Unit -2

Topics to be covered

Control Statements: Definite iteration for Loop, Formatting Text for output, Selection if and if else Statement, Conditional Iteration the While Loop

Strings and Text Files: Accessing Character and Substring in Strings, Data Encryption, Strings and Number Systems, String Methods Text Files.

Control Statements

Introduction: The control statements are the program statements that allow the computer to select a part of the code or repeat an action for specified number of times. These statements are categorized into categories: Selection statements, Iterative/Loop statements and Jump statements. First, we will learn **for** iterative statement, then we will learn selection statements **if** and **else**, later we will learn conditional iteration statement **while**.

Definite Iteration: The for Loop

The repetition statements execute an action for specified number of times. These statements are also known as '***Loops***'. Each repetition of an action is called '***Pass***', or '***Iteration***'. If a loop able to repeat an action for a predefined number of times then it is said to be ***definite Iteration***. The Python's for loop supports the definite iteration.

Executing a statement, a Given Number of Times

If we want to print the line "It's Alive! It's Alive! It's Alive". This can be easily printed using the definite loop statement as follow:

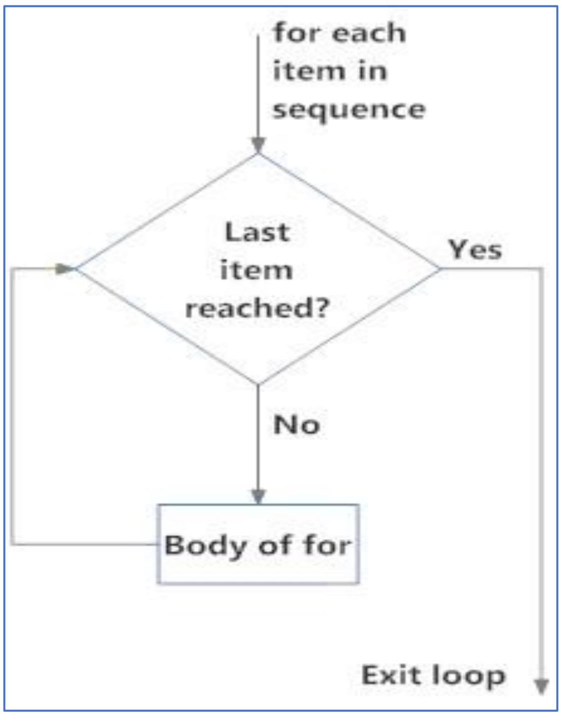
Exp1.py	Output

`for i in range(3):`
`    print("It's Alive!",end=" ")`
	It's Alive! It's Alive! It's Alive!

This for loop repeatedly calls one function that print () function. The constant 3 for the range() function specifies the number of times to repeat action. If we want it 10 or 100 times then we can change 3 to 10 or to 100. The general form of the for loop will be as follow:

```
for variable in sequence:  # Loop Header
    #Loop Body
    statement1
    statement2
    …………
    Statement N
```

The first line of code in a loop is sometimes called the loop ***header***. The sequence could be list, tuple, set, or string which contain finite number of elements on which the loop iterates for each iteration. A sequence of elements also can be created using ***range ()*** function. The **colon (:)** ends the loop header. The ***loop body*** comprises the statements in the remaining lines of code, below the header. All the statements in the loop body must be ***indented*** and ***aligned*** in the same column. These statements are executed in sequence on each pass through the loop.



Write a python program to demonstrate calculating the exponentiation using for loop.

We need three variables to represent the number, the exponent, and the product. We can calculate the product on each pass by repeating the loop for exponent number of times.

Exp2.py	**Output**

number=int(input('Enter number:'))
exponent=int(input('Enter Exponent value:'))
product=1
for i in range(exponent):
    product=product*number
    print(product,end=" ")
	Enter number:2
	Enter Exponent value:3
	2 4 8

Count-Controlled Loops
Loops that count through a range of numbers are also called count-controlled loops. The value of the count on each pass is used in computations. When Python executes the type of for loop just discussed above, it actually counts from 0 to the value of the integer expression minus 1 placed inside range () function. On each pass through the loop, the loop variable is bound to the current value of the count.

Fact.py	**Output**

num=int(input('Enter the number:'))
product=1
for i in range(num):
    """***i is loop variable,***
    its value used inside body
    for computing product"""
    product=product*(i+1)
print(f'{num}! is {product}')
	Enter the number:4
	4! is 24

Augmented Assignment
Expressions such as ***x = x + 1 or x = x + 2*** occur so frequently in loops. The assignment symbol can be combined with the ***arithmetic*** and ***concatenation*** operators to provide augmented assignment operations. The general form is as follow:

<variable> <operator>=<expression>
Which is equal to
<variable> =<variable> <operator><expression>

Following are several examples: Num+=2; x*=5; y/=12;

Traversing the Contents of a Data Sequence

The data sequence can be list, or tuple, string or set. The loop variable is bound to the next value in the sequence. The sequence of numbers generated using the range function can be converted into list or tuple. This tuple or list can be later used in the place of the sequence.

Add.py	**Output**

l=list(range(1,11))
print(l)
s=0
for x in l: #here x is loop variable and l is list
    s=s+x
print('Sum of Elements of list is',s)
	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
	Sum of Elements of list is 55

Loops That Count Down

All of our loops until now have counted up from a lower bound to an upper bound. Once in a while, a problem calls for counting in the opposite direction, from the upper bound down to the lower bound. When the step argument is a ***negative number***, the range function generates a sequence of numbers from the first argument down to the second argument plus 1.

Countdown.py	**Output**

l=list(range(10,0,-1))
print(l)
s=0
for x in l: #here x is loop variable and l is list
    s=s+x
print('Sum of Elements of list is',s)
	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
	Sum of Elements of list is 55

Formatting Text for output

Many data-processing applications in our day to day life require output needs to be display in a tabular format. In this format, numbers and other information are aligned in columns that can be either left-justified or right-justified. A column of data is left-justified if its values are vertically aligned beginning with their leftmost characters. A column of data is right-justified if its values are vertically aligned beginning with their rightmost characters.

