LEARN

LOOPS IN PYTHON IN 6 STEPS

- DEFINITIONS
- 'WHILE' LOOP
- 'FOR' LOOP
- WORKING WITH THE ITERATORS
- OPERATIONS WITH THE 'FOR' LOOP
- NESTED 'FOR' LOOP

LOOPS

April 3, 2023

Loops

1 Definitions

- When writing programs in Python, in some cases it is not enough to execute our block of code only once. The loops are used to repeat (iterate) the execution of a block of code.
- As one of the most main functions in programming, loops are an important part of almost every programming language. Loops enable programmers to set certain sections of their code to repeat through a number of loops which are referred to as iterations.
- This topic covers using multiple types of loops and applications of loops in Python. You will learn two types of loops which are :
- while Loop,
- for Loop.

2 'while' Loop

- while loops have a boolean logic, similar to if statements. As long as the result of the condition returns True, the code block under while loop runs. When the condition returns to False, the loop execution is terminated and the program control moves further to the next operation

[ ]:
```
while condition:
   body
```

- We will not use this loop as often as the for loop, but it is still worth understanding.

[1]:
```
number = 0

while number < 6:
    print(number)
    number += 1
print('now, number is bigger or equal to 6')
```

```
0
1
2
3
4
```

```
5
now, number is bigger or equal to 6
```

- The variable number acts as a counter in this loop. This variable changes its value after each iteration. When the value of a counter reaches 6, the program control moves to the next operation and prints the text above.
- If we make a logical mistake in the loop variable (since you don't increase your variable, a condition never becomes False and can work forever), we can start an infinite loop! For this reason, we have to specify a condition for the loop to give False to exit the loop.
- We can call a list in while loop.

[2]:
```
my_list = ["a", "b", "c", "d", "e"]
a= 0
while a < len(my_list):
    print(f'square of {a} is : {a**2}')
    a += 1
```

```
square of 0 is : 0
square of 1 is : 1
square of 2 is : 4
square of 3 is : 9
square of 4 is : 16
```

Remember

- variable += number is the same as variable = variable + number.

Always use valid syntax and make comments. In the beginning, it may seem that the while loop is not so easy to apply, but several times later, you'll understand that it's a very useful tool. Lastly, let's play famous 'guessing a number game' using while loop :

[5]:
```
answer = 44

question = ("What number am I thinking of ?")

print("Let's play the guessing game!")

while True:
    guess = int(input(question))

    if guess < answer:
        print('Little higher')
    elif guess > answer:
        print('Little lower')
    else: #guess == answer
        print('Congratulations! You guessed the correct number.')
        break
```

```
Let's play the guessing game!
Little lower
Little lower
Little lower
Little higher
Congratulations! You guessed the correct number.
```

In the example above;

We have written a program that does not exit the while loop until you find t he c orrect number, We used break keyword in order to quit and exit the while loop, When the user knows the answer (44) and enters input, it takes the value of 44 and assigns to variable guess, in the end, else works and breaks the loop.

3 'for' Loop

- You'll learn one of the most used, very simple and useful syntaxes in Python: for loop. When you want to iterate a block of code you will use for loop. To create a for loop, you need a variable and an iterable object.

[ ]:
```
for variable in iterable :
    code block
```

- We need an iterable so we can use a list.

[6]:
```
for i in [1, 2, 3, 4, 5]:
    print(i)
```

```
1
2
3
4
5
```

- In the structure of the for loop, you can use also an iterable variable of course

[9]:
```
seasons = ['spring', 'summer', 'autumn', 'winter']
for season in seasons:
    print(season)
```

```
spring
summer
autumn
winter
```

4 Working with the Iterators

- Let us explain the term iteration a little more.

- Iterable object can be anything for which items are received one by one, forward only. In Python, the process of recurrent execution of a block of code is called an iteration.

We can basically classify iterations as two headings :

If the number of repetitions is predetermined, it is called definite iteration.

The repetition structure that makes the code block run as long as the predetermined condition generates True is called indefinite iteration.

For example; string, list, tuple, dictionary or set are the iterable types of data.

[1]:
```
course = 'github'

for i in course :
    print(i)
```

```
g
i
t
h
u
b
```

5 Operations with the 'for' Loop

- In this topic, you will learn about how we use the for loop using several functions and methods.
- In the example below, you'll get a number from the user and print a sentence the number of times we receive from the user.

[3]:
```
times = int(input("How many times should I say 'I love you'"))

for i in range(times):
    print('I love you')

# As we stated before, input() function can get the value of different data␣
 ↪types and assign a variable you chose. In the example above, it gets a␣
 ↪number and assigns it to times. If the user enters 3 then the output will be␣
 ↪:
```

```
I love you
I love you
I love you
```

- If you want the user to input numbers, use the input() function together with the int() function. Otherwise, the value entered by the user will be in the string data type.

Let's get to know the features of range() function in details:

The range() function creates an iterable sequence of numbers. And it can be simply converted into an iterable object: list, set, and tuple.

[4]:
```
b = list(range(11))
print(b)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[5]:
```
a = set(range(0,10))
print(a)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

[6]:
```
c = tuple(range(11))

print(c)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

- You can keep in mind the formula syntax below for range() function:

The formula syntax is : range(start, stop, step)

Besides, you can use starred expression * before range() function to separate its elements. See these examples :

[7]:
```
print(range(5)) # it will not print  the numbers in sequence
print(*range(5)) # '*' seperates its elements
```

```
range(0, 5)
0 1 2 3 4
```

[8]:
```
print(*range(5,25,2))
```

```
5 7 9 11 13 15 17 19 21 23
```

- Starred expression * can separate other iterable objects. For example, you can separate a string:

[10]:
```
print(*('separate'))
```

```
s e p a r a t e
```

- You can create reverse sequence numbers using a negative step

[13]:
```
print(*range(10,0,-2))
```

```
10 8 6 4 2
```

In some cases, you will need to set up the for loop with multiple variables and the iterables

[14]:
```
text = ['one','two','three','four','five']
numbers = [1, 2, 3, 4, 5]
for x, y in zip(text, numbers):
    print(x, ':', y)
```

```
one : 1
two : 2
three : 3
four : 4
five : 5
```

- zip() function make an iterator that aggregates elements from each of the iterables.

6 Nested 'for' Loop

- As a programmer, you may sometimes need to interact with a single element of an iterable data and all other elements simultaneously, that is, your code block in a loop can also contain a loop. Yes, we're talking about nested loops.
- In Python, you can easily place one loop inside another one. First outer loop then inner one runs.

[23]:
```
who =['I am', 'You are']
mood = ['happy', 'confident']
for i in who:
    for ii in mood:
        print(i + ii)
```

```
I amhappy
I amconfident
You arehappy
You areconfident
```

If you want to get deeper into it, you can find the details about the loops https://docs.python.org/3.8/tutorial/controlflow.html?highlight=else#b and-continue-statements-and-else-clauses-on-loops