Introduction to Python

Assignment operator

In performing various mathematical operations on a variable it is convenient to a shortened notation of Assignment operator prefixed with mathematical operator. For example, the following operation

sum = sum + x

Can be written as

sum+=x; (This is same as adding **x** to **sum** and updating its value.

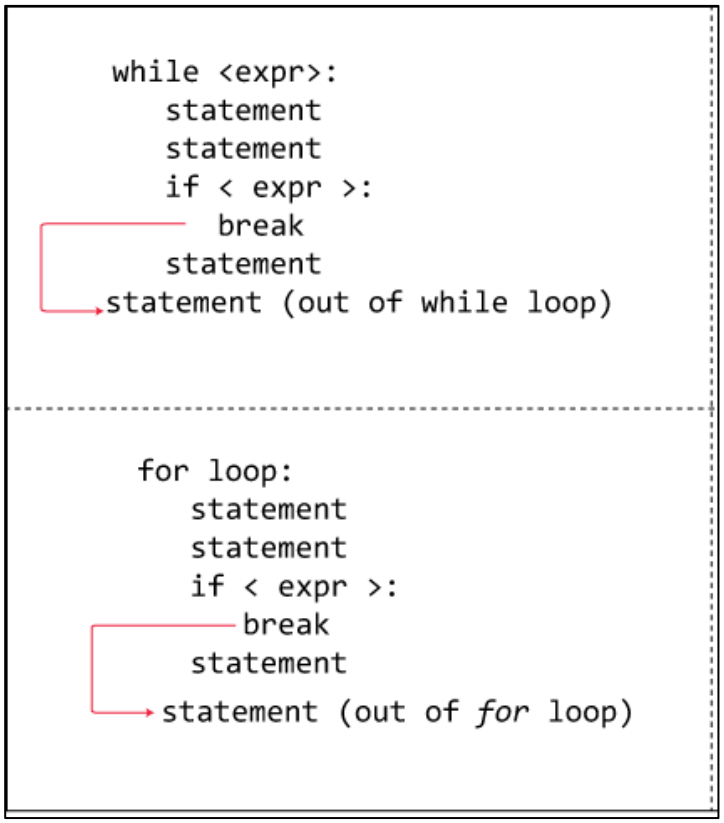
```
a += 2
a -= 2
a *= 2
a /= 2
a **= 2
```

```
a %= 2
a //= 2
```

Introduction to Python

Control of flow using Break/continue/pass

break: The break statement exists or terminates a loop entirely. The statement is used under a loop, usually, after **if** statement. It terminates the loop that contains the statement. In a nested loop, it breaks at the closest loop or the inner nested loop. It can terminate **for/while** loop.



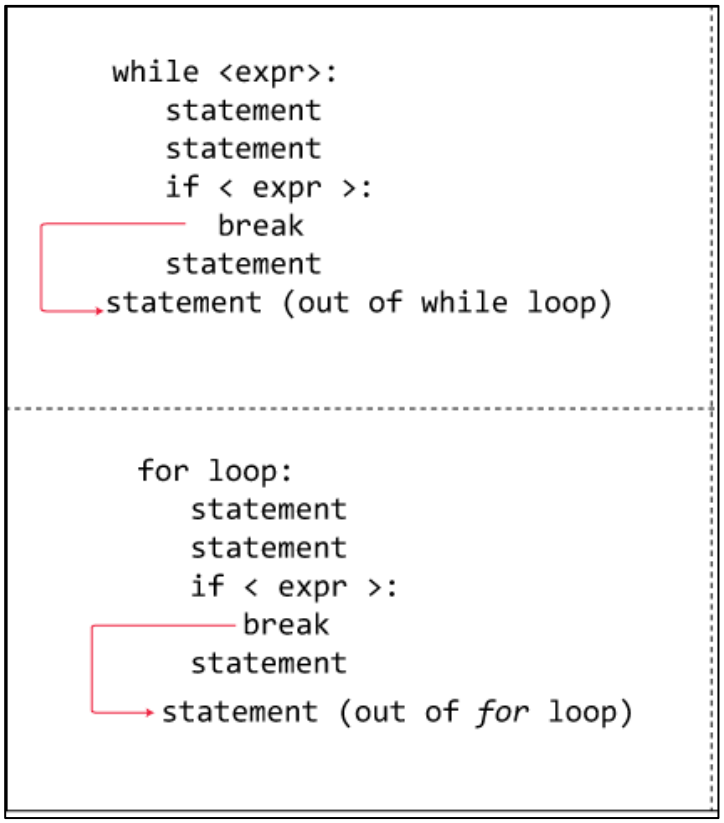

```
n=0
sum=0
while n < 10:
  if n == 8 :
    print('break statement executes and loop stops.')
    break
  print(n)
  sum += n
  n+=1
print(sum)
```

```
0
1
2
3
4
5
6
7
break statement executes and loop stops.
28
```