The total number of data characters and additional spaces for a given datum in a formatted string is called its ***field width***. The print function automatically begins printing an output datum in the first available column.

Python includes a general formatting mechanism that allows the programmer to specify ***field widths*** for different types of data. The simplest form of this operation is:

```
<format_string>%<datum>
```

This version contains a ***format string***, the ***format operator %,*** and a single ***data value*** to be formatted. The format string is represented as: ***%<field width>s***. The format string begins with % operator, and positive number for right-justification, and 's' is used for string.

Format Information	**character**
Integer	d
String	s
Float	f

The format information for a data value of type float has the form:

```
%<fieldwidth>.<precision>f
```

where ***.<precision>*** is optional.

Examples:

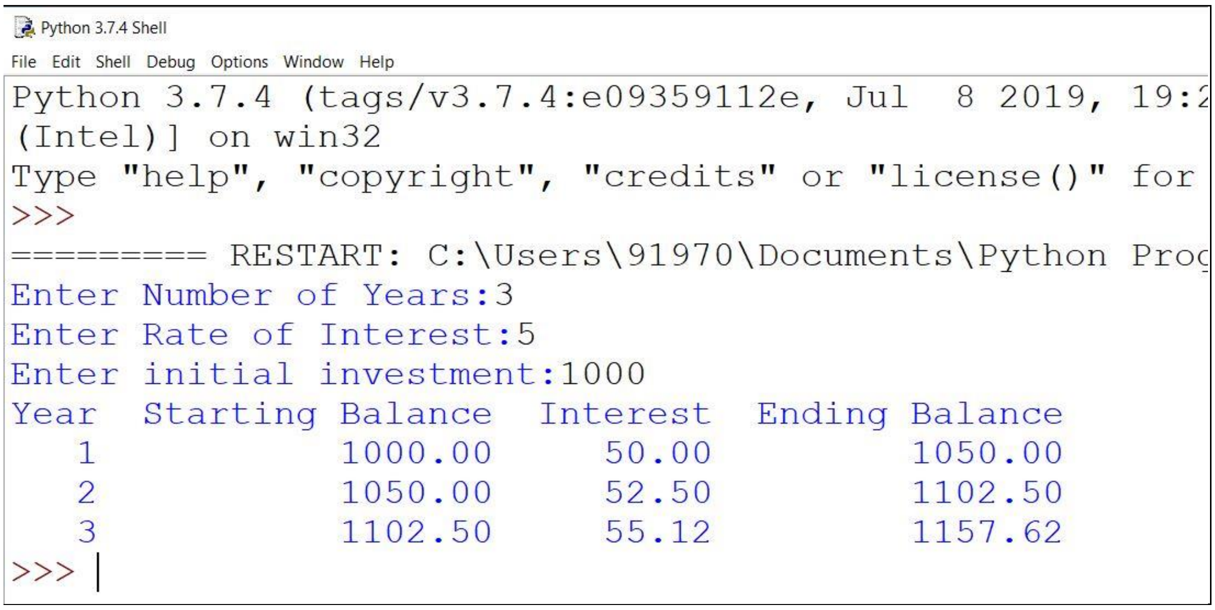
```
>>> "%8s" % "PYT"  # field width is 8, right justification, datum is string PYT.
'     PYT'
>>> "%-8s" % "PYT"  # field width is -8, left justification, datum is string PYT.
'PYT     '
>>> "%8d" % 1234   # field width is 8, right justification, datum is int 1234.
'    1234'
>>> "%-8d" % 1234  # field width is -8, left justification, datum is int 1234.
'1234    '
>>> "%6.2f" % 1234.678 # field width is 6, 2 decimal points, right justification,
datum is int 1234.678
'1234.68'
```

Write a python program to display the following table format.

Read the number of years, starting balance, Interest, and Ending balance from the keyboard.

Year	Starting balance	Interest	Ending balance
1	10000.00	500.00	10500.00
2	10500.00	525.00	11025.00
3	11025.00	551.25	11576.25

```
year=int(input('Enter Number of Years:'))
rate=float(input('Enter Rate of Interest:'))
balance=float(input('Enter initial investment:'))
print("%4s%18s%10s%16s" % ("Year", "Starting Balance", "Interest",  "Ending
Balance"))
endbal=0.0
for i in range(1,year+1):
   inter=(balance*rate)/100
   endbal=balance+inter
   print("%4d%18.2f%10.2f%16.2f" % (i,balance,inter,endbal))
   balance=endbal
```



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:2
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for
>>>
========= RESTART: C:\Users\91970\Documents\Python Proc
Enter Number of Years:3
Enter Rate of Interest:5
Enter initial investment:1000
Year  Starting Balance  Interest  Ending Balance
   1           1000.00     50.00         1050.00
   2           1050.00     52.50         1102.50
   3           1102.50     55.12         1157.62
>>> |
```

Selection: if and if-else Statements (Two-Way Selection)

Sometimes based on the condition some part of the code should be selected for execution, otherwise other part of the should be selected for execution. This kind of statements are called selection statements. If the condition is true, the computer executes the first ***alternative action*** and skips the ***second alternative***. If the condition is false, the computer skips the first alternative action and executes the second alternative.

In this section, we explore several types of selection statements, or control statements, that allow a computer to make choices.

The ***if-else*** statement is the most common type of selection statement. It is also called a ***two-way selection statement***, because it directs the computer to make a choice between two possible alternatives.

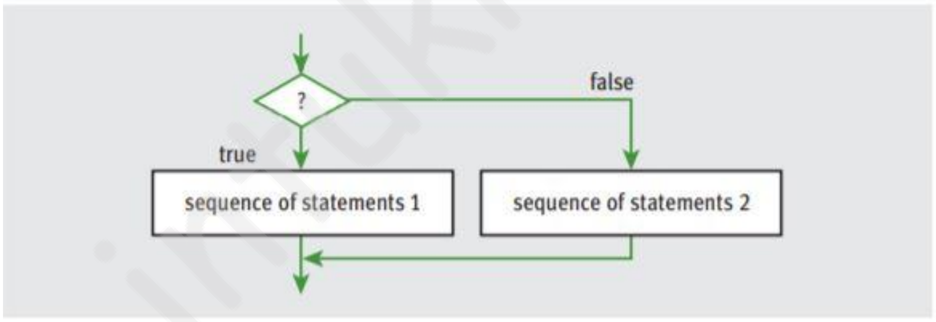
The ***if-else*** statement is often used to check inputs for errors and to respond with error messages if necessary. The alternative is to go ahead and perform the computation if the inputs are valid. For example, suppose a program takes area of a circle as input and need to compute its radius. The input must always a positive number. But, by mistake, the user could still enter a zero or a negative number. Because the program has no choice but to use this value to compute the radius, it might produce a meaningless output or error message.

