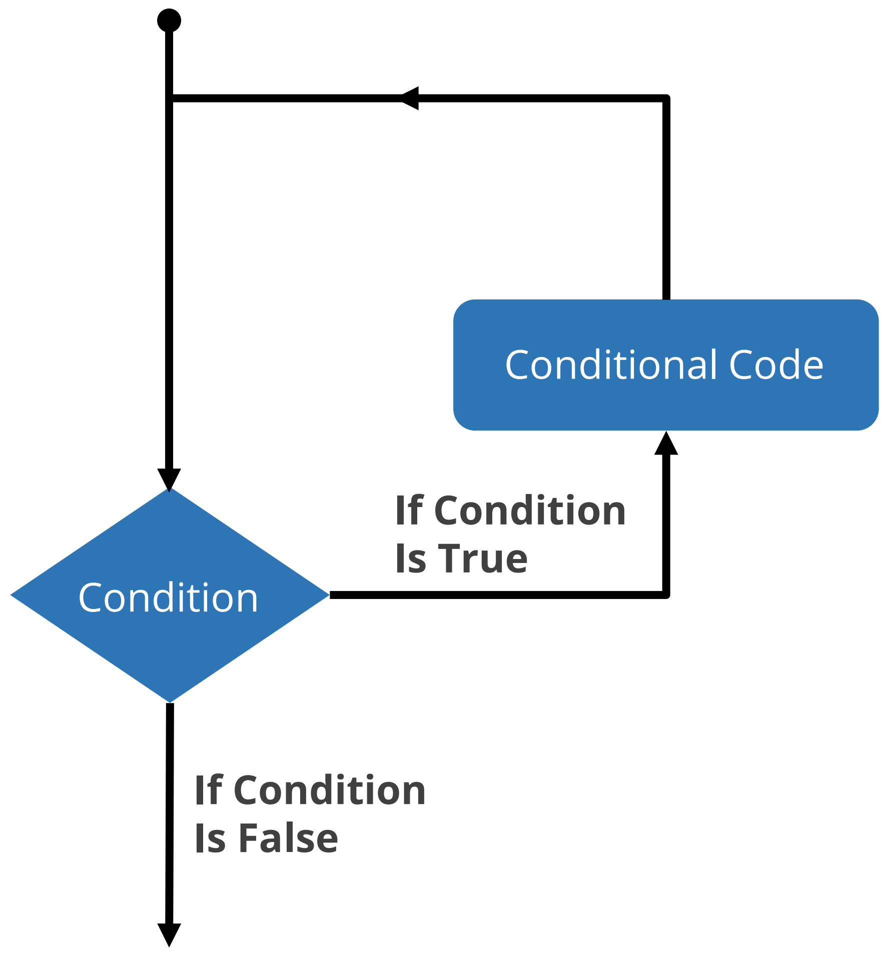
- Numerical zero values
- Boolean value False
- Empty strings
- Empty list, tuples, and dictionaries.
- None

All other values are considered True in Python.

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(I = 0; I <= 10 ; I ++) {
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 {
Body of loop
}
```

Collection Controlled Loop

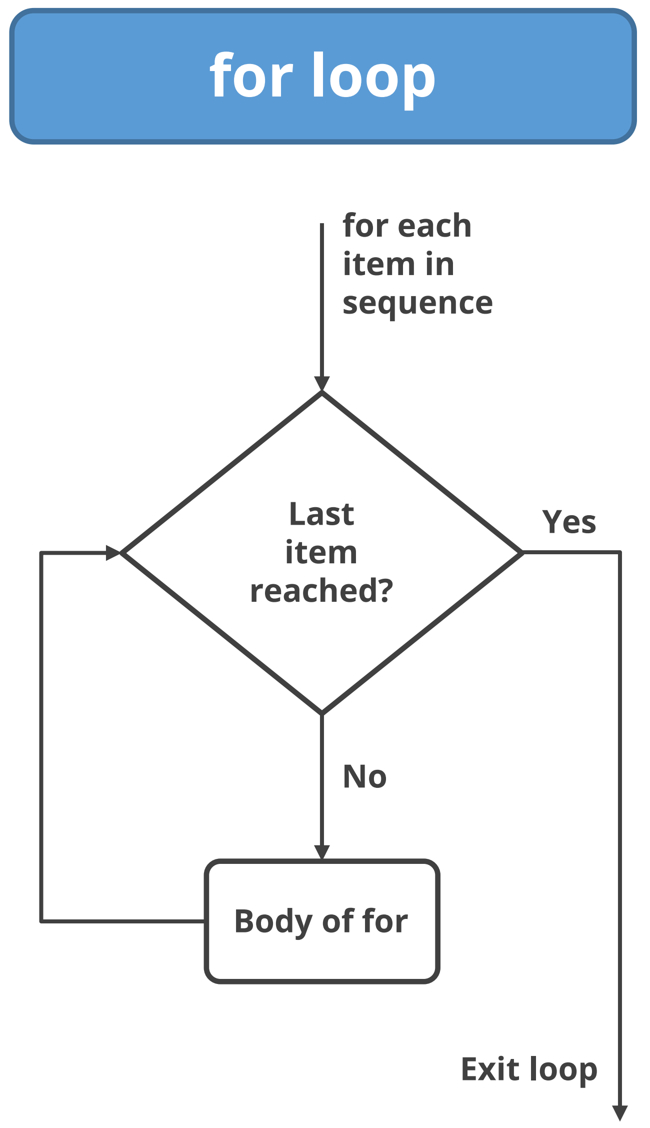
This is a special construct that allows looping through the elements of a “collection”, which can be an array, list, or other ordered sequences.

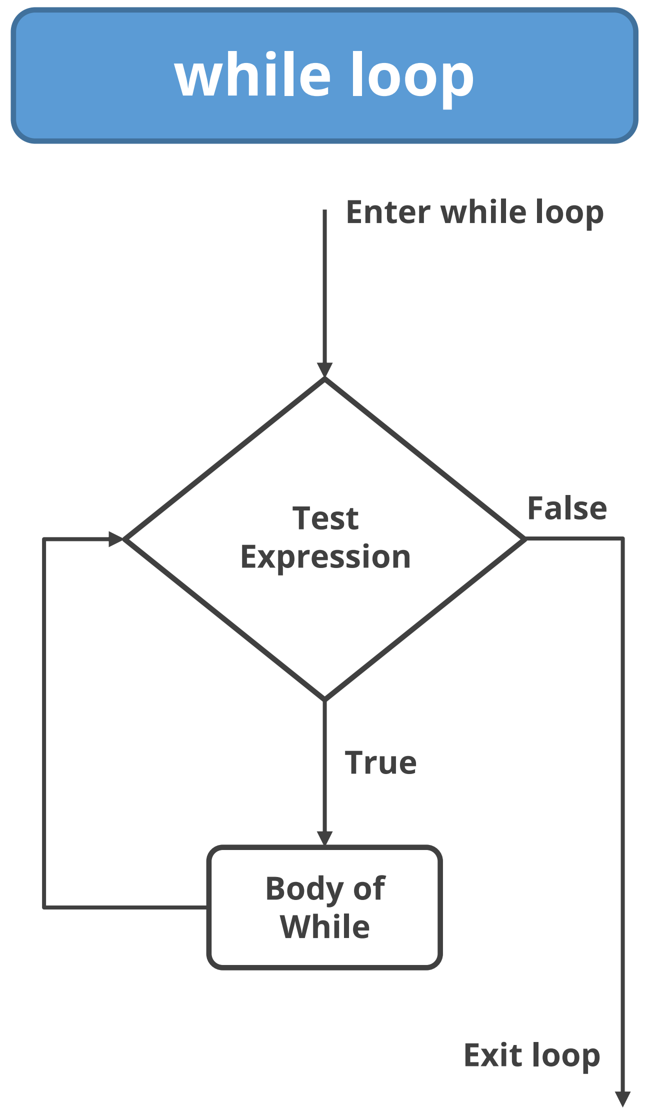
Syntax

