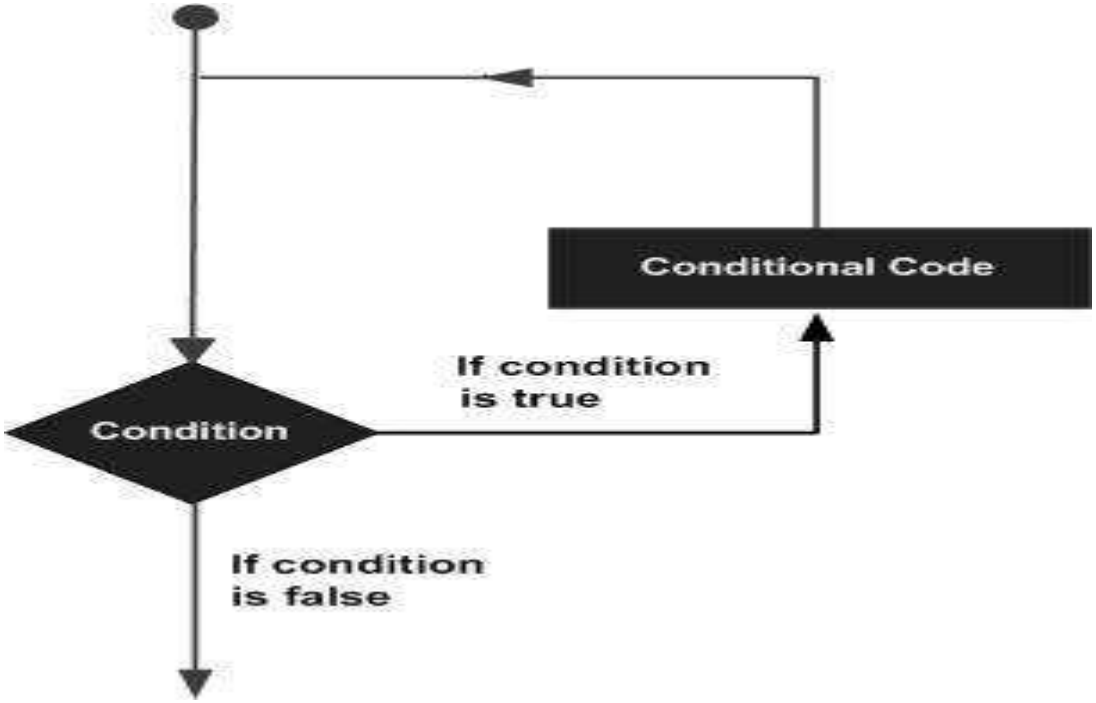
LOOPS in PYTHON

- 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.

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



- **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 loop	You can use one or more loop inside any another while, for or do..while loop.

while loop

- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

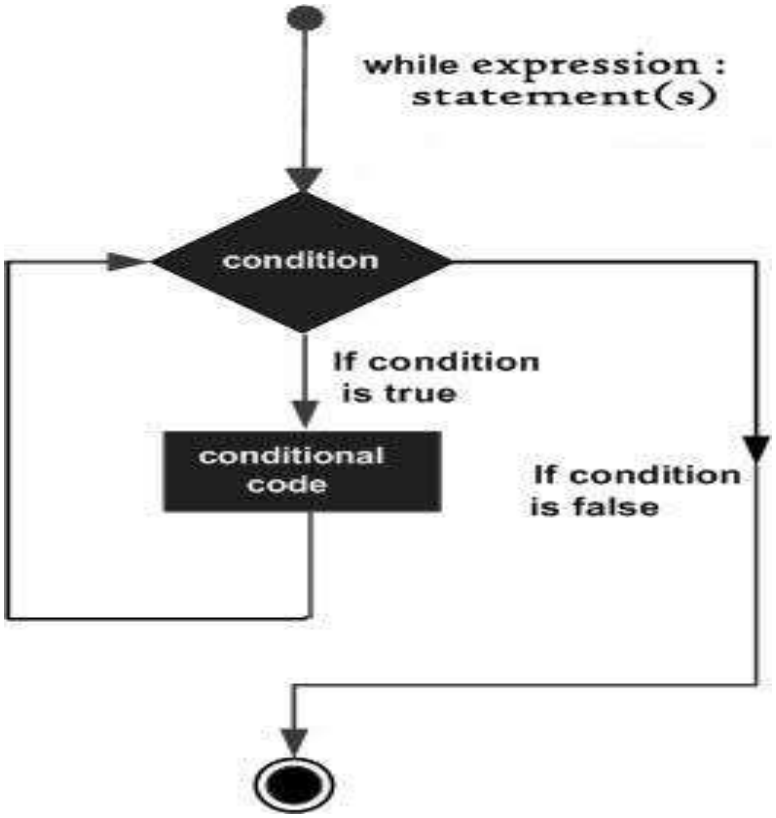
Syntax

- 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.**
- **<u>Flow Diagram</u>**



