Python Programming Language

Prepared by: Mohamed Ayman

Machine Learning Engineer

spring 2018


facebook.com/sw.eng.MohamedAyman


sw.eng.MohamedAyman@gmail.com

wuzzuf.net/me/engMohamedAyman


codeforces.com/profile/Mohamed_Ayman

Loops

Outline

1- Loop Definition

2- Types of Loops

3- while Loop Statements

4- else with while Loop

5- Single statement suites at while Loop

6- for Loop Statements

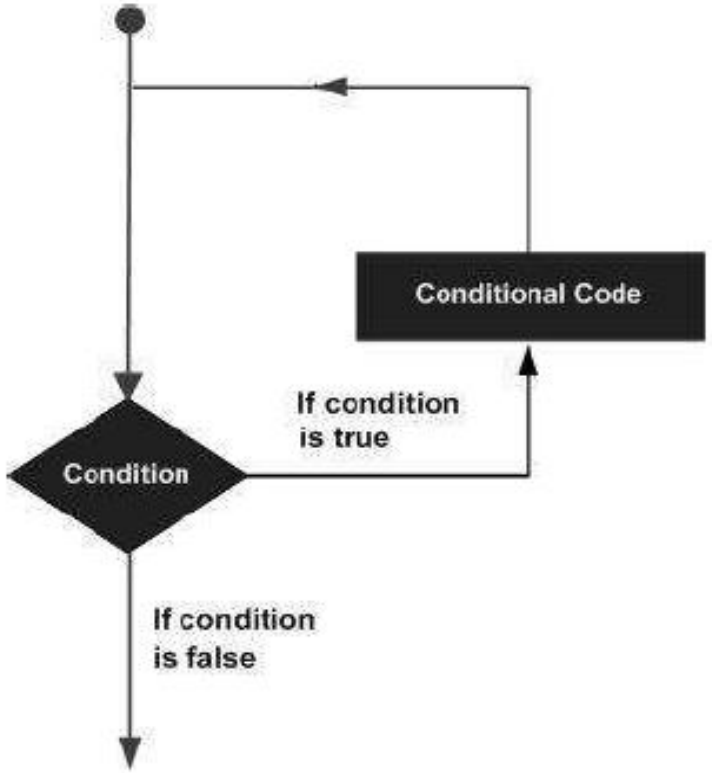
7- else with for Loop

8- Nested loops

9- Loop Control Statements

Loop Definition

- In general, statement are executed sequentially - The first statement in a function is executed first, followed by the second. and so on.
- There may be a situation when you need to execute a block of code several number of times.
- Programming language provide various control structure that allow more complicated execution paths.
- A loop statement allows us to execute a statement of group of statement multiple times.



Types of Loops

- Python programming language provides the following types of loops to handle looping requirement.