```
for a in collection :
    body of loop
```

Loops in Python

Python supports for loop and 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

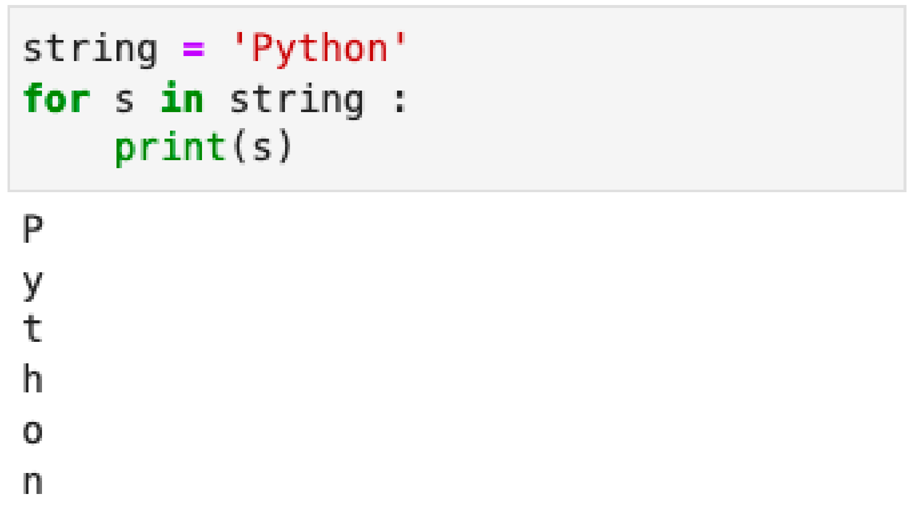
The while loop is used to iterate over a block of code if the test expression is true. Its syntax is :

```
while test_expression :
 body
 of
 loop
```

Loops in Python: Example

Examples to illustrate the use of for and while loop.

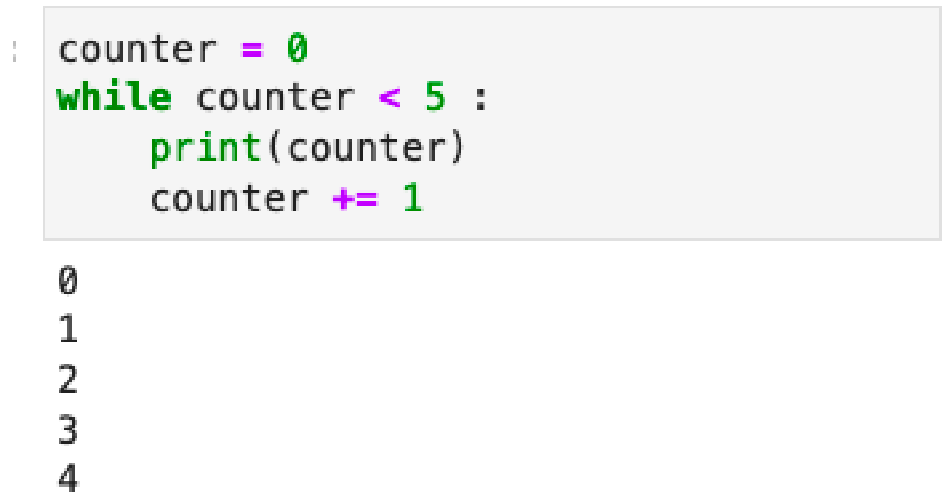
for loop



```
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.

