Data Science with Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

Loops in Python

Uplatz

Learning outcomes:

Introduction to Loops

Types of Loops

- for loop
- while loop
- nested loop

Loop Control Statements

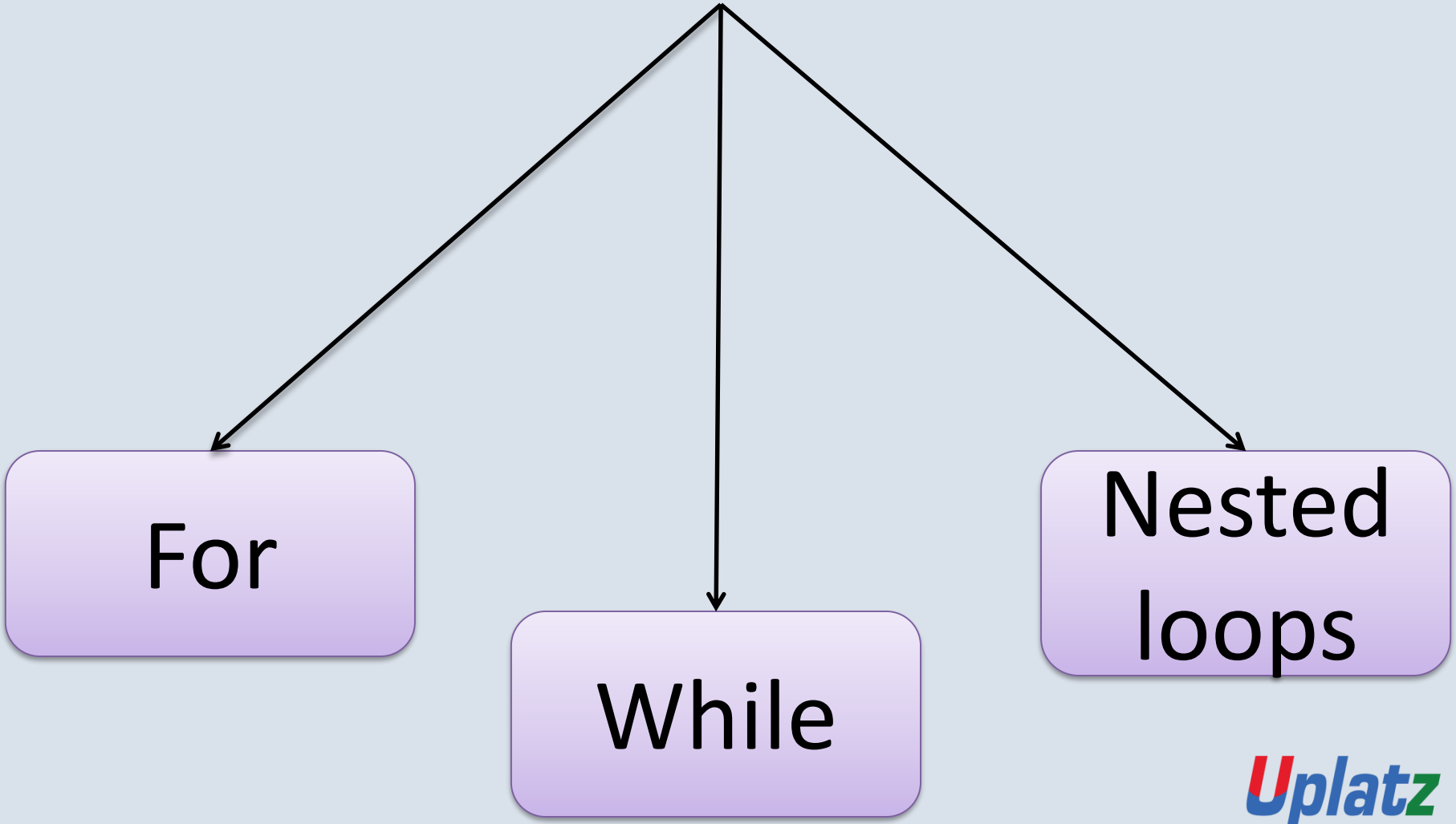
- Break, continue and pass statement

Introduction to Loops

There could be a situation where you got to execute a block of code several numbers of times. Generally, statements are executed sequentially. The primary statement during a function is executed first, followed by the second, and so on.

A loop statement allows us to execute a statement or group of statements multiple times.

Types of Loops



Types of Loops

For loop:

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- A for loop is used for iterating over a sequence.
- It is like an iterator method as found in other object-orientated programming languages.

Types of Loops

For loop:

Syntax:

for var in sequence:

statements(s)

Here **var** is a variable that is used for iterating over a **sequence**. On every iteration it takes the **next value** from sequence until the end of sequence is reached.

sequence : A sequence of values that will be assigned to the target variable **var**. Values are provided using a list or a string or from the built-in function range().

Types of Loops

For loop:

Example:

```
for letter in ‘Expert':
 print ('Current letter is:', letter)
```

range() function:

range() is a built-in function of Python. It is used when a user needs to perform an action for a specific number of times. The range() function is used to generate a sequence of numbers.

Types of Loops

For loop:

Example:

```
for x in range (6) :
    print (x)
```

Example:

```
for y in range (15, 20) :
    print (y)
```

Types of Loops

while loop:

- Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
- While Loop is used to execute number of statements or body till the condition passed in while is true.
- Once the condition is false, the control will come out of the loop.

Types of Loops

while loop:

Syntax:

```
while expression:
  statement(s)
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Types of Loops

while loop:

Example-1 :

```
count = 0
while (count < 6):
    print ('The count is:', count)
    count = count + 1
```

Example-2:

```
table =8
while table < 81 :
    print (table)
    table+=8
```

Types of Loops

nested loop:

The placing of one loop inside the body of another loop is named nesting. Once you “nest” two loops, the outer loop takes control of the amount of complete repetitions of the inner loop. Thus inner loop is executed N- times for each execution of outer loop.

Python programming language allows to use one loop inside another loop .

Types of Loops

nested loop:

Syntax for nested for loop :

```
for iterating_var in sequence:
  for iterating_var in sequence:
        statements(s)
  statements(s)
```

Syntax for nested while loop :

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Types of Loops

nested loop:

Example:

```
for i in range(1,6):
  for j in range (1,i+1):
          print (i, end=" ")
  print ()
```

Types of Loops

Infinite loop:

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Types of Loops

Infinite loop:

Example:

```
cnt = 10
while (cnt > 9):
   print ('The count is:', cnt)
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

break

continue

pass

Loop Control Statements

break statement:

Terminates the loop statement and transfers execution to the statement immediately following the loop.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

Loop Control Statements

break statement:

Example:

```
for i in range(11,21):
    if(i==17):
        break
    print(i)
```

Loop Control Statements

continue statement:

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

It returns the control to the beginning of the while loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

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

Loop Control Statements

continue statement:

Example:

```
for i in range(16,21):
    if(i==18):
        continue
    print(i)
```

Loop Control Statements

Pass statement:

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet.

Loop Control Statements

- Pass is a null statement.

Then what is the difference between **pass** and **comment**???

Interpreter **ignores a comment** entirely

Nothing happens when **pass is executed**

Loop Control Statements

Pass statement:

Example:

```
for i in range(26,41):
    if(i==37):
        pass
    print(i)
```