Introduction to Python

Control of flow using Break/continue/pass

break: The break statement exists or terminates a loop entirely. The statement is used under a loop, usually, after **if** statement. It terminates the loop that contains the statement. In a nested loop, it breaks at the closest loop or the inner nested loop. It can terminate **for/while** loop.



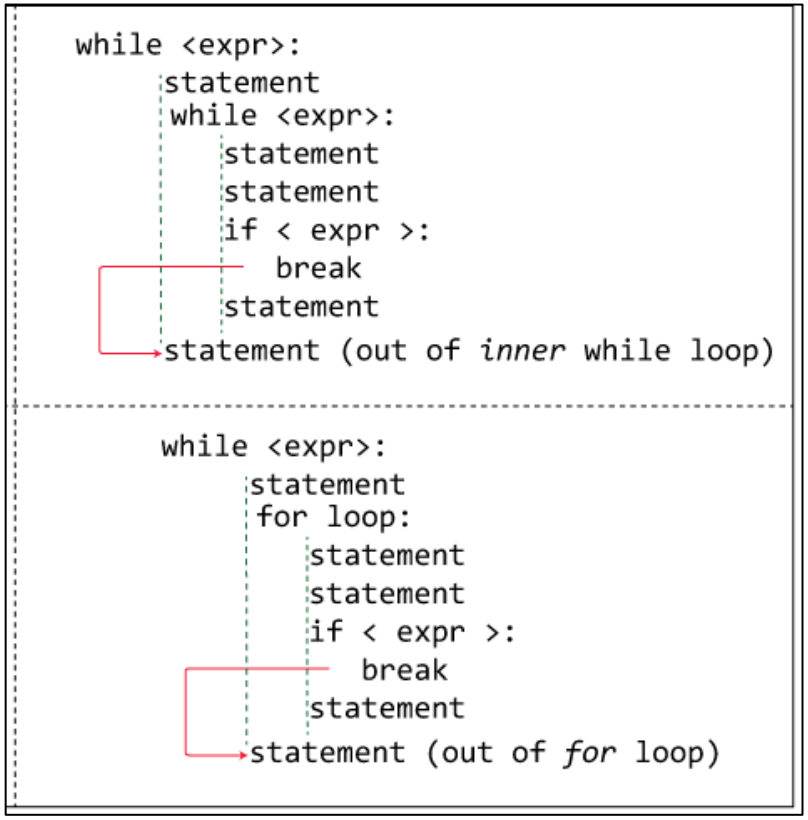

```
n=0
sum=0
for i in range(10):
  if i == 8:
    print('break statement executes and loop stops.')
    break
  print(i)
  sum += i
print(sum)
```

```
0
1
2
3
4
5
6
7
break statement executes and loop stops.
28
```