```
If area<=0 then convey user to enter valid area as input
Otherwise compute the radius as sqrt(area/PI)
```

Here is the Python syntax for the if-else statement:

```
if <condition>:
        <sequence of statements-1>
else:
        <sequence of statements-2>
```

The ***condition*** in the if-else statement must be a ***Boolean expression***—that is, an expression that evaluates to either True or False. The two possible actions each consist of a sequence of statements. The Flow diagram of the if - else statement will be as follow as shown in the Figure and Dimond symbol used for representing the condition.



Write a program to find the radius from the area of the circle.

```
import math
area=float(input('Enter area:'))
if area<=0:
   print("Wrong Input!")
else:
   radius=math.sqrt(area/math.pi)
   print('The radius is:',radius)
```

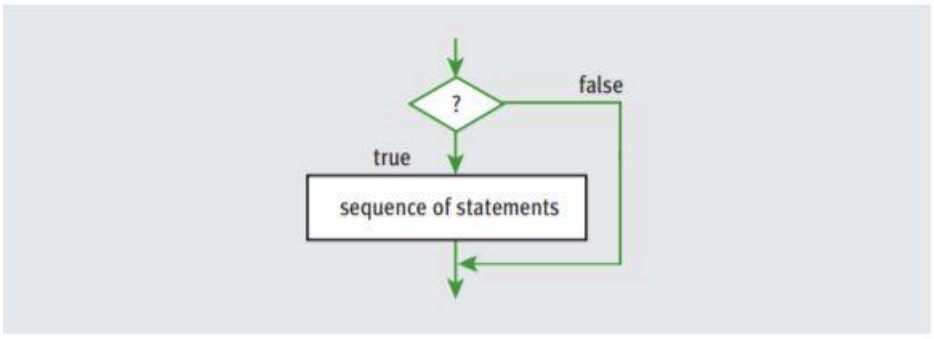
One-Way Selection Statements

The simplest form of selection is the if statement. This type of control statement is also called a ***one-way selection statement***, because it consists of a condition and just a single sequence of statements. If the condition is True, the sequence of statements will be executed. Otherwise, control proceeds to the next statement following the entire selection statement.

The Syntax will be as follow:

```
If <Boolean_Expression>:
        Sequence of statement to execute
Next ststement after if selection statement
```

Flow diagram of if

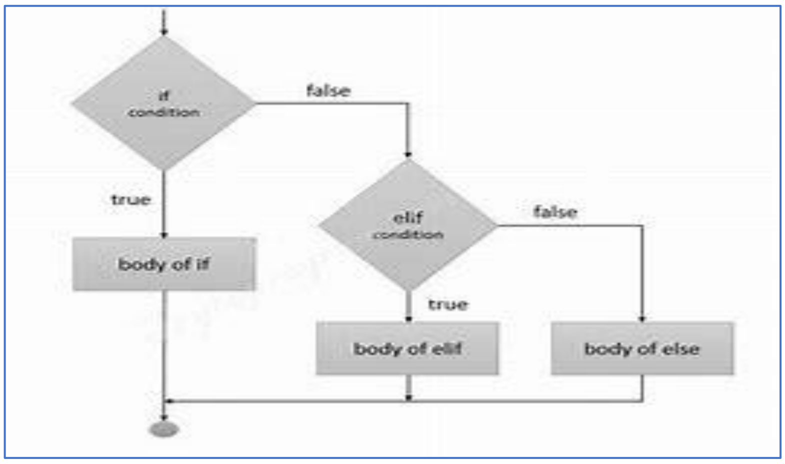


Multi-Way if Statements

The process of testing several conditions and responding accordingly can be described in code by a ***multi-way selection*** statement. This multi-way decision statement is preferred whenever we need to select one choice among multiple alternatives. The keyword '**elif**' is short for 'else if'. The else statement will be written at the end and will be executed when no if or elif blocks are executed. The syntax of will be as follow:

```
if Boolean_expression1:
        Statements
elif Boolean_expression2:
        Statements
elif Boolean_exxpression3:
        Statements
else:
        Statements
```

Flow diagram



Logical Operators and Compound Boolean Expressions

Often a course of action must be taken if either of two conditions is true. For example, valid inputs to a program often lie within a given range of values. Any input above this range should be rejected with an error message, and any input below this range should be dealt with in a similar fashion. The two conditions can be combined in a Boolean expression that uses the ***logical operator or***. The ***logical operator and*** also can be used to construct a different compound Boolean expression to express this logic.

Example Program using the logical or

```
marks=int(input('Enter your marks'))
if marks>100 or marks < 0:
    print('Marks must be within the range 0 to 100')
else:
    print('Here you can write the code to process marks')
```

Example Program using the logical and