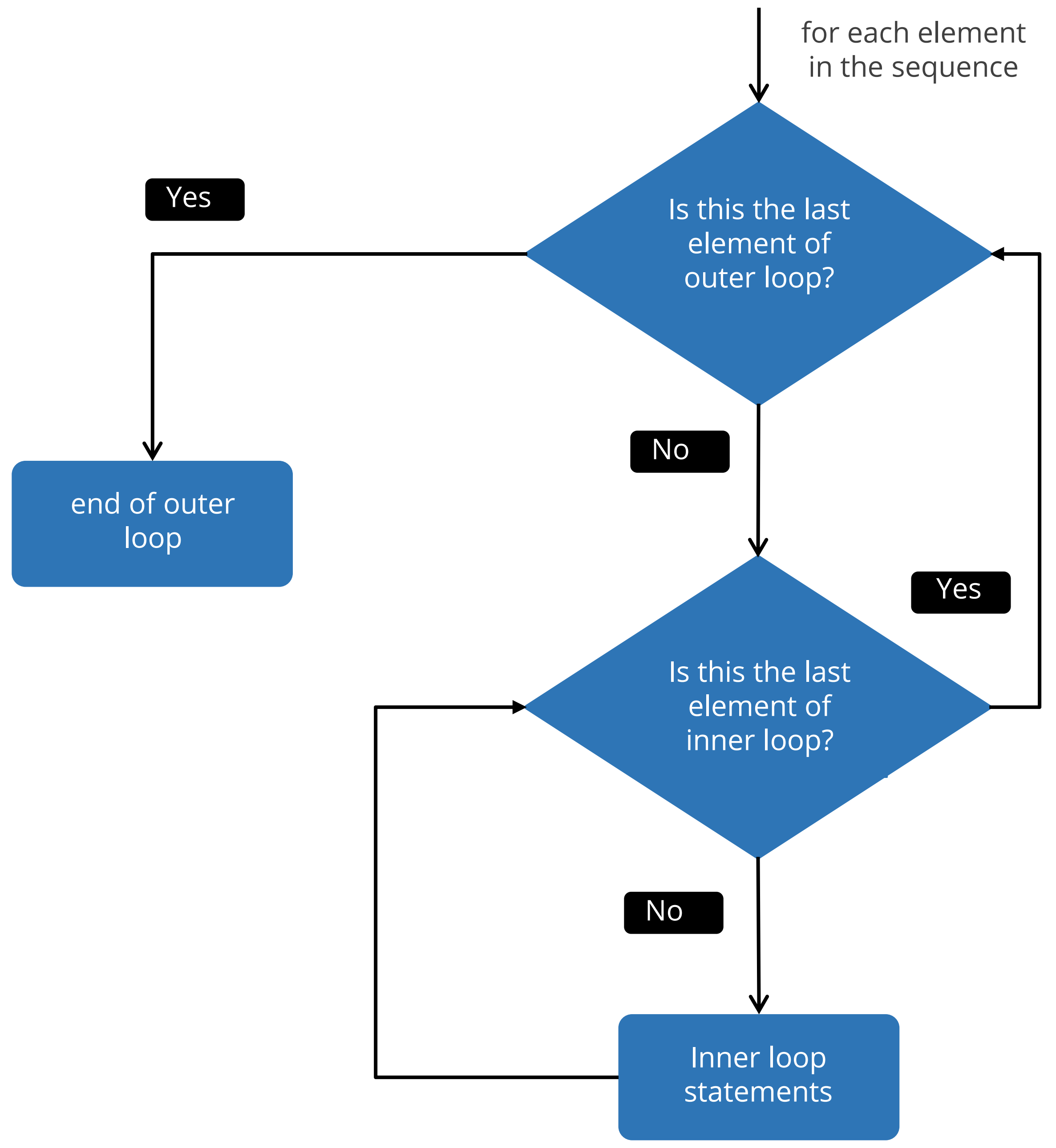
Syntax

```
# 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

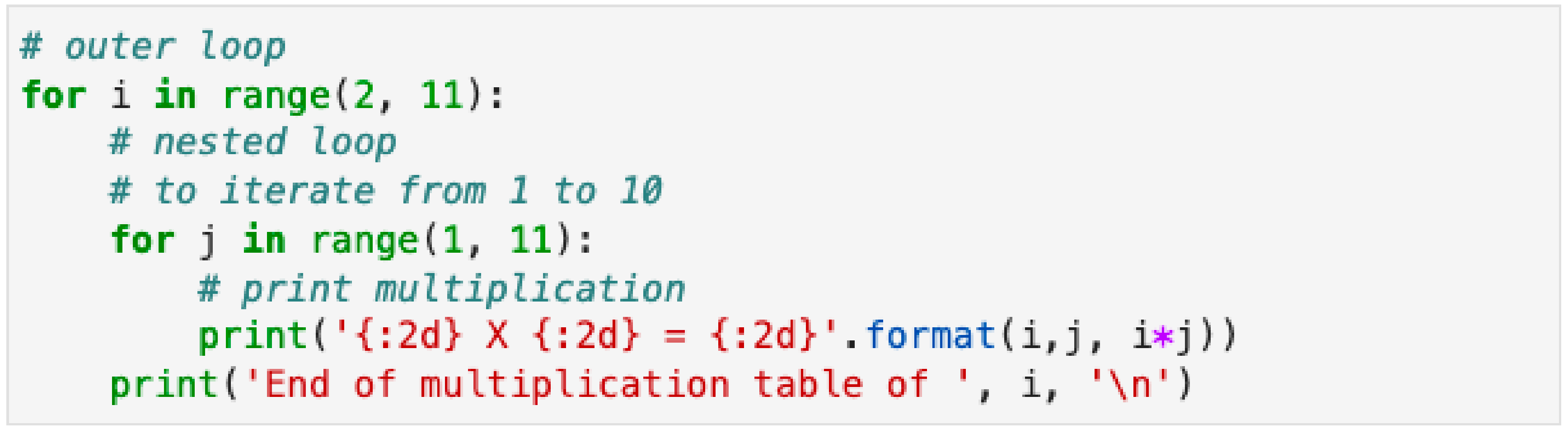
A nested loop is demonstrated in the flowchart below:



Nested Loops: Example

The following example illustrates the use of the nested loop.

Code to print multiplication table from 2 to 10

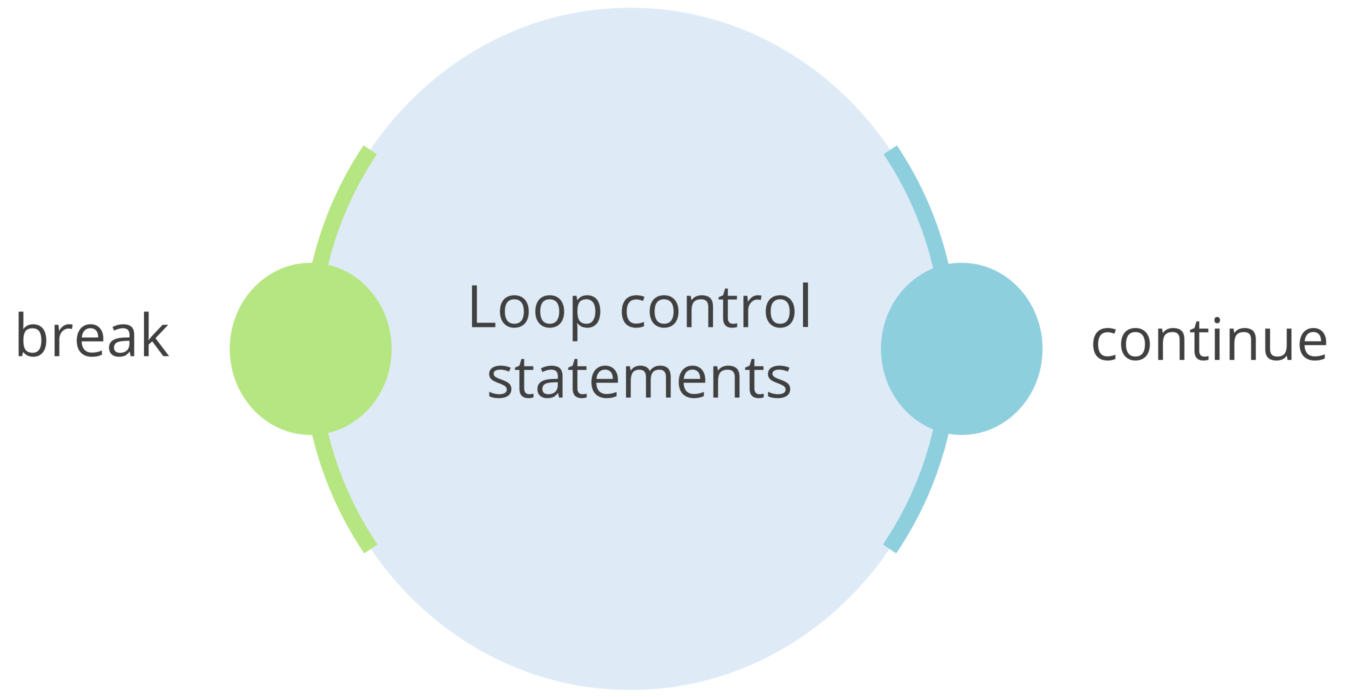


