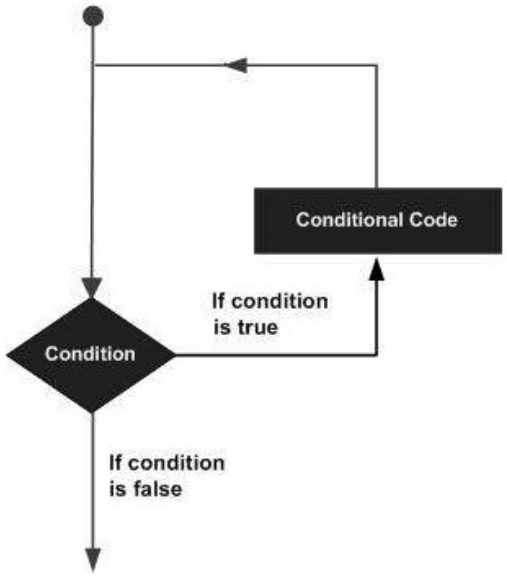
Loops

In general, statements 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 languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement –



Loops

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

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, for or do..while loop.

while Loops

The syntax of a **while** loop in Python programming language is –

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

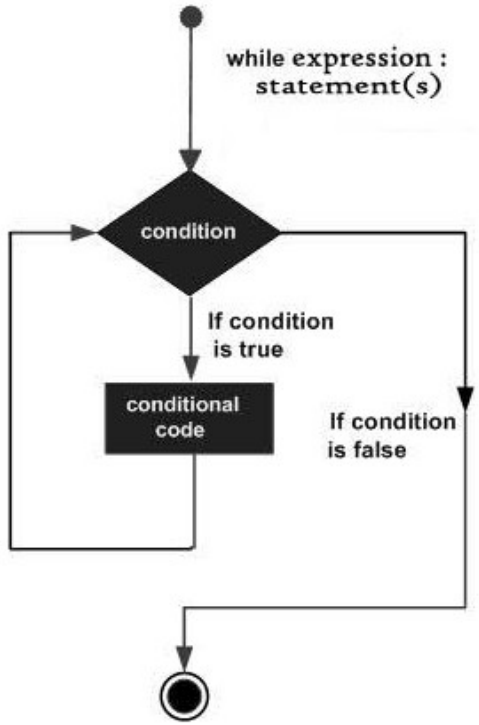
Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

while Loops

Flow Diagram



simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
count = 0
while (count < 9):
   print ("The count is:", count)
   count = count + 1

print ("Good bye!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf
tel)] on win32
Type "copyright", "credits" or
>>>
 RESTART: C:\Documents and Set
ple.py
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
>>> |
```

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
count=int(input("Enter initial value:"))
limit=int(input("Enter limit value:"))
inc=int(input("Enter increment value:"))

while (count < limit):
   print ("The count is:", count)
   count = count + inc

print ("Good bye!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efca
tel)] on win32
Type "copyright", "credits"
>>>
 RESTART: C:\Documents and S
ple.py
Enter initial value:10
Enter limit value:24
Enter increment value:2
The count is: 10
The count is: 12
The count is: 14
The count is: 16
The count is: 18
The count is: 20
The count is: 22
Good bye!
```

Control Loop Variable

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
count=int(input("Enter initial value:"))
limit=int(input("Enter limit value:"))
inc=int(input("Enter increment value:"))

while (count < limit):
   print ("The count is:", count)
   count = count + inc

print ("Good bye!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efca
tel)] on win32
Type "copyright", "credits"
>>>
 RESTART: C:\Documents and S
ple.py
Enter initial value:10
Enter limit value:24
Enter increment value:2
The count is: 10
The count is: 12
The count is: 14
The count is: 16
The count is: 18
The count is: 20
The count is: 22
Good bye!
```

Loops

Three Activities to Coordinate

Three activities of a loop must work together:

1. The initial values must be set up correctly.
2. The *condition* in the `while` statement must be correct.
3. The change in variable(s) must be done correctly.

In the above program we wanted to print the integers "1, 2, 3". Three parts of the program had to be coordinated for this to work correctly.