```
marks=int(input('Enter your marks'))
if marks>0 and marks < 100:
    print('Here you can write the code to process marks')

else:
    print('Marks must be within the range 0 to 100')
```

Conditional Iteration The While Loop.

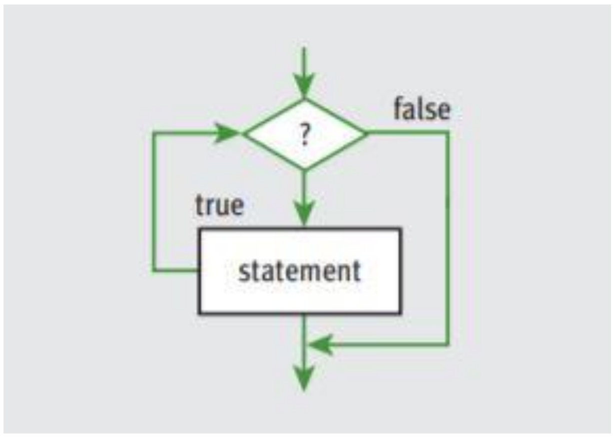
Conditional iteration requires that a condition be tested within the loop to determine whether the loop should continue. Such a condition is called the loop's ***continuation condition***. If the continuation condition is false, the loop ends. If the continuation condition is true, the statements within the loop are executed again.

The while loop is conditional iteration statement and is frequently used to execute a block of statements repeatedly until some condition remains true. The syntax will be as follow:

```
while <condition>:
        <sequence of statements>
```

Here while is the keyword, condition will be always a Boolean expression, and the loop end with **colon (:).** The first line is called loop body. The sequence of statement that we write inside the body is called loop body.

The flow diagram will be as follow:



Write a python program to find the sum of all the numbers entered from the user.

Tot.py	Output

data=input('Enter any number:')
tot=0
while data!="":
    data=int(data)
    tot=tot+data
    data=input('Enter any number:')
print('The sum of numbers is:',tot)
	Enter any number:2
	Enter any number:3
	Enter any number:4
	Enter any number:5
	Enter any number:
	The sum of numbers is: 14

Count Control with a while loop

You can also use a while loop for a ***count-controlled loops***. This loop control variable must be explicitly ***initialized*** before the loop header and

incremented in the loop body. The count variable must also be examined in the explicit continuation condition. The following are the some example for count-controlled loops.

Write a python program to find sum of all the digits of a given number?

Total.py	**Output**
num=int(input('Enter number :'))	Enter number :234
rem=0	The sum of all digits is: 9
tot=0	
while num>0:	
rem=num%10	
tot=tot+rem	
num=num//10	
print('The sum of all digits is:',tot)	

Write a python program to determine whether a given number is palindrome or not.

Palindrome.py	**Output**
num=int(input('Enter number :'))	Enter number :121
rem=0	The reversed number is: 121
x=num	121 is palindrome
rev=0	
while num>0:	Enter number :123
rem=num%10	The reversed number is: 321
rev=rev*10+rem	123 is not palindrome
num=num//10	
print('The reversed number is:',rev)	
if x==rev:	
print(x,'is palindrome')	
else:	
print(x,'is not palindrome')	

Write a python program to determine whether a given number is Armstrong number or not.

A number is an Armstrong Number if it is equal to the sum of its own digits raised to the power of the number of digits.

Armstrong.py	**Output**
d=input('Enter number :')	Enter number :153
num=int(d)	The sum of cubes is: 153
tot=0	153 is Armstrong
x=num	
rev=0	Enter number :435

```
while num>0:
    rem=num%10
    tot=tot+(rem**int(len(d)))
    num=num//10
print('The sum of cubes is:',tot)
if x==tot:
    print(x,'is Armstrong')
else:
    print(x,'is not Armstrong')
```

```
The sum of cubes is: 216
435 is not Armstrong

Enter number :1634
The sum of cubes is: 1634
1634 is Armstrong
```

Jump statements with while loop

We can use the jump statements inside the while loop

we have three jump statements: break, continue and pass.

- ✓ When **break** is used inside the while loop it terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.
- ✓ The **continue** statement skips all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- ✓ The **pass** statement does nothing, but it simply transfers control to the next statement.

Else with while loop

It is executed when the condition becomes false (with while loop), but not when the loop is terminated by a break statement.

Syntax:

```
i = 1

while i < 6:

  print(i)

  i += 1

else:

  print ("i is no longer less than 6")
```

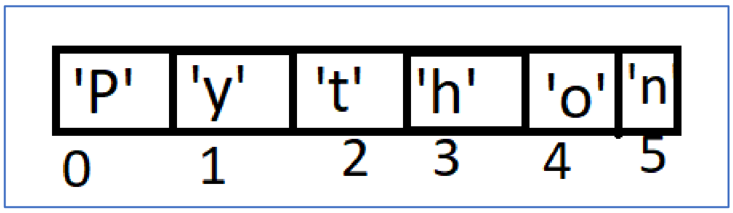
Strings and Text Files: (Part -2 of Unit 2)

Accessing Character and Substring in Strings

So far, we have learned how to use strings for input and output. We have learned how to combined two strings which is called concatenation. We also have learned how to fetch each character from the string using the for loop. In this section we will learn how to fetch the desired portions of the string called substrings.

The Structure of Strings

A string is a sequence of zero or more characters. It is treated as a data structure. A data structure is a compound unit that consists of several smaller pieces of data. When working with strings, the programmer sometimes must be aware of a string's length and the positions of the individual characters within the string. A string's length is the number of characters it contains. The length can be obtained using the **len()** function by passing the string as an argument to it. The positions of a string's characters are numbered from **0**, on the left, to the length of the string minus 1, on the right.



In the above string 'Python', the number of characters is 6, the position of the starting character is o, and last character is length of the string minus 1 (i.e 6-1).

The Subscript Operator

Though the for loop helps to fetch individual characters from the given string. Sometimes it may be need to extract the character at specified position. This can be done using the ***subscript operator***. The form of a subscript operator is the following:

<a string> [<an integer expression>]

The first part of the subscript operator is the string that you want to inspect, the integer expression in square brackets is the position of the character that we want to inspect in that string. This integer expression is also called 'Index'. Example:

S= "Python"

S[0] -> gives 'P'

S[1] -> gives 'y'

S[5] -> gives 'n'

Slicing for Substrings

The portions of the strings or parts of the strings are called "Substrings". we can use Python's subscript operator to obtain a ***substring*** through a process called ***slicing***. To extract a substring, the programmer places a colon (:) in the subscript. An integer value can appear on either side

of the colon. The form of the slicing will be as follow: **<a string> [start_position : End_position]**, here the substring is obtained from the start_position but not including the End_position.

Example:

S[0:3] -> gives a substring ‘**Pyt**’

S[2:5] -> gives the substring ‘**tho**’

Testing for a Substring with the in Operator

Sometimes we want to fetch the strings from the list of strings that contain a substring. This can be done using the membership operator ‘in’. If the substring is part of the strings in a list then we want to put all such strings into separate list or can be displayed.

Example program:

Write a python program that display all the strings that match with the given substring.

