Iteration

Iteration repeats the execution of a sequence of code. Iteration is useful for solving many programming problems. Iteration and conditional execution form the basis for algorithm construction.

1. The while Statement

Consider the following program which does the count-down of launching a rocket.

Program	Output
print(1) print(2) print(3) print(4) print(5) print(6) print(7) print(8) print(9) print(10)	1 2 3 4 5 6 7 8 9 10

How are you going to write the code to count up to 10,000? You could, but that would be impractical! Counting is such a common activity, and computers routinely count up to very large values, so there must be a better way.

What we really would like to do is print the value of a variable (call it count), then increment the variable (ie. count = count + 1 or count + = 1), and repeat this process until the variable (from count = 1 to count = 10000). This process of executing the same section of code over and over is known as iteration, or looping.

Python has two different statements for looping; while statement and for statement.

The above program can be rewritten to :

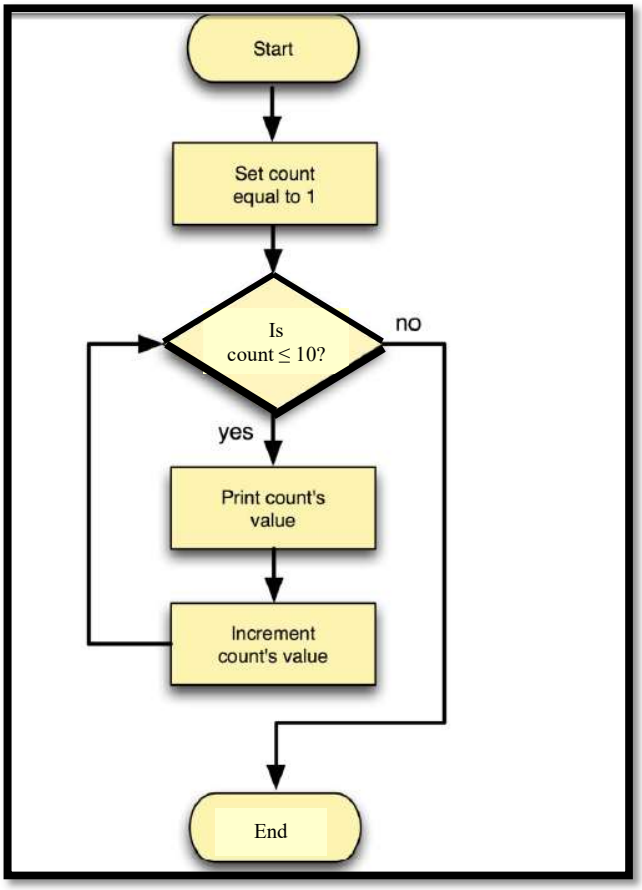
Using While statement	Using For statement
count = 1 #initialize the counter while count <>11: *#pay attention to ":"* print (count) *#pay attention to indentation* count = count + 1 #or count +=1 *#Note the block of statements after while. They are called the body of the loop. They must be indented.*	For count in range(1, 11): print(count)

while count < 11 (or while count <=10) begins the while statement. The expression following the while keyword is the condition that determines if the statement block is executed or stopped to execute. As long as the condition is true, the program executes the code block over and over again (the indented section). When the condition becomes false, the loop is finished.

The while statement has the general form:
while *condition* **:**
line 1
line 2 } *body of the loop*
...

Except for the reserved word *while* instead of *if*, *while* statements look identical to *if* statements. *While* is the *if* statement in repetition. Refer to the following diagram.

The condition is checked before the body is executed, and then checked again each time after the block has been executed. If the condition is initially false the block is not executed. If the condition is initially true, the block is executed repeatedly until the condition becomes false, at which point the loop terminates.



Activity 1:

1. Write a program to count down from 10 for the launching of a rocket. A sample output is as follows:

10
9
...
Blast Off!

2. Write a program to add up all positive integers entered by the user. When the user enter a negative number, the program stops and the sum of the integers will be shown. A sample output is as follows:

Please enter a positive number:
5
Please enter a positive number:
10
Please enter a positive number:
33
Please enter a positive number:
-1
The sum of all the numbers you have entered = 48

Definite Loops vs. Indefinite Loops

Consider Program 2 from Activity 1

Program

sum = 0
num = int(input("Please enter a positive number:"))
while num>0:
  sum = sum + num
  num = int(input("Please enter a positive number:"))
print("The sum of all the numbers you have entered = ", sum)

This kind of loop is called the indefinite loop since we do not know the number of iterations the loop performs. The number of iterations depends on the input provided by the user.

While-loop is ideal for indefinite loop.

If we want to add up the 5 numbers entered by the user, we can use the definite loop; ie. the For-loop.

2. The For statement

The above program can be rewritten to add the 5 numbers entered by the user.

Program

sum=0
for x in range(1, 6):
  num=int(input("Please enter 5 positive number: "))
  sum = sum + num
print("The sum of all the numbers you have entered = ", sum)

Python provides a more convenient way to express a definite loop. The for statement iterates over a range of values. These values can be a numeric range, string or list or tuple.

- Begin is the first value in the range; if omitted, the default value is 0
- End is one past the last value in the range; the end value may not be omitted
- Step is the amount to increment or decrement; if the step parameter is omitted, it defaults to 1

The for loop has the general form:

for in range(*begin, end, step***):**

line 1
line 2
...
} *body of the loop*

For example, the following program will print out the odd number from 1 to 20

```
for x in range(1,20, 2):
  print(x)
```

The range function is very flexible. *Begin* integer can be smaller than the *end* integer. Besides, negative integers can also be used in the *step* integer. Consider the program below:

```
for n in range (21, 0, -3)
 print(n, " ", end=" ")                    #gives the output of 21  18  15  12  9  6  3
```

Program practice (submit 2 and 3)

Write a program to

1. print out the first ten numbers of 13's multiples
2. print out the sum of 100 consecutive numbers starting from 1.
3. print out the sequence of the first 20 Fibonacci sequence (1, 1, 2, 3, 5, 8 …..). ie. starting from the third term, each term is the sum of the last two terms.