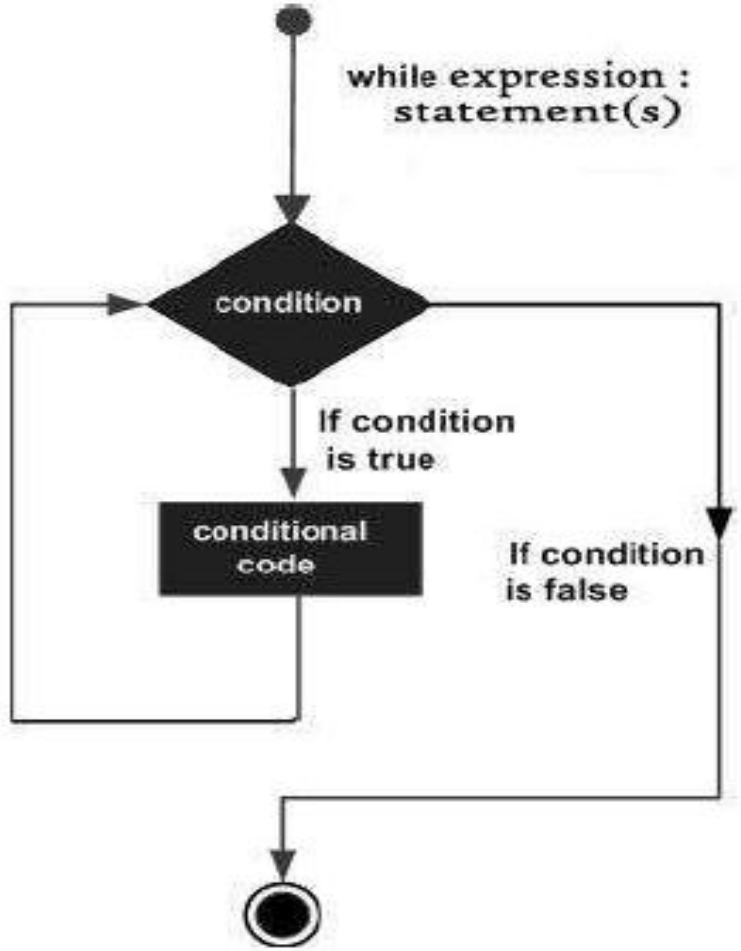
Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, or for loop.

while Loop Statements

- A while loop statement in Python programming language repeatedly executes a target statement as long as given condition is true.

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

- Expression can be TRUE by 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.



while Loop Statements


```
i = 0
while (i < 5):
    print(i)
    i+=1
print('finish')
```

```
0
1
2
3
4
finish
```

else with while Loop

- Python support having an else statement associated with a loop statement.
- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.
- The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement.
- In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

else with while Loop


```
i = 0
while (i < 5):
    print(i,'is less than 5')
    i+=1
else:
    print(i,'is not less than 5')
```

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

Single statement suites at while Loop

- Similar to the if statement syntax, if your while clause consists only of a single statement, it may be placed on the same line as the while header.

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

Problem 1
Power two

- Take as an input n and print 2^0, 2^1, 2^2 and so on ... 2^n using while loop such that n >= 0 and integer

- Test Cases:

```
 5
1 2 4 8 16 32
```

```
 10
1 2 4 8 16 32 64 128 256 512 1024
```

```
 3
1 2 4 8
```

```
 1
1 2
```

```
 12
1 2 4 8 16 32 64 128 256 512 1024 2048 4096
```

Problem 1 Solution
Power two


```
n = int(input())

i=0
while(i <= n) :
    print(2**i, end=' ')
    i+=1
```

Problem 2
Factorization

- Take as an input n and print it's factors using while loop such that n >= 1 and integer
- Test Cases:

```
 10
1 2 5 10
```

```
 12
1 2 3 4 6 12
```

```
 20
1 2 4 5 10 20
```

```
 25
1 5 25
```

```
 120
1 2 3 4 5 6 8 10 12 15 20 24 30 40 60 120
```

```
 13
1 13
```

Problem 2 Solution
Factorization


```
n = int(input())
i=1
while(i<=n):
    if(n%i==0):
        print(i,end=' ')
    i+=1
```