```
# outer loop
for i in range(2, 11):
    # nested loop
    # to iterate from 1 to 10
    for j in range(1, 11):
        # print multiplication
        print('{:2d} X {:2d} = {:2d}'.format(i,j, i*j))
    print('End of multiplication table of ', i, '\n')
```

Loops Control Statements

Loops Control Statements

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



Loops Control Statements: Break

Syntax

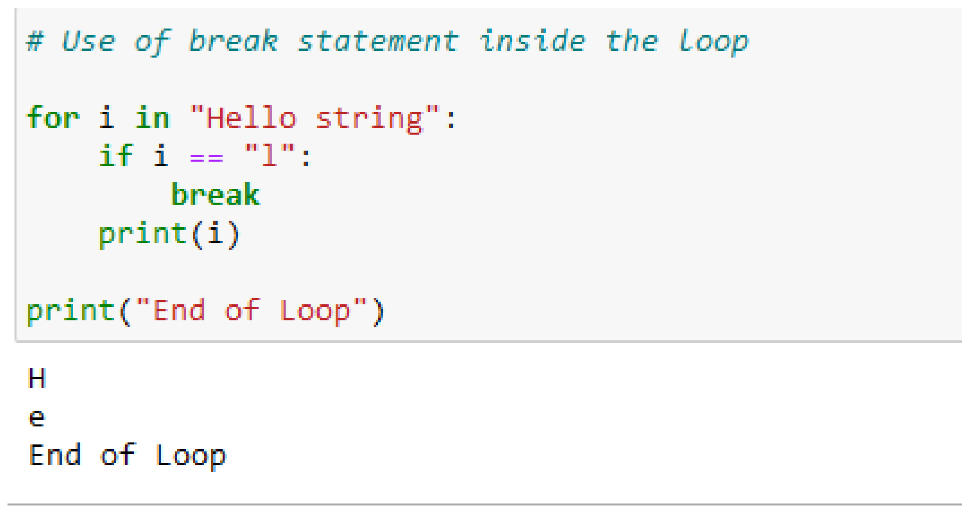
```
break
```

- 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 the 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

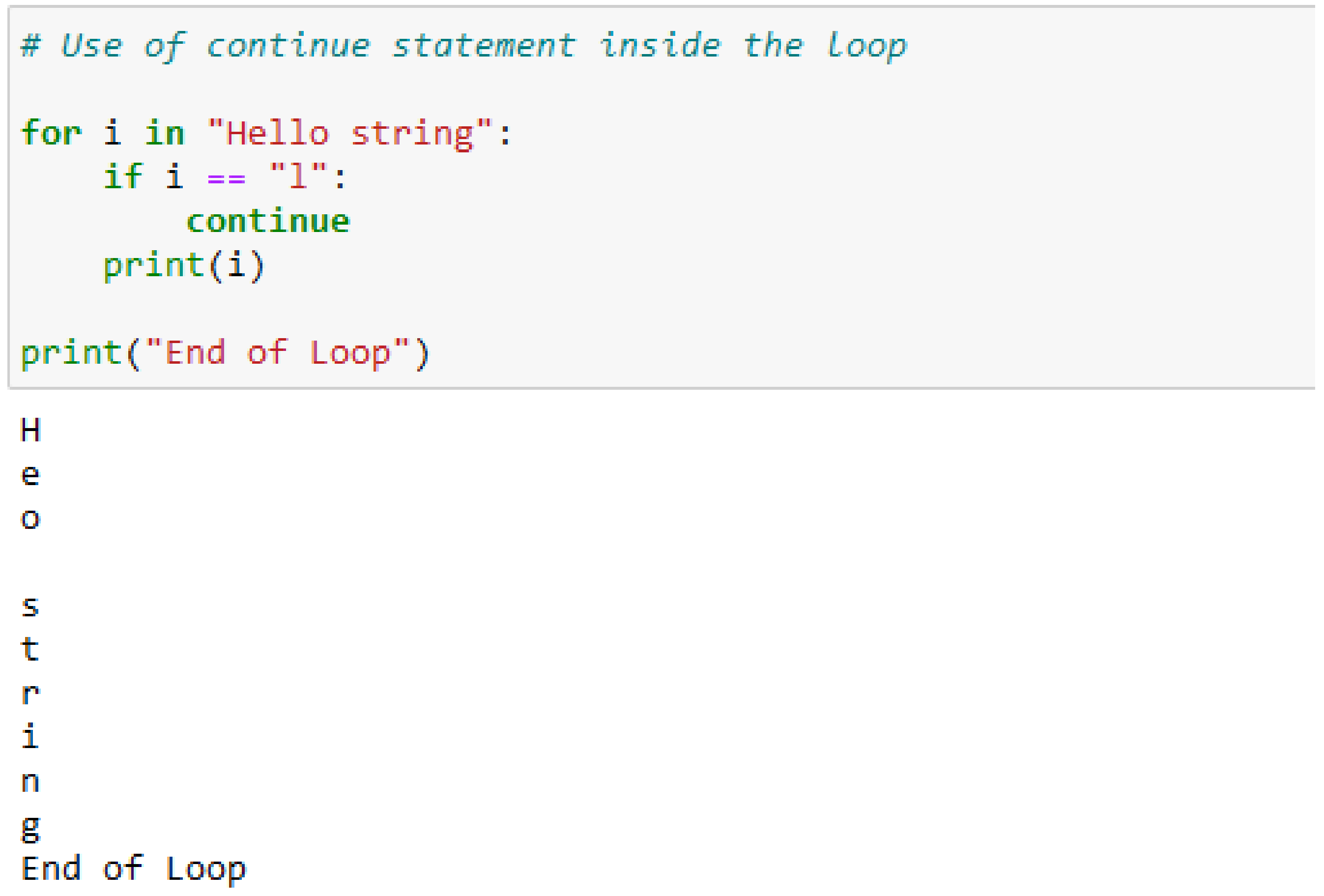
```
continue
```

- 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



