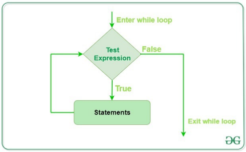
Python While Loops

- In Python, While Loops is used to execute a block of statements repeatedly until a given condition is satisfied.
- And when the condition becomes false, the line immediately after the loop in the program is executed.
- While loop falls under the category of indefinite iteration.
- Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance.



In [1]:

```python
# While Loops

temp = 0
while temp!=-1000:
    temp = eval(input('Enter a temperature (-1000 to quit): '))
    print('In Fahrenheit that is', 9/5*temp+32)
print('The loop is over now')
```

```
Enter a temperature (-1000 to quit): 95
In Fahrenheit that is 203.0
Enter a temperature (-1000 to quit): 96
In Fahrenheit that is 204.8
Enter a temperature (-1000 to quit): 100
In Fahrenheit that is 212.0
Enter a temperature (-1000 to quit): -1000
In Fahrenheit that is -1768.0
The loop is over now
```

In [2]:

```python
i = 0
while i !=10:
    i+= 1
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [3]:

```python
i = 0
while i!=10:
    print(i)
    i+= 1
```

```
0
1
2
3
4
5
6
7
8
9
```

In [4]:

```python
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [5]:

```python
temp = 0
while temp!=-1000:
    temp = eval(input('Enter a temperature (-1000 to quit): '))
    if temp!=-1000:
        print('In Fahrenheit that is', 9/5*temp+32)
    else:
        print('Bye!')
```

```
Enter a temperature (-1000 to quit): 400
In Fahrenheit that is 752.0
Enter a temperature (-1000 to quit): 96
In Fahrenheit that is 204.8
Enter a temperature (-1000 to quit): -1000
Bye!
```

In [6]:

```python
from random import randint
secret_num = randint(1,5)
guess = 0
while guess != secret_num:
    guess = eval(input('Guess the secret number: '))
print('You finally got it!')
```

```
Guess the secret number: 4
Guess the secret number: 3
Guess the secret number: 1
Guess the secret number: 2
You finally got it!
```

In [7]:

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [8]:

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

```
0
1
2
3
4
5
6
7
8
9
```

In [9]:

```
temp = eval(input('Enter a temperature in Celsius: '))
if temp<-273.15:
    print('That temperature is not possible.')
else:
    print('In Fahrenheit, that is', 9/5*temp+32)
```

```
Enter a temperature in Celsius: 45
In Fahrenheit, that is 113.0
```

In [10]:

```
temp = eval(input('Enter a temperature in Celsius: '))
while temp<-273.15:
    temp = eval(input('Impossible. Enter a valid temperature: '))
print('In Fahrenheit, that is', 9/5*temp+32)
```

```
Enter a temperature in Celsius: 58
In Fahrenheit, that is 136.4
```

In [11]:

```
i = 0
while i<20:
    print(i, end = ' ')
    i=i+2
print('Bye!')
```

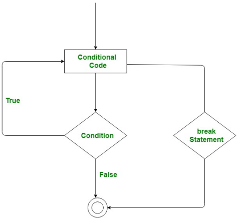
```
0 2 4 6 8 10 12 14 16 18 Bye!
```

In [ ]:

```
# Infinte Loops
i=0
while i<10:
    print(i)
```

Break statement

- The break statement is used to terminate the loop or statement in which it is present.
- After that, the control will pass to the statements that are present after the break statement, if available.
- If the break statement is present in the nested loop, then it terminates only those loops which contains break statement.



In [13]:

```
# The break statement

for i in range(10):
    num = eval(input('Enter number: '))
    if num<0:
        break
```

```
Enter number: 45
Enter number: 2
Enter number: -1
```

In [14]:

```
# Same in While Loop

i=0
num=1
while i<10 and num>0:
    num = eval(input('Enter a number: '))
```

```
Enter a number: 45
Enter a number: 5
Enter a number: -1
```

In [15]:

```python
temp = 0
while temp!=-1000:
    temp = eval(input('Please give a temp: '))
    if temp!=-1000:
        print(9/5*temp+32)
    else:
        print('Bye!')
```

```
Please give a temp: 96
204.8
Please give a temp: -1000
Bye!
```

