Control Statements in Python

What we will learn?

Decision making

> if statement
>
> if..else statement
>
> if..elif..else statement

Loops

> while loop
>
> for loop
>
> Nested loops

Else suite

> for..else
>
> while..else

Loop control statements

> break
>
> continue
>
> pass
>
> assert
>
> return

Control statements are statements which control or change the flow of execution.

Decision making statements in python

Python programming language provides following types of decision making statements to handle conditional requirements.

if statement

It executes one or more statements depending on the condition whether it is true or not.

Syntax:

```
if condition:
    statements
```

```
var=100
if var<=100:
  print("Var=",var)
  print("Value of variable var is printed")
```

```
Var= 100
Value of variable var is printed
```

if..else statement

It executes a group of statements when a condition is true; otherwise it will execute anpther group of statements.

Syntax:

```
if condition:
    statements1
else:
    statements2
```

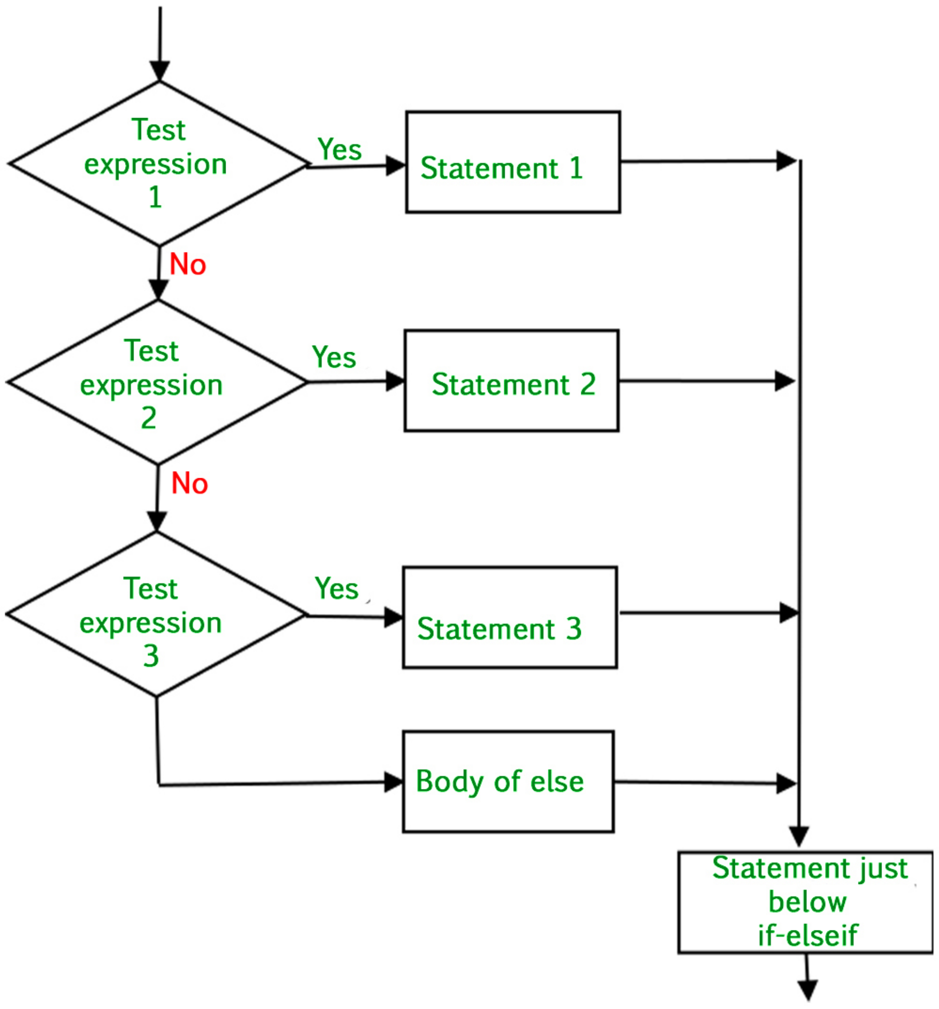
Program to check the number is even or odd.

```
x=10
if x%2==0:
  print("Even number")
else:
  print("Odd number")
```

```
Even number
```

if..elif..else statement

Here, if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



Syntax:

```
if condition1:
    statements1
elif condition2:
    statements2
    .
    .
    .
else:
    statements3
```