```
l=['prg1.py','sort.txt','substring.py','add.c','mul.cpp','eventest.py']
l2=[]
sub=input('Enter the substring:')
for x in l:
    if sub in x:
        l2.append(x)
        print(x)
#display final list2
print(l2,end=" ")
```

Data Encryption

The information travelling through the network is vulnerable(exposed) to the spies and potential thieves. By using the right sniffing software, a person can get the information passing between any two computers. Many applications use Data Encryption to protect the information transmitted on the network.

General scenario of security attacks

Security attack: Any action that compromises the security of information owned by an organization is called security attack.

Security attacks are classified into two: Passive and Active

Passive Attacks - Passive attacks are in the nature of eavesdropping (secretly listening private conversation) on transmissions. There are two types as follow:

- Release of message contents –unauthorized person listens to the message from sender to the receiver
- Traffic analysis - unauthorized person observes the patterns of messages.

Active Attacks

Active attacks involve some modification of data stream or the creation of a false stream. These are sub divided into four categories :

- Masquerade-one entity pretends to be a different entity
- Replay- involves the passive capture of a data until its subsequent retransmission to produce an unauthorized effect.
- modification of messages - some portion of a legitimate message is altered
- denial of service -Another form of service denial is the disruption of an entire network

Need of data encryption

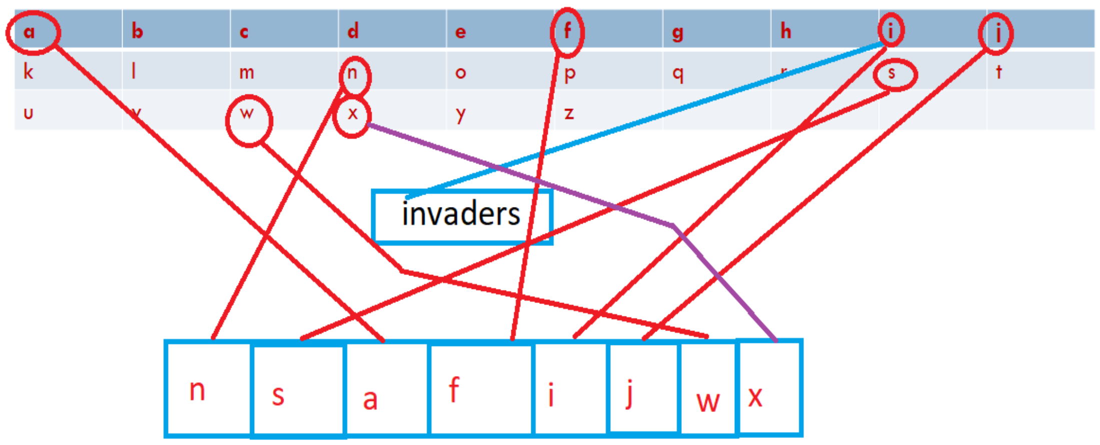
The process of converting the information in such way that it cannot be understood by the unauthorized user is called data encryption. The reverse process is called decryption. Some application protocols such as FTPS, and HTTPS are used to provide security to the information transmitted over the network.

Basics of Data Encryption

- The information that is to be transmitted is called 'Plain Text'
- The sender encrypts the message by translating it into a secret code, which is known as 'Cipher Text'
- The receiver at the other end decrypts the cipher text into original message or plain text.
- Both parties use keys to encrypt and decrypt messages which are known as secret keys.
- A very simple encryption method that has been in use for thousands of years is called a Caesar cipher.

Caesar cipher

- This encryption strategy replaces each character in the plain text with the character that occurs at given distance away in the sequence.
- For positive distances, the method wraps around to the beginning of the sequence to locate the replacement characters for those characters near its end.
- For example, if the distance value of a Caesar cipher equals five characters, the string "invaders" would be encrypted as "nsafijwx."
- Here, 'invaders' is called plain text and 'nsafijwx' is called cipher text.



Implementing Caesar cipher method for encryption

```
pt=input('Enter your text:')
dist=int(input('Enter distance:'))
ct=""
for ch in pt:
    ordvalue=ord(ch)
    ciphervalue=ordvalue+dist
    if ciphervalue>ord('z'):
        ciphervalue=ord('a')+dist-(ord('z')-ordvalue+1)
    ct=ct+chr(ciphervalue)
print(ct)
```

Implementing Caesar cipher method for decryption