```python
# 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

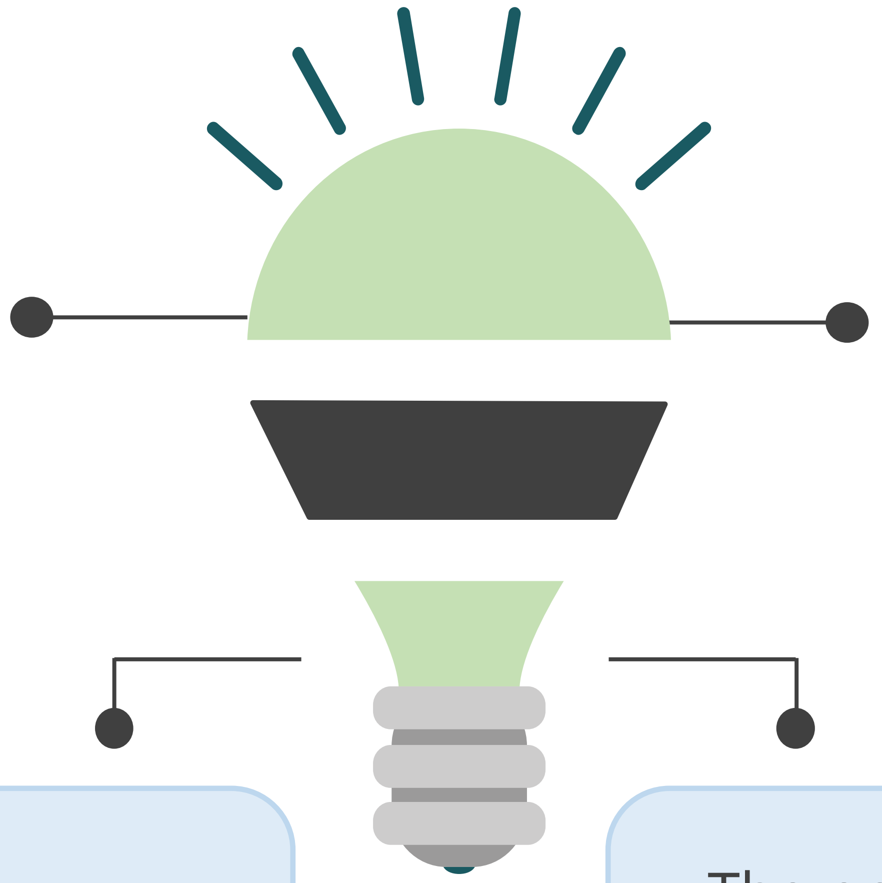
Loop Else Statement

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.

Else clause is defined after the body of the loop.