In [16]:

```python
while True:
    temp = eval(input('Please give a temp: '))
    if temp==-1000:
        print('Bye')
        break
    print(9/5*temp+32)
```

```
Please give a temp: 45
113.0
Please give a temp: -1000
Bye
```

In [17]:

```python
# Using Else with For Loop

for i in range(10):
    num = eval(input('Enter number: '))
    if num<0:
        print('Stopped early')
        break
    else:
        print(num*num)
else:
    print('User entered all ten values')
```

```
Enter number: 96
9216
Enter number: -1
Stopped early
```

In [18]:

```python
# to check if an integer num is prime

num = eval(input('Please enter a number to see if the given input is Prime or not: '))
for i in range(2, num):
    if num%i==0:
        print('Not prime')
        break
else:
    print('Prime')
```

```
Please enter a number to see if the given input is Prime or not: 5
Prime
```

In [19]:

```python
num = eval(input('Please enter a number to see if the given input is Prime or not: '))
i = 2
while i<num and num%i!=0:
    i=i+1
if i==num:
    print('Prime')
else:
    print('Not prime')
```

```
Please enter a number to see if the given input is Prime or not: 4
Not prime
```

The guessing game - Using Both While and For Loop

- The player only gets five turns.
- The program tells the player after each guess if the number is higher or lower.
- The program prints appropriate messages for when the player wins and loses.

In [20]:

```python
secret_num = randint(1,100)
```

In [21]:

```python
from random import randint
num_guesses = 0
guess = 0
while guess != secret_num and num_guesses <= 4:
    guess = eval(input('Enter your guess (1-100): '))
    num_guesses = num_guesses + 1
    if guess < secret_num:
        print('HIGHER.', 5-num_guesses, 'guesses left.\n')
    elif guess > secret_num:
        print('LOWER.', 5-num_guesses, 'guesses left.\n')
    else:
        print('You got it!')

if num_guesses==5 and guess != secret_num:
    print('You lose. The correct number is', secret_num)
```

```
Enter your guess (1-100): 12
LOWER. 4 guesses left.

Enter your guess (1-100): 4
HIGHER. 3 guesses left.

Enter your guess (1-100): 15
LOWER. 2 guesses left.

Enter your guess (1-100): 3
HIGHER. 1 guesses left.

Enter your guess (1-100): 8
LOWER. 0 guesses left.

You lose. The correct number is 6
```

In [22]:

```
for num_guesses in range(5):
    guess = eval(input('Enter your guess (1-100): '))
    if guess < secret_num:
        print('HIGHER.', 5-num_guesses, 'guesses left.\n')
    elif guess > secret_num:
        print('LOWER.', 5-num_guesses, 'guesses left.\n')
    else:
        print('You got it!')
        break
else:
    print('You lose. The correct number is', secret_num)
```

```
Enter your guess (1-100): 3
HIGHER. 5 guesses left.

Enter your guess (1-100): 4
HIGHER. 4 guesses left.

Enter your guess (1-100): 8
LOWER. 3 guesses left.

Enter your guess (1-100): 15
LOWER. 2 guesses left.

Enter your guess (1-100): 12
LOWER. 1 guesses left.

You lose. The correct number is 6
```

In [23]:

```
# Exercises

#1 Print 2, 5, 8, 11, 14, 17, 20 with a while loop.

i = 2
while i < 21:
    print(i, end =' ')
    i = i + 3
```

```
2 5 8 11 14 17 20
```

In [24]:

```
#2 Print 100, 99, 98, ... 1 with a while loop.

i = 100
while i != 0:
    print(i, end =' ')
    i= i - 1
```

```
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 4
8 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

In [25]:

```
#3 User enters numbers until a negative.  Then sum is printed, excluding negative.

num = 0
total = 0
while num >= 0:
    num = eval(input('Please enter a Positive number, (enter a negative number to break): '))
    if num >=0:
        total = num + total
print(total)
```