```
code=input('Enter your text:')
dist=int(input('Enter distance:'))
plaintext=""
for ch in code:
    ordvalue=ord(ch)
    ciphervalue=ordvalue-dist
    if ciphervalue<ord('a'):
        ciphervalue=ord('z')-(dist-(ord('a')-ordvalue+1))
    plaintext=plaintext+chr(ciphervalue)
print(plaintext)
```

Strings and Number Systems

When you perform arithmetic operations, you use the decimal number system. This system, also called the base ten number system, uses the ten characters 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 as digits. The binary number system is used to represent all information in a digital computer. The two digits in this base two number system are 0 and 1. To identify the system being used, you attach the base as a subscript to the number. For example, the following numbers represent the quantity 415_{10}

The digits used in each system are counted from 0 to n - 1, where n is the system’s base.

- Binary system base is 2, hence it includes digits from 0 to 2-1 [0,1]
- Octal system base is 8, hence it includes digits from 0 to 8-1 [0,1,2,3,4,5,6,7]
- Decimal system base is 10, hence it includes digits from 0 to 10-1 [0,1,2,3,4,5,6,7,8,9]
- Hexadecimal system base is 16, hence it includes digits from 0 to 16-1[0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F]

The Positional System for Representing Numbers

- The value of each digit in a number is determined by the digit’s position in the number.
- In other words, each digit has a positional value.
- The positional value of a digit is determined by raising the base of the system to the power specified by the position ($base^{position}$).
- For an n-digit number, the positions (and exponents) are numbered from n - 1 down to 0, starting with the leftmost digit and moving to the right.
- the positional values of the three-digit number 415_{10} are 100 (10^2), 10 (10^1), and 1 (10^0) { where n is 3, hence positions are 3-1 to 0}
- To determine the quantity represented by a number in any system from base 2 through base 10, you multiply each digit (as a decimal number) by its positional value and add the results.
- The following example shows how this is done for a three-digit number in base 10:

$$415_{10} =$$
$$4 * 10^2 + 1 * 10^1 + 5 * 10^0 =$$
$$4 * 100 + 1 * 10 + 5 * 1 =$$
$$400 + 10 + 5 = 415$$

Converting Binary to Decimal

- Like the decimal system, the binary system also uses positional notation.
- However, each digit or bit in a binary number has a positional value that is a power of 2.
- The binary number can be referred as a string of bits or a bit string.
- To get the decimal number, multiply the value of each bit (0 or 1) by its positional value and add the results. Let’s do that for the number 1100111_2:

$$1100111_2 =$$

$$1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 =$$

$$1 * 64 + 1 * 32 + 0 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 1 * 1 =$$

$$64 + 32 + 4 + 2 + 1 = 103$$

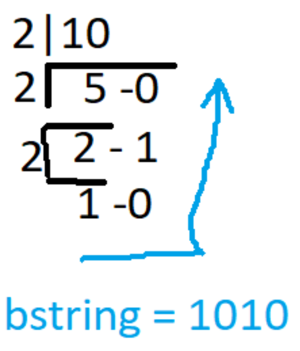
Implementing Binary number to Decimal number in Python

```
bstring=input('Enter the bit string :')
decimal=0
exponent=len(bstring)-1
for digit in bstring:
    decimal=decimal+int(digit)*2**exponent
    exponent=exponent-1
#printing the equivalnet decimal value
print('The decimal value is :',decimal)
```

```
Enter the bit string :101
The decimal value is : 5
Enter the bit string :1100111
The decimal value is : 103
```

Implementing Decimal to Binary in Python

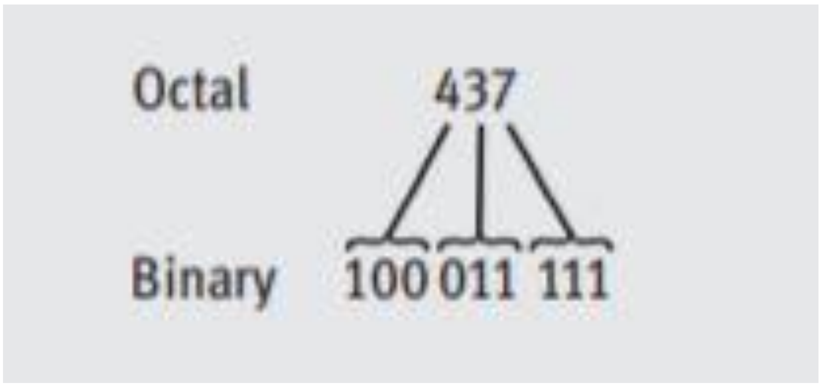
```
Enter number:10
   5     0       0
   2     1      10
   1     0     010
   0     1    1010
The binary representation is: 1010
```



```
decimal=int(input('Enter number:'))
bstring=""
if decimal ==0:
   print(0)
else:
   while decimal >0:
      rem=decimal%2
      bstring=str(rem)+bstring
      decimal=decimal//2
      print("%5d%8d%12s" % (decimal, rem, bstring))
   print('The binary representation is :',bstring)
```

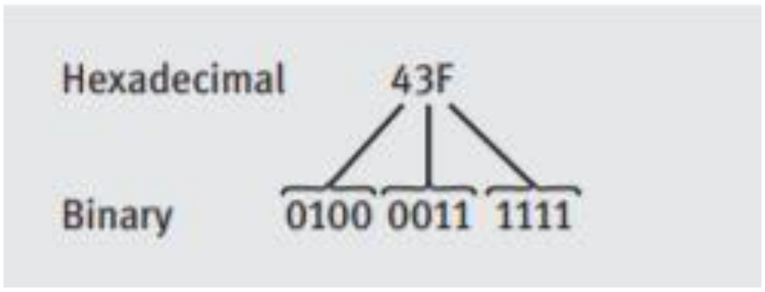
Octal and Hexadecimal Numbers

- ☐ We have function like oct() and hex() to convert the numbers into decimal to octal, and decimal to hexadecimal numbers.
- ☐ To convert from octal to binary, you start by assuming that each digit in the octal number represents three digits in the corresponding binary number.
- ☐ You then start with the leftmost octal digit and write down the corresponding binary digits, padding these to the left with 0s to the count of 3, if needed.



- ☐ Each digit in the hexadecimal number is equivalent to four digits in the binary number.

- Thus, to convert from hexadecimal to binary, you replace each hexadecimal digit with the corresponding 4-bit binary number.
- To convert from binary to hexadecimal, you factor the bits into groups of four and look up the corresponding hex digits.



String Methods

Text processing involves many different operations on strings. Python includes a set of string operations called methods. A method behaves like a function, but has a slightly different syntax. Unlike a function, a Method is always called with a given data value called an object, which is placed before the method name in the call. The syntax of a method call is the following:

<object>.method_name(**arg1,arg2,....argn**)

Example:

```
txt="apple is good for health than mango, one apple a day keeps a doctor away".