for Loop Statements

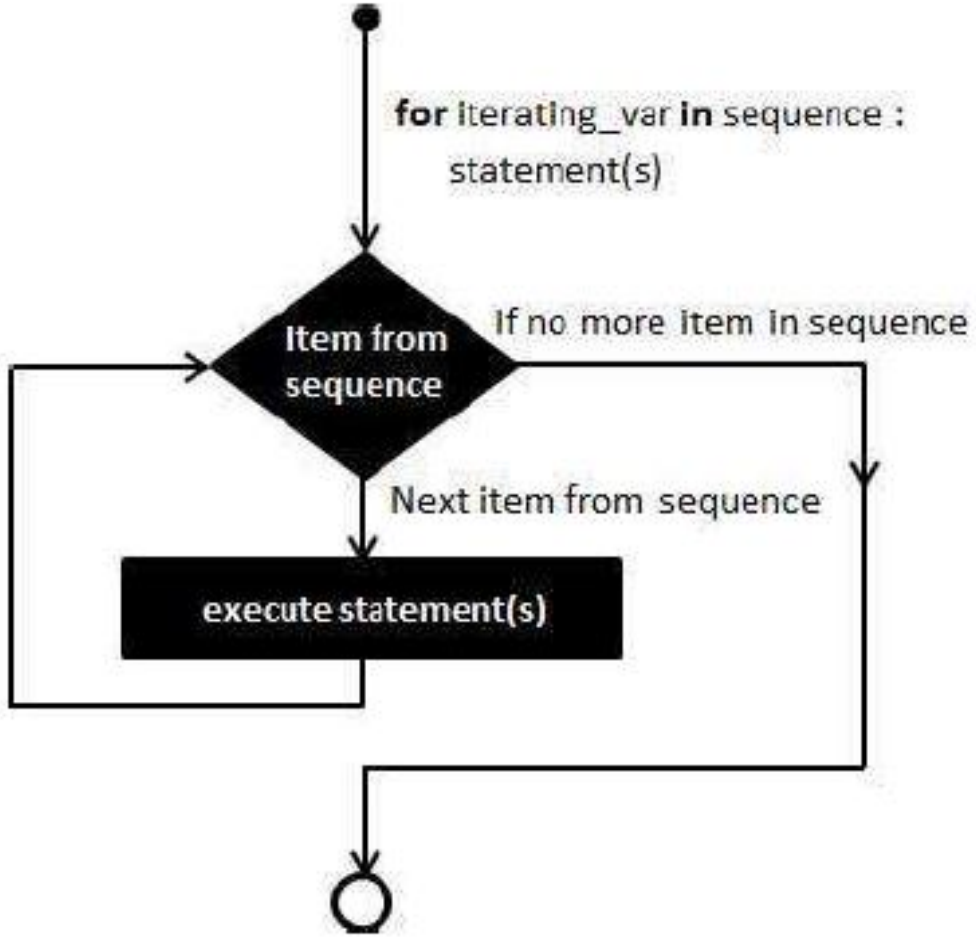
- The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.

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

- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statement block is executed. Each item in the list is assigned to iterating_var, and the statement(s) blocks is executed until the entire sequence is exhausted.

for Loop Statements

- Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.
- We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).



for Loop Statements


```python
for i in 'python' :
    print('current letter is : ',i)

print()

fruits = ['banana', 'apple', 'mango']
for i in fruits:
    print('current fruit is : ',i)
```

```
current letter is :  p
current letter is :  y
current letter is :  t
current letter is :  h
current letter is :  o
current letter is :  n

current fruit is :  banana
current fruit is :  apple
current fruit is :  mango
```

The range() function

- The built-in function range() is the right function to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.
- range() generates an iterator to progress integers starting with 0 up to n-1. To obtain a list object of the sequence, it is type casted to list(). Now this list can be iterated using the for statement.
- We can also define the start, stop and step size as range(start, stop, step). step size defaults to 1 if not provided.
- This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.
- To force this function to output all the items, we can use the function list().

The range() function


```
print( list( range(5) ) )

for i in range(5):
    print(i)
```

```
[0, 1, 2, 3, 4]
0
1
2
3
4
```

```
print( list( range(2,7) ) )

for i in range(2,7):
    print(i)
```

```
[2, 3, 4, 5, 6]
2
3
4
5
6
```

The range() function


```
print( list( range(3,11,2) ) )

for i in range(3,11,2):
    print(i)
```

```
[3, 5, 7, 9]
3
5
7
9
```

```
print( list( range(8,3,-1) ) )

for i in range(8,3,-1):
    print(i)
```

```
[8, 7, 6, 5, 4]
8
7
6
5
4
```