```
count=int(input("Enter initial value:"))
limit=int(input("Enter limit value:"))
inc=int(input("Enter increment value:"))

while (count < limit):
   print ("The count is:", count)
   count = count + inc

print ("Good bye!")
```

Down Count

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
count=int(input("Enter initial value:"))
limit=int(input("Enter limit value:"))

while (count > limit):
   print ("The count is:", count)
   count = count - 1

print ("Done counting down!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf
tel)] on win32
Type "copyright", "credits" or
>>>
 RESTART: C:\Documents and Set
ple.py
Enter initial value:20
Enter limit value:7
The count is: 20
The count is: 19
The count is: 18
The count is: 17
The count is: 16
The count is: 15
The count is: 14
The count is: 13
The count is: 12
The count is: 11
The count is: 10
The count is: 9
The count is: 8
Done counting down!
```

Loop and other control structures

- **Loop control structure can be used with other control structures : if , if/else**

N=5

C	A	G	O
1	0	0	0
2	1	2	1
3	3	6	4
4	6		9
5	10		
6	15		

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
N=int(input("Enter limit value:"))
sumAll, sumEven, sumOdd=0,0,0

count=1
while (count <=N):
    sumAll=sumAll + count
    if ( count % 2 ==0):
            sumEven=sumEven + count
    else:
            sumOdd=sumOdd + count
    count = count + 1

print ("Sum of all", sumAll)
print ("Sum of even", sumEven)
print ("Sum of odd", sumOdd)
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:
tel)] on win32
Type "copyright", "cr
>>>
 RESTART: C:\Document
ple.py
Enter limit value:15
Sum of all 120
Sum of even 56
Sum of odd 64
>>>
```

Nested Loop

- **Loops may be nested, one loop in another loop**

Nested Loop

Rows of Stars

We want a program that writes out five rows of stars, such as the following:

```
*******
*******
*******
*******
*******
```

This could be done with a while loop that iterates five times. Each iteration could execute

nestedw.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/13/nestedw.py (3.4.4)

File Edit Format Run Options Window Help

```
# collect input data from user
numRows  = int(input("How many Rows? "))
numStars = int(input("How many Stars per Row? "))

row  =  1
while ( row <= numRows ):
    star = 1
    while ( star <= numStars ):
        print("*", end=" ")
        star = star + 1
    print()          # need to do this to end each line
    row = row + 1;
print("Program Terminate")


#>>>
#Output
#>>>
    How many Rows? 4
    How many Stars per Row? 5
    * * * * *
    * * * * *
    * * * * *
    * * * * *
    Program Terminate
>>>
```

Types of Loop

- Counter Controlled Loop
- Sentinel Controlled Loop
- Result Controlled Loop

Counter Controlled Loop

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
count=int(input("Enter initial value:"))
limit=int(input("Enter limit value:"))
inc=int(input("Enter increment value:"))

while (count < limit):
   print ("The count is:", count)
   count = count + inc

print ("Good bye!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efca
tel)] on win32
Type "copyright", "credits"
>>>
 RESTART: C:\Documents and S
ple.py
Enter initial value:10
Enter limit value:24
Enter increment value:2
The count is: 10
The count is: 12
The count is: 14
The count is: 16
The count is: 18
The count is: 20
The count is: 22
Good bye!
```

Sentinel Controlled Loop

senloop.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\senloop.py (3.4.4)

File Edit Format Run Options Window Help

```
value=int(input("Enter first integer (or 0 to quit):"))
sum=0

while (value!=0):
    sum=sum + value
    value=int(input("Enter next integer (or 0 to quit):"))

print ("Sum of the integers:", sum)
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf5a6, De
tel)] on win32
Type "copyright", "credits" or "licen
>>>
 RESTART: C:\Documents and Settings\a
loop.py
Enter first integer (or 0 to quit):1
Enter next integer (or 0 to quit):2
Enter next integer (or 0 to quit):3
Enter next integer (or 0 to quit):4
Enter next integer (or 0 to quit):0
Sum of the integers: 10
>>>
```

Sentinel Controlled Loop

addup.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/test/addup.py (3.4.4)

File Edit Format Run Options Window Help

```
value=int(input("Enter first integer (or 0 to quit):"))
sum, count=0, 1


while (value!=0):
    sum=sum + value
    count=count+1
    if (count ==2):
        suffix="nd"
    elif (count ==3):
        suffix="rd"
    else:
        suffix="th"
    print("Enter the", count , suffix, "integer (or 0 to quit):", end="")
    value=int(input())

print ("Sum of the integers:", sum)
```

value	sum	Co	Suffix
2	0	1	nd
5	2	2	rd
6	7	3	th
9	13	4	th
	22	5	

Enter the 2 nd integer

3 rd

4 th

5 th

Sentinel Controlled Loop

```
Python 3.4.4 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 1
tel)] on win32
Type "copyright", "credits" or "license()" for mo
>>>
 RESTART: C:/Documents and Settings/admin/Desktop,
up.py
Enter first integer (or 0 to quit):2
Enter the 2 nd integer (or 0 to quit):4
Enter the 3 rd integer (or 0 to quit):6
Enter the 4 th integer (or 0 to quit):8
Enter the 5 th integer (or 0 to quit):10
Enter the 6 th integer (or 0 to quit):0
Sum of the integers: 30
>>>
```