print(txt.count('apple'))
```

Methods can also expect arguments and return values. A method knows about the internal state of the object with which it is called. The methods are as useful as functions, but we need to use the dot notation which we have already discussed with module concept. Every data type includes a set of methods to use with objects of that type.

We will see some useful methods of string object.

Method	**Description**	**Example**

s.center(width)	Returns a copy of **s** centered within the given number of spaces.	s='apple'
		
		s.center(20)
		
		' apple '

s.count(substring, start,end)	Returns the number of non-overlapping occurrences of substring in s. start and end are used to verify in a slice	txt="apple is good for health than mango, one apple a day keeps a doctor away" txt.count('apple')

s.endswith(substring)	Returns True if s ends with substring or False
	
	otherwise
		s='program1.py'
		
		s.endswith('.py')
		
		True

s.find(substring)	Returns the index of first occurrence of substring if found, otherwise return -1	s='My kids like apple than orange' s.find('apple') 13 >>> s.find('papaya') -1

s.isalpha()	Returns True if s contains only letters or
	False otherwise.
		s='My kids like apple than orange'
		>>> s.isalpha()
		False
		>>> s="apples"
		>>> s.isalpha()
		True

s.isdigit()	Returns True if s contains only digits or
	False otherwise
		>>> s='123456'
		>>> s.isdigit()
		True

s.join(sequence)	Takes all the items in the sequence, and joins them into one string using the separator as string.	>>> l=['twitter','facebook','whatsapp'] >>> '#'.join(l) 'twitter#facebook#whatsapp'

s.lower()	Returns a copy of s converted to lowercase	>>> s='APPLE'
		>>> s.lower()
		'apple'

s.replace(old, new)	Returns a copy of s with all occurrences of substring old replaced by new.	>>> s='i like ice cream. it is very tasty'
		>>> s.replace('i','I')
		'I lIke Ice cream. It Is very tasty'

s.split([sep])	Splits the string into list of words using the separator. The default separator is space. We can use comma, or any other character as separator.	>>> s='hello, i am KSR, i teach python programming'
		>>> s.split(',')
		['hello', ' i am KSR', ' i teach python programming']

s.startswith(substring)	Returns True if the string starts with the specified substring, otherwise False	>>> s='Python is powerful language among all oop'
		>>> s.startswith('Pyt')
		True
		>>> s.startswith('pyt')
		False

s.strip([aString])	Removes the white spaces in the beginning and ending of a string	>>> s=' APPLE '
		>>> s
		' APPLE '
		>>> s.strip()
		'APPLE'

s.upper()	Returns a copy of s converted to uppercase.	>>> s='apple'
		>>> s.upper()
		'APPLE'

s.isalnum()	Returns if the string contains alphanumeric characters, otherwise False	>>> s='apple'
		>>> s.isalnum()
		True

s.isalpha()	Returns True if the string contains only alphabets otherwise False.	>>> s='Python Programming'
		>>> s.isalpha()
		False

s.title()	Converts the given string into title format, where first	>>> s='python programming'
		>>> s.title()
		'Python Programming'

	character of each word is capitalized	

Some other methods of string class

Lstrip() - Used to remove leading spaces

Rstrip() - Used to remove spaces after the string or trailing spaces

Istitle() -used to test whether it is a title or not. If it is title returns True, otherwise returns False.

Casefold() - The casefold() method is an aggressive lower() method which converts strings to case folded strings for caseless matching.

Format() –Format Type

:<, left assign

:> , Right assign

:, Inserts comma as thousand separator

:b, converts string into binary

:o, converts string into octal

:x, converts string into hexadecimal

Example:

'{:,}'.format(12345678)

Output:

12,345,678

Isidentifier() -used to test whether it is an identifier.

Maketrans('s1','s2') – Creates a translate table.

Translate(table) - This translate the string using the translate table.

Partition(substring) -The partition() method searches for a specified string, and splits the string into a tuple containing three elements.

Text Files

Using a text editor such as Notepad or TextEdit, you can create, view, and save data in a text file. The data in a text file can be viewed as characters, words, numbers, or lines of text. When the data are treated as numbers (either integers or floating-points), they must be separated by whitespace characters—spaces, tabs, and newlines. For example, a text file containing six floating-point numbers might look like:

```
34.6  22.33     66.75     77.12     21.44     99.01
```

All data output to or input from a text file must be strings. Thus, numbers must be converted to strings before output, and these strings must be converted back to numbers after input.

Writing Text to a File

- Data can be output to a text file using a file object.
- Python's open function, which expects a file pathname and a mode string as arguments, opens a connection to the file on disk and returns a file object.
- The mode string is **'r'** for input files and **'w'** for output files.
- Thus, the following code opens a file object on a file named myfile.txt for output:

  f=open("myfile.txt",'w')

- If the file does not exist, it is created with the given pathname.
- If the file already exists, Python opens it. When data are written to the file and the file is closed, any the data that is already existing in the file will be erased.
- String data are written (or output) to a file using the method write method with the file object. The write method expects a single string argument.
- If you want the output text to end with a newline, you must include the escape character \n in the string.

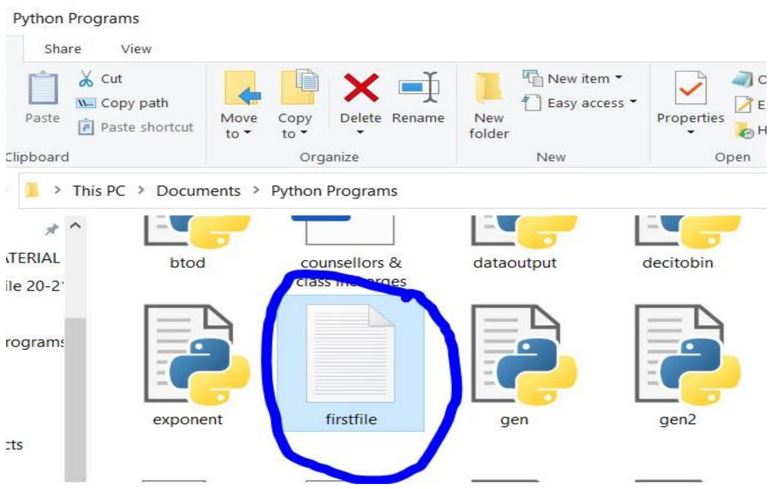
- **f.write("First line.\nSecond line.\n")**

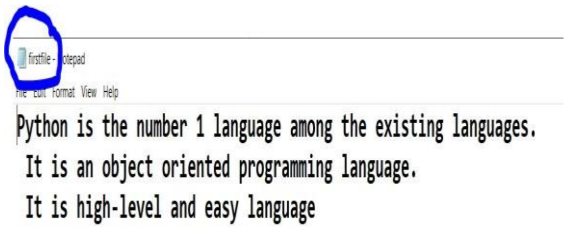
- When all of the outputs are finished, the file should be closed using the method close, as follows:

- **f.close()**