The range() function


```
fruits = ['banana', 'apple', 'mango']

for i in range( len(fruits) ):
    print('current fruit is : ',fruits[i])
```

```
current fruit is :  banana
current fruit is :  apple
current fruit is :  mango
```

else with for Loop

- Python support having an else statement associated with a loop statement.
- If the else statement is used with a for loop, the else blocks is executed only if for loops terminates normally (and not by encountering break statement)
- A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.
- break statement can be used to stop a for loop. In such case, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

else with for Loop


```
numbers = [11, 33, 45, 67, 21, 13]

for i in numbers:
    if i%2 == 0 :
        print('this list contain an even number')
        break
else :
    print('this list does not contain even numbers')
```

```
this list does not contain even numbers
```

Problem 3
Power two

- Take as an input n and print 2^0, 2^1, 2^2 and so on ... 2^n using for loop such that n >= 0 and integer
- Test Cases:

```
 5
1 2 4 8 16 32
```

```
 10
1 2 4 8 16 32 64 128 256 512 1024
```

```
 3
1 2 4 8
```

```
 1
1 2
```

```
 12
1 2 4 8 16 32 64 128 256 512 1024 2048 4096
```

Problem 3 Solution
Power two


```
n = int(input())

for i in range(n+1) :
    print(2**i, end=' ')
```

Problem 4
Factorization

- Take as an input n and print it's factors using for loop such that n >= 1 and integer
- Test Cases:

```
 10
1 2 5 10
```

```
 12
1 2 3 4 6 12
```

```
 20
1 2 4 5 10 20
```

```
 25
1 5 25
```

```
 120
1 2 3 4 5 6 8 10 12 15 20 24 30 40 60 120
```

```
 13
1 13
```

Problem 4 Solution
Factorization


```
n = int(input())
for i in range(1,n+1):
    if(n%i==0):
        print(i,end=' ')
```

Nested loops

- Python programming language allows the use of one loop inside another loop.

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

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

Nested loops

```
for i in range(1,4) :
    for j in range(1,4) :
        print(i*j, end=' ')
    print()
```

```
1 2 3
2 4 6
3 6 9
```

```
i = 1
while i < 4 :
    j = 1
    while j < 4 :
        print(i*j, end=' ')
        j+=1
    print()
    i+=1
```

```
1 2 3
2 4 6
3 6 9
```

Problem 5
Identity Matrix

- Take as an input n and print identity matrix using while loop, such that n >= 1 and integer
- Test Cases:

```
 5
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

```
 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

```
 3
1 0 0
0 1 0
0 0 1
```

```
 2
1 0
0 1
```

```
 1
1
```

Problem 5 Solution
Identity Matrix


```
n = int(input())

i = 0
while (i < n):
    j = 0
    while(j < n):
        if(i == j):
            print(1, end=' ')
        else:
            print(0, end=' ')
        j+=1
    print()
    i+=1
```

Problem 6
Identity Matrix

- Take as an input n and print identity matrix using for loop, such that n >= 1 and integer
- Test Cases:

```
 5
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

```
 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

```
 3
1 0 0
0 1 0
0 0 1
```

```
 2
1 0
0 1
```

```
 1
1
```

Problem 6 Solution
Identity Matrix


```
n = int(input())

for i in range(n):
    for j in range(n):
        if(i == j):
            print(1, end=' ')
        else:
            print(0, end=' ')
    print()
```