Sentinel Controlled Loop

New Example

Sometimes the user is asked explicitly if the loop should continue. The user enters "yes" or "no" (or maybe "y" or "n"). Now the sentinel is of type String or char. The next example illustrates this: Say that you are interested in the value of the polynomial:

$$7x^3 - 3x^2 + 4x - 12$$

for various values of x. The value x is a double precision value. For example, when x == 2.0 the polynomial is equal to:

$$7*2^3 - 3*2^2 + 4*2 - 12 = 7*8 - 3*4 + 8 - 12 = 40$$

The program lets the user enter various values for x and see the result for each one.

EvalPoly.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/test/EvalPoly.py (3.4.4)

File Edit Format Run Options Window Help

```
response="y"

while (response=="y"):
   x=float(input("Enter a value for x: "))
   result=7*x*x*x-3*x*x+4*x-12
   print("The value of the polynomial at x", x, "is: " , result)
   response=input("Continue (y or n)? ")
print("Good Bye")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Documents and Settings/admin/Desktop/intro-python/exa
lPoly.py
Enter a value for x: -1
The value of the polynomial at x -1.0 is:  -26.0
Continue (y or n)? y
Enter a value for x: 1.178
The value of the polynomial at x 1.178 is:  -0.008209736000004852
Continue (y or n)? n
Good Bye
>>>
```

Result Controlled Loop

- Counter Controlled Loop
- Sentinel Controlled Loop
- Result Controlled Loop

Result Controlled Loop

A **third** kind of loop can be built from the fundamental control statements . This is the **result-controlled** loop. (Other names for it are *free loop* and *general loop*). A result-controlled loop keeps looping until the computation has reached a particular goal. It is like the instruction in a cookie recipe that says "keep stirring until the ingredients are thoroughly blended." You know when to quit only when the desired result has been achieved.

Million Dollar Question

Usually banks pay interest daily or monthly, but for simplicity let us stick with inte
paid once at the end of each year. At the end of the second year you will have $1
+ $1050*0.05 = $1102.50. Here is what your account looks like at the end of the
several years:

year	Interest for the Year	End of Year Amount
1	1000*0.05=50	1050.00
2	1050*0.05=52.5	1102.50
3	1102.50*0.05=55.125	1157.625
4	1157.625*0.05=57.88125	1215.50625
5	1215.50625*0.05=60.77531	1276.28156

What if you are interested in becoming a millionaire? How long will it take to reac
million dollars? There are formulas for this. (Computer spreadsheets have these
formulas built in, as do financial electronic calculators.) But pretend that you don'
know that.

Result Controlled Loop

Dollar.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/test/Dollar.py (3.4.4)

File Edit Format Run Options Window Help

```
dollars = 1000.00
rate = 0.05
year = 0

while ( dollars < 1000000.00 ):
      dollars = dollars + dollars*rate

      # add another year's interest
      year    = year + 1

print("It took " , year , " years to reach your goal.")
```

```
output:

   It took  142  years to reach your goal.
```

Dollar.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/test/Dollar.py (3.4.4)

File Edit Format Run Options Window Help

```
dollars = 1000.00
rate = 0.05
year = 0

while ( dollars < 1000000.00 ):
      # add another year's interest
      dollars = dollars + dollars*rate

      #add in this year's contribution
      dollars = dollars + 1000 ;

      year    = year + 1

print("It took " , year , " years to reach your goal.")
```

output:

```
   It took  80  years to reach your goal.
```

Dollar.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/test/Dollar.py (3.4.4)

File Edit Format Run Options Window Help

```
initialAmount = 1000.00 ;
dollars = 0.0
rate =  0.0

while ( dollars < 1000000.00 ):
       # change to the next rate
       rate = rate + 0.001
       # compute the dollars after 40 years at the current rate
       year =  1
       dollars = initialAmount
       while (  year <= 40 ) :
         dollars = dollars + dollars*rate   # add another year's interest
         dollars = dollars + 1000           # add in this year's contribution
         year    =  year + 1


print("After 40 years at " , rate*100,
      " percent interest you will have " , dollars , " dollars" )


#output:

    After 40 years at  12.600000000000009
    percent interest you will have  1021746.3104116677  dollars
```

Using else Statement with Loops

- **Python supports to have an else statement associated with a loop statement.**
  - If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.
  - If the else statement is used with a while loop, the else statement is executed when the condition becomes false.
- **The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed**