Program to check entered number is positive, negative or zero

```
x=int(input("Enter integer "))
if x>0:
  print(x," is positive")
elif x<0:
  print(x," is negative")
```

```
else:
  print(x," is zero")
```

```
    Enter integer -4
    -4  is negative
```

Loops in python

Python programming language provides following types of loops to handle looping requirements.

while loop

Syntax:

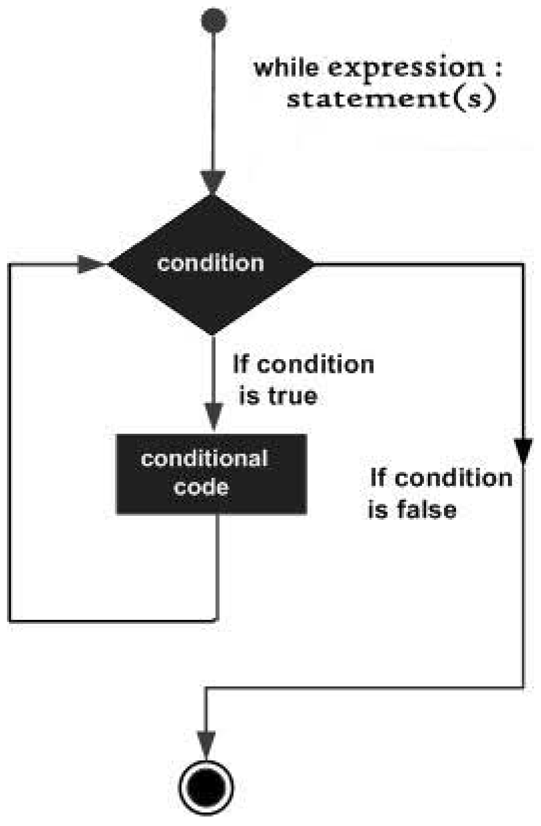
```
while condition:

    statements
```

Statements represents one statement or a suite of statements.

If the condition is true, statements will be executed.

Python interpreter execute statements again and again. Once the condition is false it will come out of the loop.



Program to display even number between 1 to 10.

```
x=1
```

```
while x>=1 and x<=10:
  if x%2==0:
    print(x)
  x=x+1
```

for loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Syntax:

```
for val in sequence:

      statements
```

Program to display character of strings

```
s1='welcome'
for i in s1:
  print(i)
```

```
w
e
l
c
o
m
e
```

Program to display string using its index

```
s2='python'
n=len(s2)
for i in range(n):
  print(s2[i])
```

```
p
y
t
h
o
n
```

To loop through a set of code a specified number of times, we can use the **range()** function, The range() function returns a sequence of numbers, starting from 0 (by default), and increments by 1 (by default), and ends at a specified number.

Syntax:

range(start, stop, step)

start: integer starting from which the sequence of integers is to be returned

stop: integer before which the sequence of integers is to be returned. The range of integers end at stop – 1.

step: integer value which determines the increment between each integer in the sequence

Program to display even number between 1 to 10.

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

```
2
4
6
8
10
```

Program to display numbers from 10 to 1 in descending order.

```
for i in range(10,0,-1):
  print(i)
```

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

Program to find sum of list using for loop.

```
list = [10,20,30,40,50]
sum=0
for i in list:
  sum+=i
print("Sum=",sum)
```

```
Sum= 150
```

Nested loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Synatx:

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

The syntax for a nested while loop statement in Python programming language is as follows

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

A final note on loop nesting is that we can put any type of loop inside of any other type of loop.

Program to print following pattern.

```
1
22
333
4444
55555
```

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

```
1
22
333
4444
55555
```

Else suite

In most of the programming languages (C/C++, Java, etc), the use of else statement has been restricted with the if conditional statements. But Python also allows us to use the else condition with for/while loops.