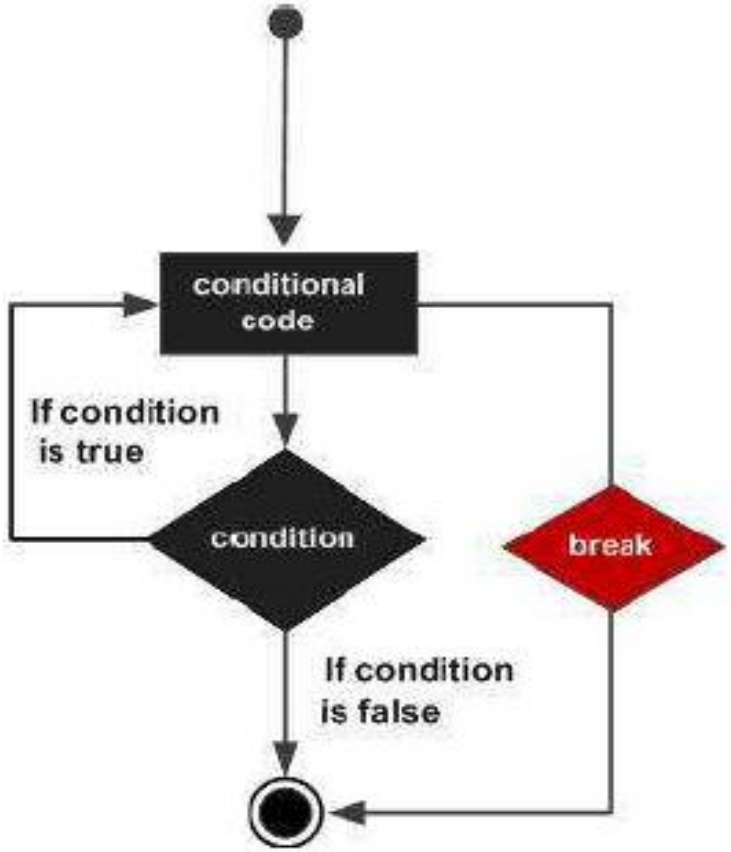
Loop Control Statements

- The Loop control statement change the execution from its normal sequence. When the execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
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.

break statement

- The break statement is used for premature termination of the current loop. After abandoning the loop, execution at the next statement is resumed.
- The most common use of 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.
- If you are using nested loops, the break statement stops execution of the innermost loop and starts executing the next line of the code after the block.



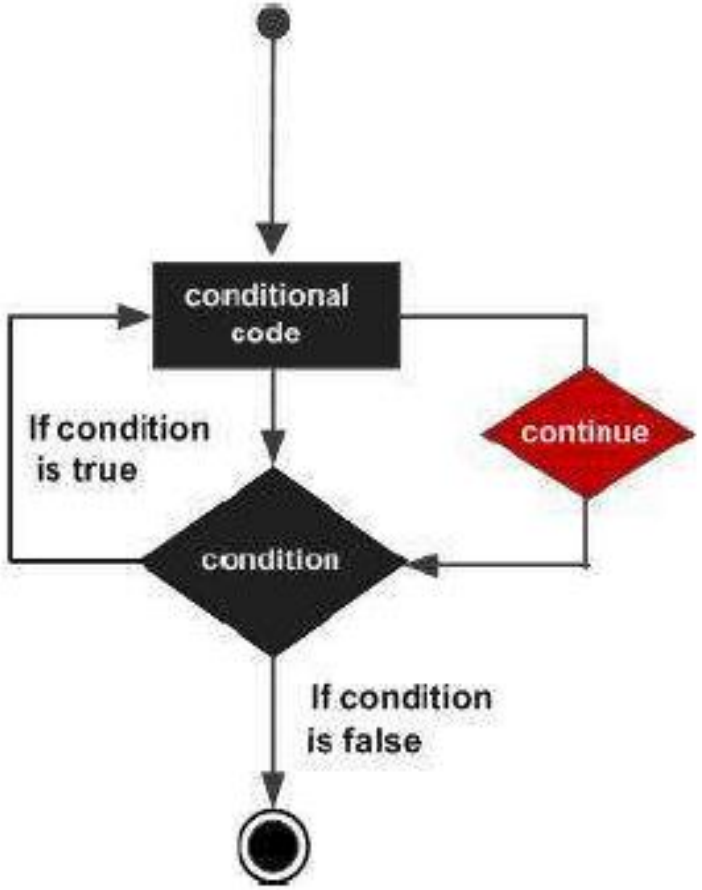
break statement


```
for i in 'python' :
    if(i == 'h') :
        break
    print('current letter is : ',i)

print()

i = 0
while(i < 7) :
    if(i == 4) :
        break
    print('current number is : ',i)
    i+=1
```

```
current letter is :  p
current letter is :  y
current letter is :  t

current number is :  0
current number is :  1
current number is :  2
current number is :  3
```