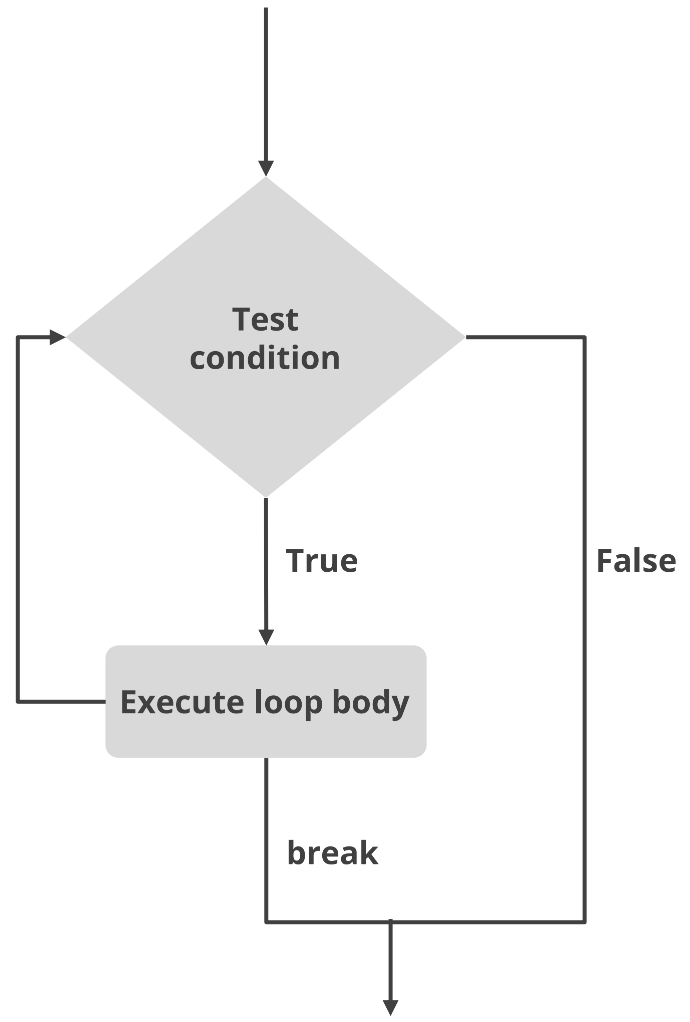
The program exits the loop only after the else block is executed.



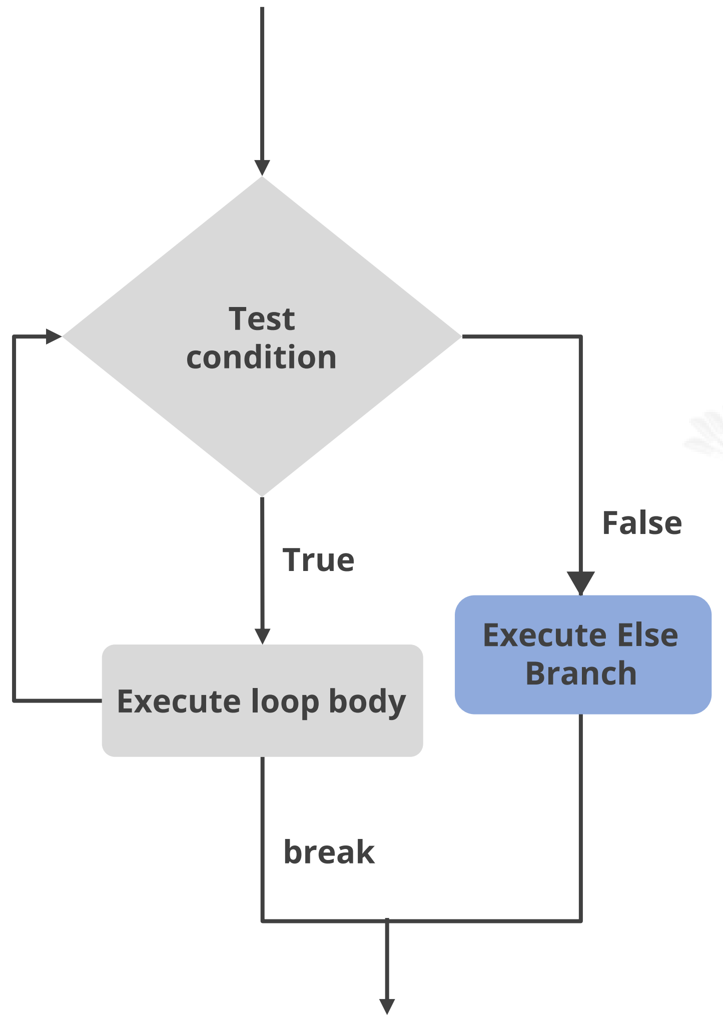
Loop Else Statement

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

Normal Loop Program Flow



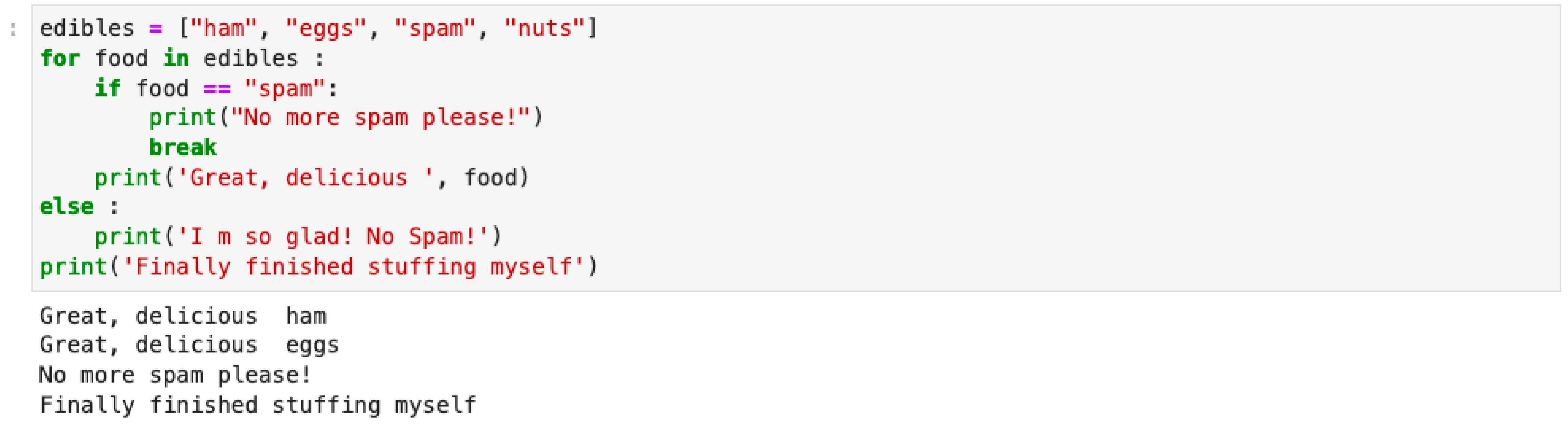
Loop Program Flow with Else



For Else Statement: Example

The following example illustrates the use of for else statement.