- **Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.**

❖ **Example1:**

```
count = 0
while (count < 5):
        print  ('The count is:‘, count)
        count = count + 1
print  ("Good bye!“)
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
Good bye!
```

While Loop

Example 2:

```
Var =1
while var == 1:
    num=input("Enter Number:")
    print("You Entered number is:", num)
print("GOOD BYE")
```

```
Enter Number:56
You Entered number is: 56
GOOD BYE
Enter Number:52
You Entered number is: 52
GOOD BYE
Enter Number:1
You Entered number is: 1
GOOD BYE
```

- **Previous example is the The Infinite Loop example.**

- **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.**

Print with %d %s

- **%s is used as a placeholder for string values you want to inject into a formatted string.**
- **%d is used as a placeholder for numeric or decimal values.**

```
name = ‘Ravi'
percentage= 78
print (‘Name %s  Percentage %d' % (name, percentage))
```

Exercise 1

- **Display the table of a given number up to 12 using while loop.**
- **Enter any int Number: 10**
- **1 X 10 = 10**
- **2 X 10 = 20**
- **……**
- **……**
- **12 X 10= 120**

Exercise 1 : Solution

```
num = 1
n = int(input("Enter a number for table: "))
while(num <=12):
    print('%d X %d =\t %d' %(n,num,n*num))
    num=num+1
```

Exercise 2

- **Using while loop display whether given number is Armstrong Number or not.**
- **Test for : 1, 153, 370, 371**

Exercise 2 : Solution

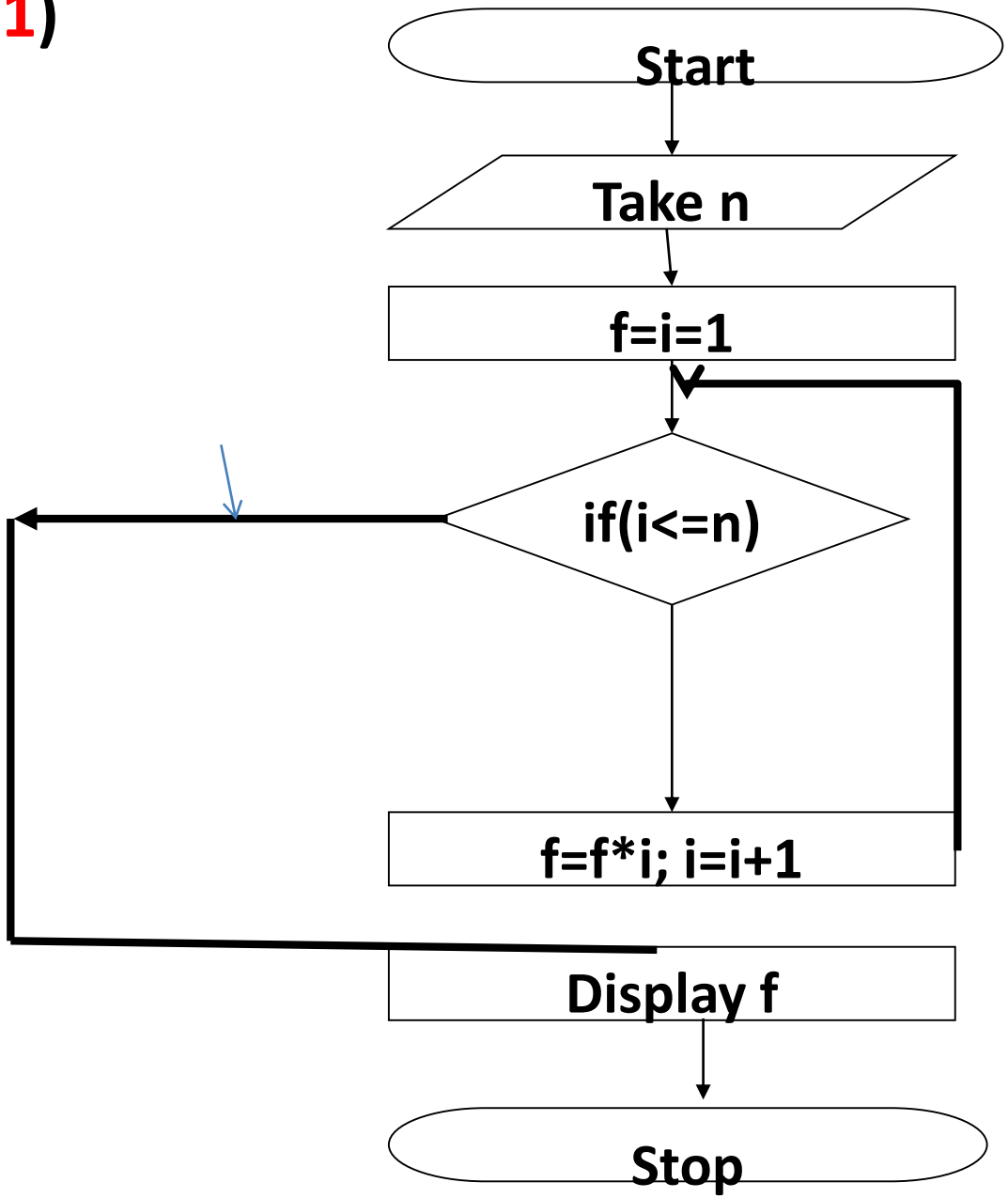
- '''ARMSTRONG NUMBER by using FOR:
- ----------'''

```
num = int(input("Enter a number: "))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum = sum+digit ** 3
    temp = temp// 10
if num == sum:
    print(num,"is an Armstrong number")
else:
    print(num,"is not an Armstrong number")
```