```
Please enter a Positive number, (enter a negative number to break): 7
Please enter a Positive number, (enter a negative number to break): 7
Please enter a Positive number, (enter a negative number to break): -1
14
```

In [29]:

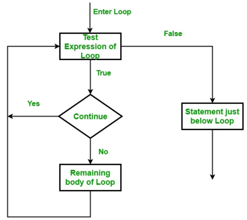
```
#4 User enters numbers from 1 to 10, stopping with a 5.  Print out how many numbers and whether a 3 was entered.
num = 0
count = 0
flag = False
while num != 5:
    num = eval(input('Enter a number (5 to stop): '))
    count += 1
    if num == 3:
        flag = True
print('The number of numbers given is:', count)

if flag:
    print('Yes, A 3 has been entered')
else:
    print('No, A 3 has not been entered')
```

```
Enter a number (5 to stop): 3
Enter a number (5 to stop): 4
Enter a number (5 to stop): 5
The number of numbers given is: 3
Yes, A 3 has been entered
```

Continue statement

- Continue is also a loop control statement just like the break statement.
- continue statement is opposite to that of break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.
- As the name suggests the continue statement forces the loop to continue or execute the next iteration.
- When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.



In [30]:

```
# loop from 1 to 10 - Continue Statement
for i in range(1, 11):
    if i == 6:
        continue
    else:
        print(i, end = " ")
```

```
1 2 3 4 5 7 8 9 10
```

In [31]:

```
for letter in 'Python':
    if letter == 'h':
        continue
    print ('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

In [32]:

```
var = 10
while var > 0:
    var = var -1
    if var == 5:
        continue
    print ('Current variable value :', var)
print("Good bye!")
```

```
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

In [33]:

```
# program to display only odd numbers
for num in [20, 11, 9, 66, 4, 89, 44]:
    if num%2 == 0:
        continue
    print(num, end =' ')
```

```
11 9 89
```

Pass Statements

- As the name suggests pass statement simply does nothing.
- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is like null operation, as nothing will happen is it is executed.
- Pass statement can also be used for writing empty loops.
- Pass is also used for empty control statement, function and classes.

In [34]:

```
# Pass Statements

for letter in 'Python':
    if letter == 'h':
        pass
        print('This is pass block')
    print('Current Letter :', letter)

print("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

Python | assert keyword

- Assertions in any programming language are the debugging tools which help in smooth flow of code.
- Assertions are mainly assumptions that a programmer knows always wants to be true and hence puts them in code so that failure of them doesn't allow the code to execute further.
- In python assert keyword helps in achieving this task.
- This statement simply takes input a boolean condition, which when returns true doesn't return anything, but if it is computed to be false, then it raises an AssertionError along with the optional message provided.

In [37]:

```
# Assert

x = "hello"
```

```
#if condition returns True, then nothing happens:
assert x == "hello"

#if condition returns False, AssertionError is raised:
#assert x == "goodbye"
```

In [39]:

```
# x = "hello"

# #if condition returns False, AssertionError is raised:
# assert x == "goodbye", "X should be 'hello'"
```

In [42]:

```
x = input('Please enter the right string: ')
guess = input('Please guess a right string: ')

#if condition returns False, AssertionError is raised:
assert x == "goodbye", "X should be {}".format(x)
```

```
Please enter the right string: gsaidheeraj
Please guess a right string: gsaidheeraj
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-42-6e63fdda1b7d> in <module>
      3 
      4 #if condition returns False, AssertionError is raised:
----> 5 assert x == "goodbye", "X should be {}".format(x)

AssertionError: X should be gsaidheeraj
```

Lamda Function

- As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions. It has the following syntax:

lambda arguments:

- This function can have any number of arguments but only one expression, which is evaluated and returned.
  - One is free to use lambda functions wherever function objects are required.
  - You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
  - It has various uses in particular fields of programming besides other types of expressions in functions.

Use of lambda() with filter()

- The filter() function in Python takes in a function and a list as arguments.
- This offers an elegant way to filter out all the elements of a sequence “sequence”, for which the function returns True.

Use of lambda() with map()

- The map() function in Python takes in a function and a list as argument.
- The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item.

Use of lambda() with reduce()

- The reduce() function in Python takes in a function and a list as argument.
- The function is called with a lambda function and a list and a new reduced result is returned.
- This performs a repetitive operation over the pairs of the list.