The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

Example:

When we traverse a list & if element is not found in list, we can display message element not found in else part.

```
group=[1,2,3,4,5]
search=int(input("Enter number to be searched "))
for element in group:
  if search==element:
    print("Element found in group")
    break
else:
  print("Element not found in group")
```

```
Enter number to be searched 7
Element not found in group
```

If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

Example: **Print a message once the condition is false**

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

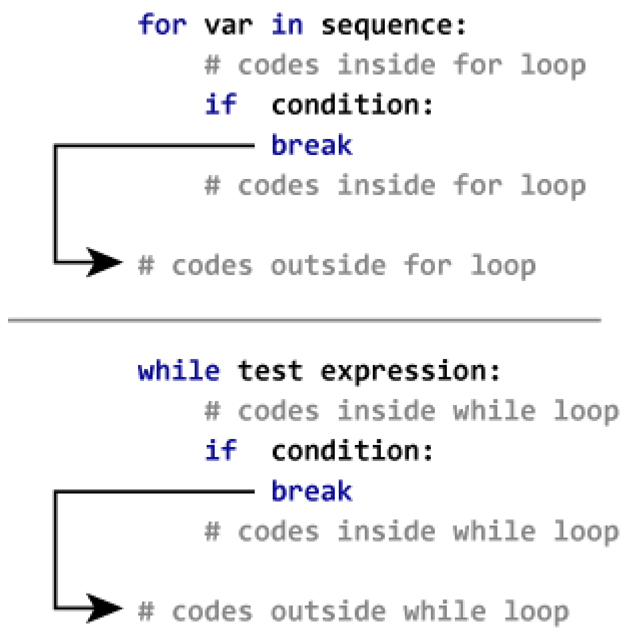
Loop control statement

Loop control statements change execution from its normal sequence.

break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.



```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

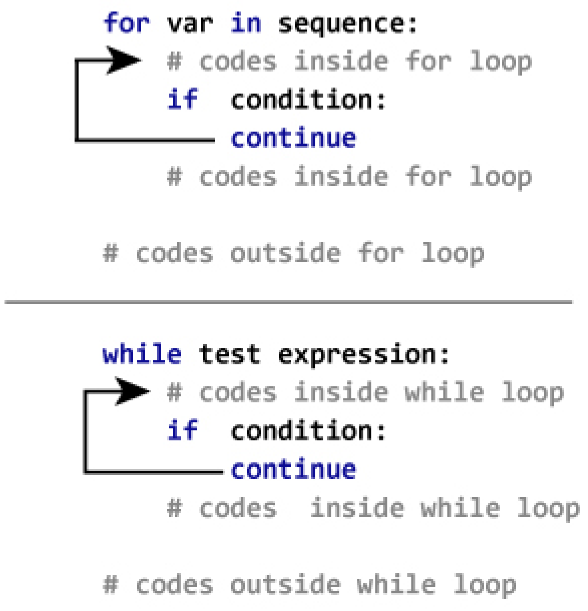
```
# Use of break statement inside the loop
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

```
s
t
r
The end
```

continue statement

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.



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

```
# Program to show the use of continue statement inside loops

for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

```
s
t
r
n
g
The end
```

pass statement

pass statement does not do anything. It is used with if statement or inside a loop to represent no operation.

We use pass statement when we need a statement syntactically but we do not want to do any operation

```
#program to show the use of pass statement
num=[1,2,-4,-5,8]
for i in num:
  if(i>0):
    pass
  else:
    print(i)
```

```
-4
```

```
-5
```

assert statement

assert statement has a condition or expression which is supposed to be always true. If the condition is false assert halts the program and gives an AssertionError.

Syntax:

```
assert expression

      or

assert expression,error message
```

Program to assert that the user enters a number greater than zero

```
x=int(input('Enter a number greater than 0 '))
assert x>0,"Wrong input entered"
print("Entered number is ",x)
```

AssertionError can be handeled using try..except statement.

```
x=int(input('Enter a number greater than 0 '))
try:
    assert x>0
    print("Entered number is ",x)
except:
    print("Wrong input entered")
```

```
Enter a number greater than 0 -6
Wrong input entered
```

return statement

A return statement is used to end the execution of the function call and “returns” the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed.

Syntax:

```
def fun():
  statements
    .
    .
  return [expression]
```

```
def add(a, b):

    # returning sum of a and b
    return a + b
# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))
```

```
Result of add function is 5
```

4s completed at 8:36 PM