Introduction to Python

Control of flow using Break/continue/pass

break: A case of nested loop.




```
for i in range(5):
  for j in range(5):
    if j > i:
      break
    a=''.join((str(i),str(j)))
    print(a,end = ' ')
  print('\n')
```

```
00

10 11

20 21 22

30 31 32 33

40 41 42 43 44
```

Control of flow using Break/continue/pass

break: A case of nested loop.

```
for i in range(5):
  for j in range(5):
    print('The inner nested loop is being executed for jth time',i, j)
    if j > i:
      break
    a=''.join((str(i),str(j)))
    print(a,end = ' ')
  print('\n')
```

```
The inner nested loop is being executed for jth time 0 0
00 The inner nested loop is being executed for jth time 0 1


The inner nested loop is being executed for jth time 1 0
10 The inner nested loop is being executed for jth time 1 1
11 The inner nested loop is being executed for jth time 1 2
```

Introduction to Python

Control of flow using Break/continue/pass

continue: The continue statement stops the current loop iteration and it jumps to the top of the loop. The while controlling expression is re-evaluated to see whether loop will execute or terminate. The same can be implemented in for loop.

```
while <expr>:
    statement
    while <expr>:
        statement
        statement
        if < expr >:
            continue
        statement
        statement
    statement (out of inner while loop)
```

```
while <expr>:
    statement
    for loop:
        statement
        statement
        if < expr >:
            continue
        statement
        statement
    statement (out of for loop)
```

```
for i in range(1,11):
  if i == 6:
    continue
  print(i)
```

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

Introduction to Python

Control of flow using Break/continue/pass

continue:

```
sumEven=0
for i in range(1,100):
  if i%2:
    continue
  sumEven+=i
print(sumEven)
```

```
for i in range(5):
  for j in range(5):
    print('The inner nested loop is being executed for jth time',i, j)
    if j > i:
      continue
    a=''.join((str(i),str(j)))
    print(a,end = ' ')
  print('\n')
```

```
00

10 11

20 21 22

30 31 32 33

40 41 42 43 44
```

Introduction to Python

Control of flow using Break/continue/pass

continue:

```
for i in range(5):
  for j in range(5):
    print('The inner nested loop is being executed for jth time',i, j)
    if j > i:
      continue
    a=''.join((str(i),str(j)))
    print(a,end = ' ')
  print('\n')
```

```
The inner nested loop is being executed for jth time 0 0
00 The inner nested loop is being executed for jth time 0 1
The inner nested loop is being executed for jth time 0 2
The inner nested loop is being executed for jth time 0 3
The inner nested loop is being executed for jth time 0 4


The inner nested loop is being executed for jth time 1 0
10 The inner nested loop is being executed for jth time 1 1
11 The inner nested loop is being executed for jth time 1 2
The inner nested loop is being executed for jth time 1 3
The inner nested loop is being executed for jth time 1 4
```

Introduction to Python

Functions

It to achieve modularity and reusability in a program. It can be considered as a self contained block of code that executes a specific or related group of tasks. The function can be called many times in a program without actually writing embedded piece of code again and again. You can think it as a subprogram.

```
def <function name> ( [ optional paramerers] ):
        statement
        statement
        [ return <value> ] # optional
```

```
def func():
  print('Hello World')

func()
```

```
def func(fname):
  print('Hello World ',fname)

fname=input('What is your name  ')
func(fname)
```

```
What is your name  IDC101
Hello World  IDC101
```

Introduction to Python

Functions

```
def func(m):
  m=m+10
  print('The value of variable in function ',m)

n=10
print('Before function the value of n ', n)
func(n)
print('After function the value of n ', n)
```

```
Before function the value of n  10
The value of variable in function  20
After function the value of n  10
```

Introduction to Python

Functions

```
## function to find whether a number is odd or not
def isOdd(m):
  odd=True
  if m % 2 == 0:
    odd=False
  return(odd)

m=11
out=isOdd(m)
if isOdd(m):
  print('The number is Odd')
else:
  print('The number is even')
```