continue Statement

- The continue statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.
- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



continue Statement


```
for i in 'python' :
    if(i == 'h') :
        continue
    print('current letter is : ',i)

print()

i = 0
while(i < 6) :
    i+=1
    if(i == 4) :
        continue
    print('current number is : ',i)
```

```
current letter is :  p
current letter is :  y
current letter is :  t
current letter is :  o
current letter is :  n

current number is :  1
current number is :  2
current number is :  3
current number is :  5
current number is :  6
```

Problem 7
Check Prime

- Take as an input n and check if n is prime or not prime using for loop such that n >= 2 and integer
- Test Cases:

```
 2
2 is prime
```

```
 3
3 is prime
```

```
 4
4 is not prime
```

```
 5
5 is prime
```

```
 6
6 is not prime
```

```
 7
7 is prime
```

```
 8
8 is not prime
```

```
 9
9 is not prime
```

```
 10
10 is not prime
```

```
 11
11 is prime
```

Problem 7 Solution
Check Prime


```
n = int(input())

for i in range(2,n):
    if(n%i==0):
        print(n,"is not prime")
        break
else:
    print(n,"is prime")
```

Problem 8
Check Prime

- Take as an input n and check if n is prime or not prime using while loop such that n >= 2 and integer
- Test Cases:

```
 2
2 is prime
```

```
 3
3 is prime
```

```
 4
4 is not prime
```

```
 5
5 is prime
```

```
 6
6 is not prime
```

```
 7
7 is prime
```

```
 8
8 is not prime
```

```
 9
9 is not prime
```

```
 10
10 is not prime
```

```
 11
11 is prime
```

Problem 8 Solution
Check Prime


```
n = int(input())
i = 2
while (i<n) :
    if(n%i==0):
        print(n,"is not prime")
        break
    i+=1
else:
    print(n,"is prime")
```

pass Statement

- In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored, however, nothing happens when pass is executed. It results into no operation.

```
n = int(input())
if (n < 0):
  pass
```

- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So, we use the pass statement to construct a body that does nothing.

- It 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 statement is also useful in places where your code will eventually go, but has not been written yet i.e. in stubs).

```
n = int(input())
for i in range(n):
  pass
```

Outline

- 1- Loop Definition
- 2- Types of Loops
- 3- while Loop Statements
- 4- else with while Loop
- 5- Single statement suites at while Loop
- 6- for Loop Statements
- 7- else with for Loop
- 8- Nested loops
- 9- Loop Control Statements

Practice

Questions ?

References

[01]	Online Courses YouTube playlists	http://bit.ly/2EcbLvZ
[02]	Programiz Tutorial	https://bit.ly/2paX2xA
[03]	Python 3 Tutorial Point PDF	http://bit.ly/2xcqp66
[04]	Python Guru Tutorial	https://bit.ly/28TtLRr
[05]	Python for Everybody PDF	http://bit.ly/2gSWkSS
[06]	A Byte of Python PDF	http://bit.ly/1I9Yqp9
[07]	Python Course Tutorial	https://bit.ly/2I076hU
[08]	Python Basics Tutorial PDF	http://bit.ly/1jMlluB
[09]	Learning to Program Using Python	http://bit.ly/2zO4f91
[10]	Think Python PDF	http://bit.ly/2cTgLIk
[11]	Python 3 Patterns, Recipes and Idioms	http://bit.ly/2by2bsN