Using else Statement with Loops

elsloop.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/13/elsloop.py (3.4.4)

File Edit Format Run Options Window Help

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

print("Program Terminate")


#>>>
#Output
#>>>
    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
    Program Terminate
```

The Infinite Loop

- A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.
- An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

The Infinite Loop

inf.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/13/inf.py (3.4.4)

File Edit Format Run Options Window Help

```
var = 1
while (var == 1) :  # This constructs an infinite loop
   num = input("Enter a number  :")
   print ("You entered: ", num)



>>>
Output
     Enter a number  :5
     You entered:  5
     Enter a number  :2
     You entered:  2
     Enter a number  :7
     You entered:  7
     Enter a number  :
```

The Infinite Loop

inf.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/13/inf.py (3.4.4)

File Edit Format Run Options Window Help

```
while (True):
    num=int(input('Enter a number : '))
    print('You entered: ', num)
```

```
>>>
Output
>>>
    Enter a number  :5
    You entered:  5
    Enter a number  :3
    You entered:  3
    Enter a number  :6
    You entered:  6
    Enter a number  :
```

break statement

- **Terminates the loop statement and transfers execution to the statement immediately following the loop.**
- **It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.**
- **The most common use for 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 the execution of the innermost loop and start executing the next line of code after the block.**

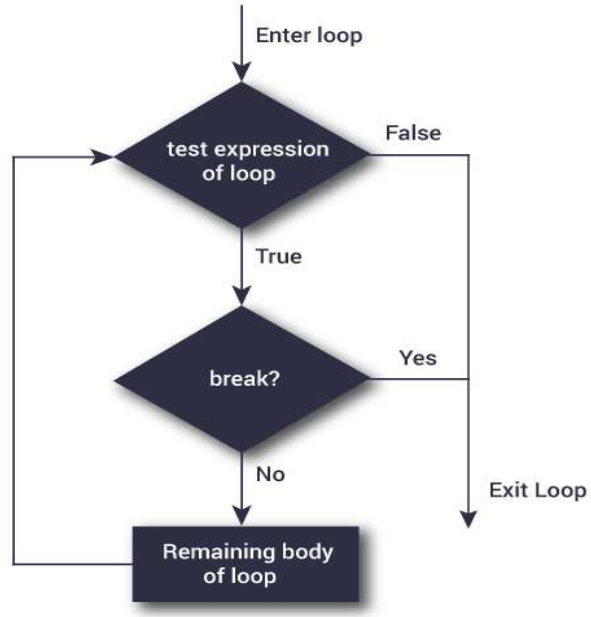
break statement

Syntax

The syntax for a **break** statement in Python is as follows –

```
break
```

Flow Diagram



The Infinite Loop break

inf.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/13/inf.py (3.4.4)

File Edit Format Run Options Window Help

```
while (True) :          # This constructs an infinite loop
    num = input("Enter a number or 0 for exit :")
    print ("You entered: ", num)
    if (num == "0"):
        break
print("Program Terminate")



#>>>
#Output
#>>>

    Enter a number or 0 for exit :5
    You entered:  5
    Enter a number or 0 for exit :3
    You entered:  3
    Enter a number or 0 for exit :0
    You entered:  0
    Program Terminate
```

num = "0"

while (num != "0"):

num = input

print (———)

print("Program Terminate")

Break and else

bn.py - /home/nowzari/Desktop/python/e

File Edit Format Run Options Window Help

```
n = 5
while n > 0:
    n = n - 1
    if n == 2:
        break
    print(n)
else:
    print("Loop is finished")
```

n

5

4

3

2

4

3

continue statement

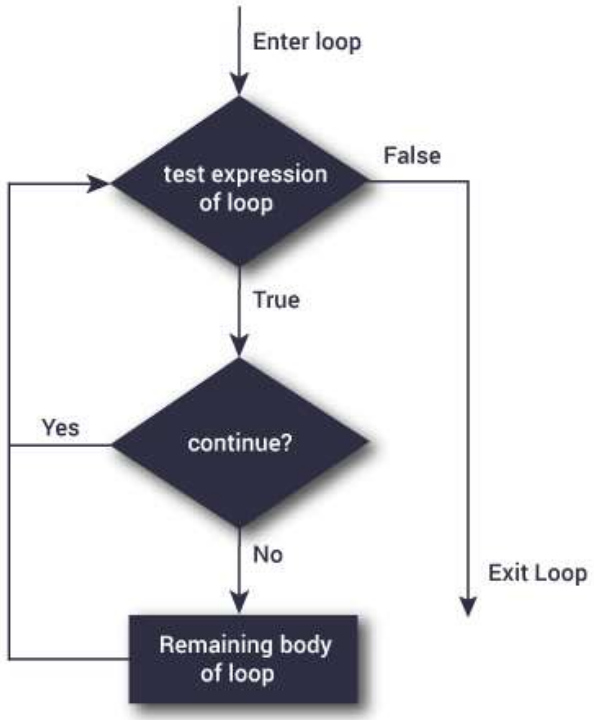
- **Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.**
- **It returns the control to the beginning of the while loop.. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.**
- **The continue statement can be used in both *while* and *for* loops.**

continue statement

Syntax