Example



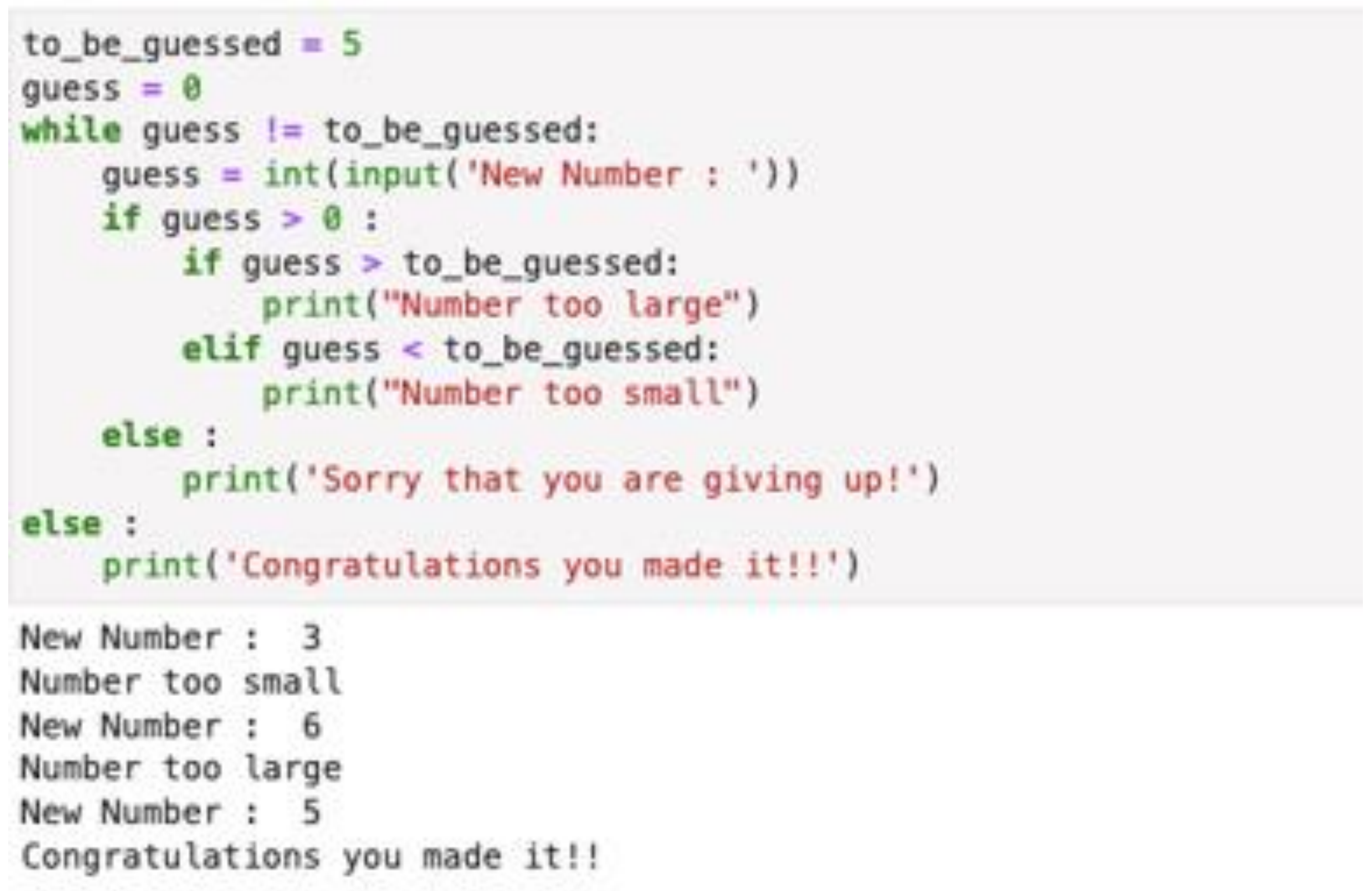
```
edibles = ["ham", "eggs", "spam", "nuts"]
for food in edibles :
    if food == "spam":
        print("No more spam please!")
        break
    print('Great, delicious ', food)
else :
    print('I m so glad! No Spam!')
print('Finally finished stuffing myself')
```

```
Great, delicious  ham
Great, delicious  eggs
No more spam please!
Finally finished stuffing myself
```

While Else Statement: Example

The following example illustrates the use of while else statement.