In [43]:

```
# Lambda Function

adder = lambda x, y: x + y
print (adder (1, 2))
```

```
3
```

In [44]:

```
x = lambda x,y : x + y
print(x(5,10))
```

```
15
```

In [45]:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

In [46]:

```
(lambda x: x + 1)(2)
```

Out[46]:

```
3
```

In [47]:

```
def myfunc(n):
    return lambda a : a * n
```

In [48]:

```
def myfunc(n):
      return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(22))
```

```
44
```

In [49]:

```
def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

In [50]:

```
# Lambda with a Filter

sequences = [10,2,8,7,5,4,3,11,0, 1]
filtered_result = filter (lambda x: x > 4, sequences)
print(list(filtered_result))
```

```
[10, 8, 7, 5, 11]
```

In [51]:

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0) , my_list))

print(new_list)
```

```
[4, 6, 8, 12]
```

In [56]:

```
sequences = 10,15
filtered_result = (lambda x: x*x) #(sequences)
print(tuple(filtered_result))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-56-4598b7cf4b5a> in <module>
      1 sequences = 10,15
      2 filtered_result = (lambda x: x*x) #(sequences)
----> 3 print(tuple(filtered_result))

TypeError: 'function' object is not iterable
```

In [57]:

```
# lambdas in map()

sequences = [10,2,8,7,5,4,3,11,0, 1]
filtered_result = map (lambda x: x*x, sequences)
print(list(filtered_result))
```

```
[100, 4, 64, 49, 25, 16, 9, 121, 0, 1]
```

In [58]:

```
# Python code to illustrate
# reduce() with lambda()
# to get sum of a list
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y: x + y), li)
print (sum)
```

```
193
```

Python Modules

- A module is a file containing Python definitions and statements.
- A module can define functions, classes and variables. A module can also include runnable code.
- Grouping related code into a module makes the code easier to understand and use.

In [63]:

```
import sys
sys.version
```

Out[63]:

```
'3.6.12 |Anaconda, Inc.| (default, Sep  9 2020, 00:29:25) [MSC v.1916 64 bit (AMD64)]'
```

In [64]:

```
sys.path
```

Out[64]:

```
['C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\python36.zip',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\DLLs',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\lib',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow',
 '',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\lib\\site-packages',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\lib\\site-packages\\win32',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\lib\\site-packages\\win32\\lib',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\lib\\site-packages\\Pythonwin',
 'C:\\Users\\saidh\\miniconda3\\envs\\tensorflow\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\saidh\\.ipython']
```

In [65]:

```
import math
math.pi
```

Out[65]:

```
3.141592653589793
```

In [66]:

```
math.e
```

Out[66]:

```
2.718281828459045
```

In [67]:

```
math.radians(30)
```

Out[67]:

```
0.5235987755982988
```

In [68]:

```
math.degrees(math.pi/6)
```

Out[68]:

```
29.999999999999996
```

In [69]:

```
math.sin(0.5235987755982988)
```

Out[69]:

```
0.49999999999999994
```

In [70]:

```
math.cos(0.5235987755982988)
```

Out[70]:

```
0.8660254037844387
```

In [71]:

```
math.log(10)
```

Out[71]:

```
2.302585092994046
```

In [72]:
```
math.log10(10)
```
Out[72]:
```
1.0
```
In [73]:
```
math.exp(10)
```
Out[73]:
```
22026.465794806718
```
In [74]:
```
math.e**10
```
Out[74]:
```
22026.465794806703
```
In [75]:
```
math.pow(2,4)
```
Out[75]:
```
16.0
```
In [76]:
```
math.sqrt(100)
```
Out[76]:
```
10.0
```
In [77]:
```
math.ceil(4.5867)
```
Out[77]:
```
5
```
In [78]:
```
math.floor(4.5687)
```
Out[78]:
```
4
```
In [79]:
```
import statistics
statistics.mean([2,5,6,9])
```
Out[79]:
```
5.5
```
In [80]:
```
statistics.median([1,2,3,8,9])
```
Out[80]:
```
3
```
In [81]:
```
statistics.mode([2,5,3,2,8,3,9,4,2,5,6])
```
Out[81]:
```
2
```
In [82]:
```
statistics.stdev([1,1.5,2,2.5,3,3.5,4,4.5,5])
```
Out[82]:
```
1.3693063937629153
```
In [ ]:
```

```