```
continue
```

Flow Diagram



continue statement

cont.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/13/cont.py (3.4.4)

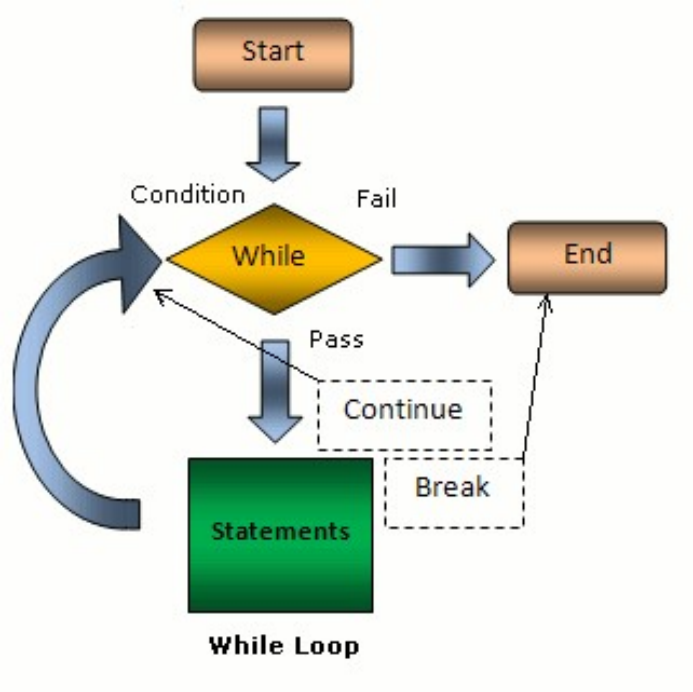
File Edit Format Run Options Window Help

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

print("Program Terminate")


#>>>
#Output
#>>>
     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
     Program Terminate
```

Break and continue statement



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.**
- **The pass statement is a *null* operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written**

```
command = None
while (command != '3'):
    command = input("Press 1 to pass, 2 to continue, or 3 to exit: ")
    if (command == '1'):
        print ("passing")
        pass
    elif (command == '2'):
        print ("continuing")
        continue
    else:
        print ("othering")
    print ("end of loop reached")

print("Program Terminate")


#>>>
#Output
#>>>
    Press 1 to pass, 2 to continue, or 3 to exit: 2
    continuing
    Press 1 to pass, 2 to continue, or 3 to exit: 5
    othering
    end of loop reached
    Press 1 to pass, 2 to continue, or 3 to exit: 1
    passing
    end of loop reached
    Press 1 to pass, 2 to continue, or 3 to exit: 3
    othering
    end of loop reached
    Program Terminate
```

Use Integers for Counting

You might want a double or float for the loop control variable. You would add 0.1 to it each iteration. This would nearly work, but leads to errors. The value 0.1 cannot be accurately represented in binary. A loop that repeatedly adds 0.1 to a variable will accumulate errors.

Just for fun, here is a program fragment that does just that. Enter various limit amounts and see how much error there is:

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\test\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
count=float(input("Enter initial value:"))
limit=float(input("Enter limit value:"))
inc=0.1
while (count < limit):
   print ("The count is:", count)
   count = count + inc

print ("Counting down!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
>>>
 RESTART: C:\Documents and Settings
ple.py
Enter initial value:0
Enter limit value:1.9
The count is: 0.0
The count is: 0.1
The count is: 0.2
The count is: 0.30000000000000004
The count is: 0.4
The count is: 0.5
The count is: 0.6
The count is: 0.7
The count is: 0.7999999999999999
The count is: 0.8999999999999999
The count is: 0.9999999999999999
The count is: 1.0999999999999999
The count is: 1.2
The count is: 1.3
The count is: 1.4000000000000001
The count is: 1.5000000000000002
The count is: 1.6000000000000003
The count is: 1.7000000000000004
The count is: 1.8000000000000005
Counting down!
>>>
```

End