```
f=open('firstfile.txt','w')
f.write('Python is the number 1 language among the existing languages.\
     \n It is an object oriented programming language.\
     \n It is high-level and easy language\n')
f.close()
```

Output





```
Python is the number 1 language among the existing languages.
 It is an object oriented programming language.
 It is high-level and easy language
```

Writing Numbers to a File

- ☐ The file method write expects a string as an argument. Therefore, other types of data, such as integers or floating-point numbers, must be converted to strings before being written to an output file.
- ☐ These data types can be easily converted into string using the str() function.
- ☐ The resulting strings are then written to a file with a space or a newline as a separator character.
- ☐ We can generate random numbers and put them in a file using the write() function by converting them to string using str() function.

Source Code

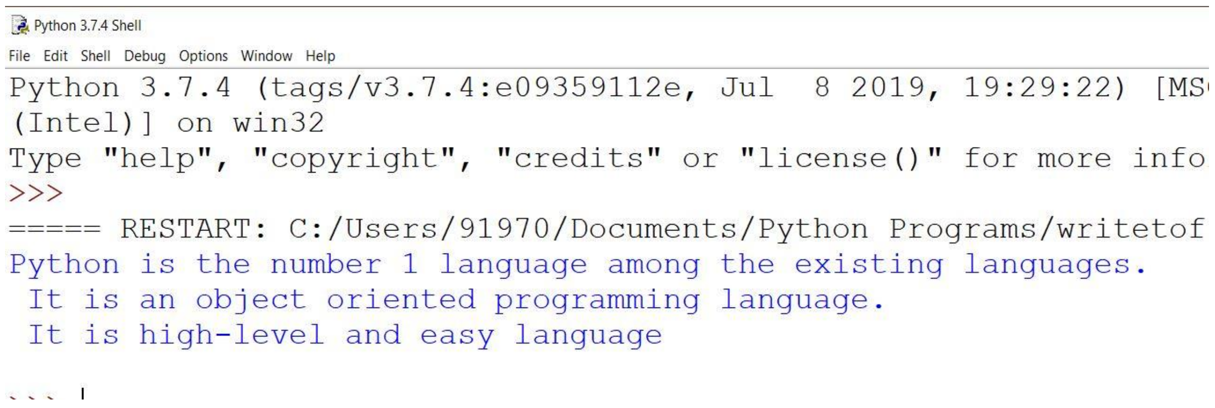
```
import random
f=open('nums.txt','w')
for n in range(20):
    number=random.randint(1,100)
    f.write(str(number)+"\n")
f.close()
```

Reading Text from a File

- You open a file for input in a manner similar to opening a file for output.
- The only thing that changes is the mode string, which, in the case of opening a file for input, is 'r'.
- However, if the pathname is not accessible from the current working directory, Python raises an error.
- Here is the code for opening myfile.txt for input:
- F=open('myfile.txt', 'r')
- There are several ways to read data from an input file.
    - We can use **read()** function which reads the entire content of a file as single string.
    - We can use **for()** loop which considers the file object as a lines of text.
    - We can use the **readline()** method of file object which is used to read specific line from the file (say first line).

```
#using the read() function

f=open('firstfile.txt','r')

s=f.read()

print(s)
```

```
#using the for() loop

f=open('firstfile.txt','r')

for line in f:

   print(line, end="")
```



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MS
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more info
>>>
===== RESTART: C:/Users/91970/Documents/Python Programs/writetof
Python is the number 1 language among the existing languages.
 It is an object oriented programming language.
 It is high-level and easy language
```

using the readline() method to read a line

```
#using the readline() method to read a line
of file object
f=open('firstfile.txt','r')
print(f.readline())
```

using the readline() method to read all the lines

```
#reading all the lines
f=open('firstfile.txt','r')
while True:
   line =f.readline()
   if line=="":
      break
```

Reading Numbers from a File

- All of the file input operations return data to the program as strings.
- If these strings represent other types of data, such as integers or floating-point numbers, the programmer must convert them to the appropriate types before processing them further.
- Each line in the file may contain spaces. These spaces can be removed using the strip() method of the string class.

Example Program to add all the integers in a text file

```
f=open('nums.txt','r')
sum=0
for line in f:
   n=line.strip()
   sum=sum+int(n)
print('The sum of all the numbers is:',sum)
```

Accessing and Manipulating Files and Directories on Disk

- ☐ Sometimes it is better to test the current working directory for a file whether it is existing or not.
- ☐ We can know the current working directory
- ☐ We can list all the files in the current working directory
- ☐ We can create new directory

os module functions

os module function	**Description**
chdir(path)	Changes the current working directory to path
getcwd()	Returns the path of the current working directory.
listdir(path)	Returns a list of the names in directory named path.
mkdir(path)	Creates a new directory named path and places it in the current working directory.
remove(path)	Removes the file named path from the current working directory.
rename(old, new)	Renames the file or directory named old to new

os.path functions

os.path functions	**Description**
exists(path)	Returns True if path exists and False otherwise.
isdir(path)	Returns True if path names a directory and False otherwise.
isfile(path)	Returns True if path names a file and False otherwise.
getsize(path)	Returns the size of the object names by path in bytes.

Example:

- os.path.exists(‘nums.txt’) → True
- Os.path.isdir(os.getcwd()) → True
- os.path.isfile('nums.txt’) → True
- os.path.getsize(os.getcwd()) → 12288 bytes

*********End of Unit 2*********