Write a Python program to find the factorial of a positive integer (Exp 1)

- **ALGORITHM:**
- **STEP1:START**
- **STEP2: INPUT N,I,F**
- **STEP3: F=I=1**
- **STEP4: IF(I<=N)**
  - **STEP5: F=F*I**
  - **STEP6: I=I+1**
- **STEP7: REPEAT FROM STEP 5 TO STEP 6 TILL STEP 4 IS TRUE**
- **STEP8: PRINT F**
- **STEP9: STOP**

Write a Python program to find the factorial of a positive integer (Exp 1)



Find the factorial of a given number

```
f=i=1
n = int(input("Factorial of which number? "))
while i<=n:
    f*=i
    i=i+1
print("The factorial of %d is %d" %(n,f))
```

break

- **The *break* Statement:**
- **The break statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found 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.

Example3:

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

print ("Good bye!")
```

Continue

- **The *continue* Statement:**
- **The continue statement in Python 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 - while

```
var = 5                    # continue in while
while var > 0:
   var = var -1
   if var == 3:
      continue
   print ('Current variable value :', var)
print ("Good bye!")
Current variable value : 4
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

Use end to suppress new line in print()

- Printing on same line Python
- Since the python print() function by default ends with newline.
- Python has a predefined format if you use print(a_variable) then it will go to next line automatically.
- But sometime it may happen that we don't want to go to next line but want to print on the same line.
- print("*", end ="")
- print("*", end =“ ")
- print("*", end =‘ ‘)

Exercise

- '''
- **Using while loop write a program to print a triangle with *. If user gives 3 then**
- *
- **
- ***
- **Take number of rows from user and print the triangle as shown above**
- '”

Exercise Solution

```
row = 1
lines = int(input("Enter number of lines "))
if lines >=2:
    while row <= lines:
        col = 1
        while col <= row:
            print("*", end ="")
            col = col +1
        print()
        row = row +1
else:
    print("Give min 2 lines")
```

The for iterator

The for loop

- **Python for Loops**
- **A for loop is used for iterating over a sequence (that is either a string, range, list, a tuple, a dictionary or a set).**
- **This is less like the for keyword in other programming language, and works more like an iterator method as found in other object-orientated programming languages.**
- **With the for loop we can execute a set of statements, once for each item in a range, string, list, tuple, set etc.**

The for loop : Membership & Identity operator

- **Membership Operators in and not in**
- Membership operators are operators used to validate the membership of a value.
- It test for membership in a sequence, such as strings, lists, or tuples.
- There are two membership **in and not in**
- **in operator :** The ‘in’ operator is used to check if a value exists in a sequence or not.
- Evaluates to true if it finds a variable in the specified sequence and false otherwise.