Example

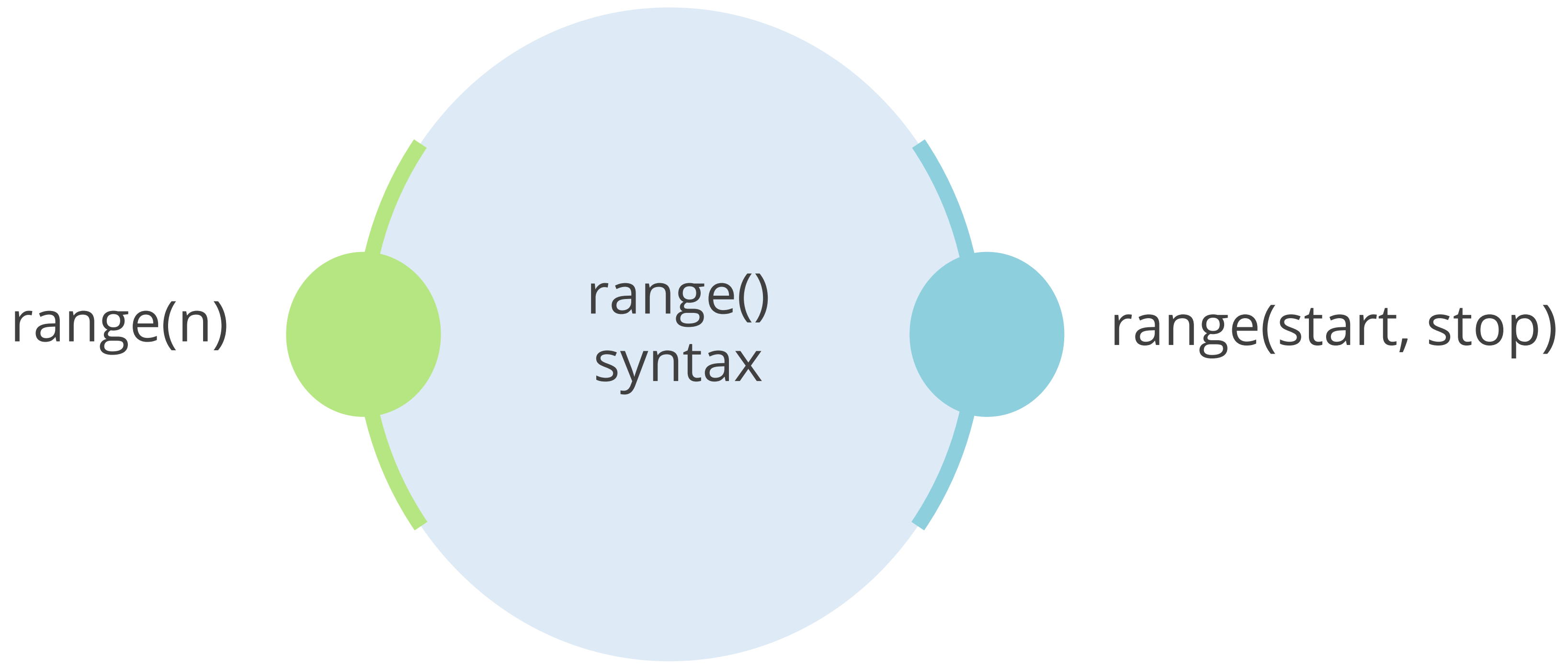


```
to_be_guessed = 5
guess = 0
while guess != to_be_guessed:
    guess = int(input('New Number : '))
    if guess > 0 :
        if guess > to_be_guessed:
            print("Number too large")
        elif guess < to_be_guessed:
            print("Number too small")
    else :
        print('Sorry that you are giving up!')
else :
    print('Congratulations you made it!!')
```

```
New Number :  3
Number too small
New Number :  6
Number too large
New Number :  5
Congratulations you made it!!
```

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.

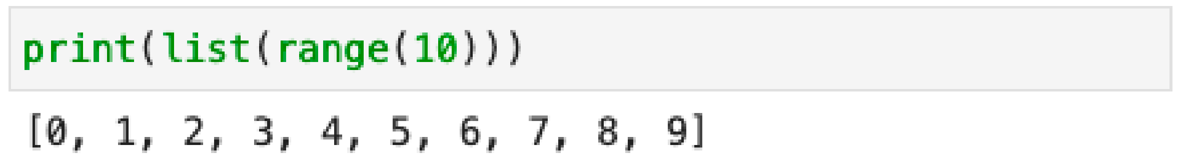


The range object can be converted to a sequence collection using a function, such as list and tuples.

range(n) Function

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

Example



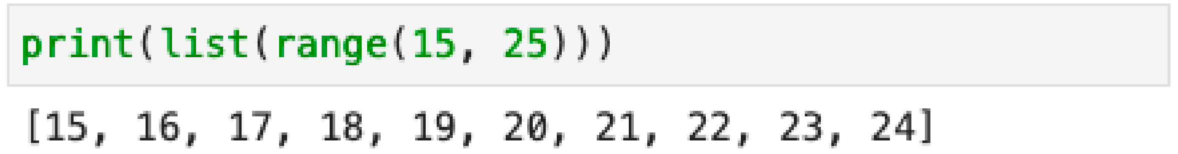
```
print(list(range(10)))
```

```
[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

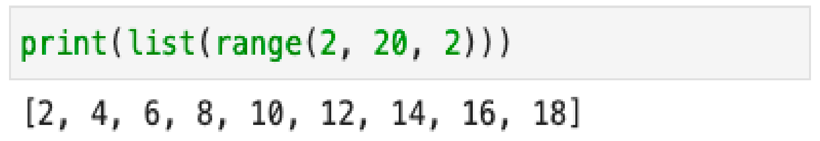
```
range(start, stop, step)
```

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.

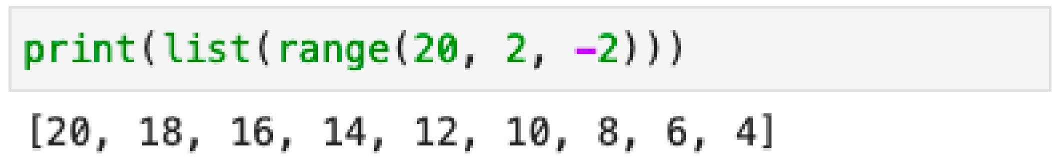
Example



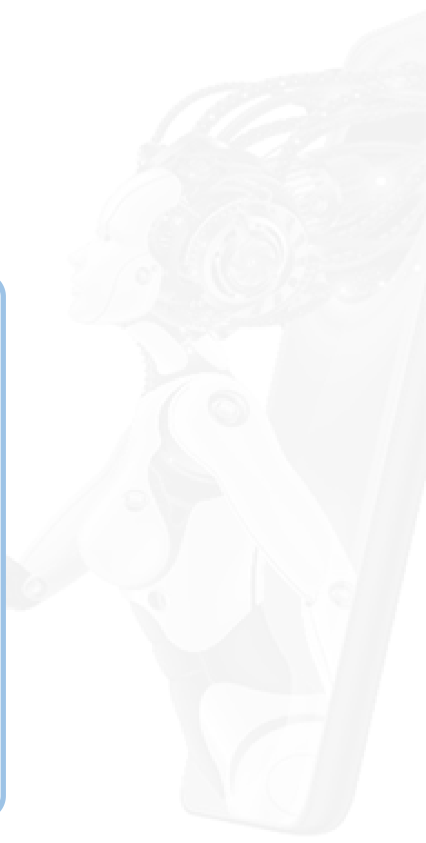
```
print(list(range(2, 20, 2)))
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Negative step value creates a reverse sequence.

Example



```
print(list(range(20, 2, -2)))
[20, 18, 16, 14, 12, 10, 8, 6, 4]
```



simplilearn

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.