The for loop : Membership operator

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

The for loop : Membership operator

```
for letter in 'Python':
    print (letter)
```

Get the output in one line with a blank between each character

```
for letter in 'Python':
    print (letter,end=' ')
```

for Loop - break

Take the string as 'Python. Search each character in the string. If the character is ‘h’ then terminate the loop and print

Current Letter h

Otherwise print each letter

Current Letter P

Current Letter y

…

for Loop - break

```
for letter in 'Python': # First Example
   if letter == 'h':
      break
   print ('Current Letter :', letter)
print ('Current Letter :', letter)
```

Output

- Current Letter : P
- Current Letter : y
- Current Letter : t
- Current Letter : h

The for loop : Membership operator

- '''
- **using membership operators in and not in write a program.**
- **Declare and initialize an object ch with P. Using in and not in display**
- **P if it is the character in 'Python' otherwise display**
- **This character is not 'P'**
- '''

The for loop : Membership operator

```
ch = 'P'
for char in 'Python':
    if (ch not in char):
        print("This character is not 'P'")
    else:
        print (char)
```

The for loop : Membership operator

- '''

Declare and initialize a string with value 'ram rahim and robert'

Identify as to howmany times character 'r' appeared in the string

Use for loop

'''

```
myStr = "ram rahim and robert"
count = 0
for char in myStr:
    if char == 'r':
        count += 1
print("r appeared %d times"%(count))
```

The for loop : Membership operator

- '''

Declare and initialize a string with value 'ram rahim and robert'

Identify as to howmany times character 'r' appeared in the string and

howmany times 'a' appeared in the same string

Use for loop

'''

The for loop : Membership operator

```
myStr = "ram rahim and robert"
countR = 0
countA = 0
for char in myStr:
    if char == 'r':
        countR += 1
    elif char == 'a':
        countA +=1
print("r appeared %d times a appeared %d
   times"%(countR,countA))
```

for Loop Example

```
'''
Draw a pattern of '*' based on the input given by user
If user gives 3
*
**
***
'''
num = int(input("Pattern of how many lines? "))
for n in range(1,num+1):
    print('+' *n)
```

For - Continue

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print ('Current Letter :', letter)
print("out of for")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
out of for
```

The *for Loop*

```
/* A Fibonacci Sequence is defined as follows:
the first and second terms in the sequence are 0 and 1. Subsequent terms are found by adding the preceding two terms in the sequence.
Write a C program to generate the first n terms of the sequence.
*/

#include <stdio.h>

void main()
{
int num1=0, num2=1,no,counter,fab;
printf("<============PROGRAM TO FIND THE FIBONACCI SERIES UP TO N NO. IN SERIES=========>");
printf("\n\n\n\t\tENTER LENGTH OF SERIES (N) : ");
scanf("%d",&no);

printf("\n\n\t\t\t<----FIBONACCI SERIES---->");
printf("\n\n\t\t%d  %d",num1,num2);

//LOOP WILL RUN FOR 2 TIME LESS IN SERIES AS THESE WAS PRINTED IN ADVANCE
for(counter = 1; counter <= no-2; counter++)
{
fab=num1 + num2;
printf("  %d",fab);
num1=num2;
num2=fab;
}
getch();
}
```

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 for loop -

- Write a program which asks user to enter and integer number and gives the output whether it is PRIME NUMBER or NOT PRIME NUMBER using FOR:

Using for loop -

- PRIME NUMBER PROGRAM using FOR:

```
----------------------------
n=int(input("Enter Number: "))
if n>1:
   for i in range(2,n):
      if(n%i)==0:
         print(n,"is not a prime number")
         print(i,"times",n//i,"is",n)
         break
   else:
      print(n,"is a prime number")
```

OUTPUT:

```
-------
>>>
Enter Number: 100
100 is not a prime number
2 times 100

>>>
Enter Number: 17
17 is a prime number
```

Using for loop -

- Write a program which asks user to enter range of integer number and gives the output whether it is all PRIME NUMBER in the range using FOR:

Using for loop -

- PRIME NUMBER PROGRAM using FOR:

```
----------------------------
n=int(input("Enter Number: "))
if n>1:
   for i in range(2,n):
      if(n%i)==0:
         print(n,"is not a prime number")
         print(i,"times",n//i,"is",n)
         break
   else:
      print(n,"is a prime number")
```

OUTPUT:

```
-------
>>>
Enter Number: 100
100 is not a prime number
2 times 100

>>>
Enter Number: 17
17 is a prime number
```

Using else: in while:

Example3:

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

```
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
```

For - Pass

- 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 yet (e.g., in stubs for example):
-

For - Pass

```
for letter in 'Python':
   if letter == 'h':
      pass
      print ('This is pass block')
      break
   print ('Current Letter :', letter)
print ("Good bye!")
```

Output

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

For - Pass

- The preceding code does not execute any statement or code if the value of *letter* is 'h'. The *pass* statement is helpful when you have created a code block but it is no longer required.
- You can then remove the statements inside the block but let the block remain with a pass statement so that it doesn't interfere with other parts of the code.