UNIT-2:

Control Statement: Definite iteration for Loop Formatting Text for output, Selection if and if elseStatement Conditional Iteration The While Loop

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

Data Encryption:

1. Data encryption to protect information transmitted on networks. Some application protocols have been updated to include secure versions that use data encryption. Examples of such versions are FTPS and HTTPS, which are secure versions of FTP and HTTP for file transfer and Web page transfer, respectively.

2. Encryption techniques are as old as the practice of sending and receiving messages. The senderencrypts a message by translating it to a secret code, called a cipher text.

3. At the other end, the receiver decrypts the cipher text back to its original plaintext form.

4. Both parties to this transaction must have at their disposal one or more keys that allow themto encrypt and decrypt messages.

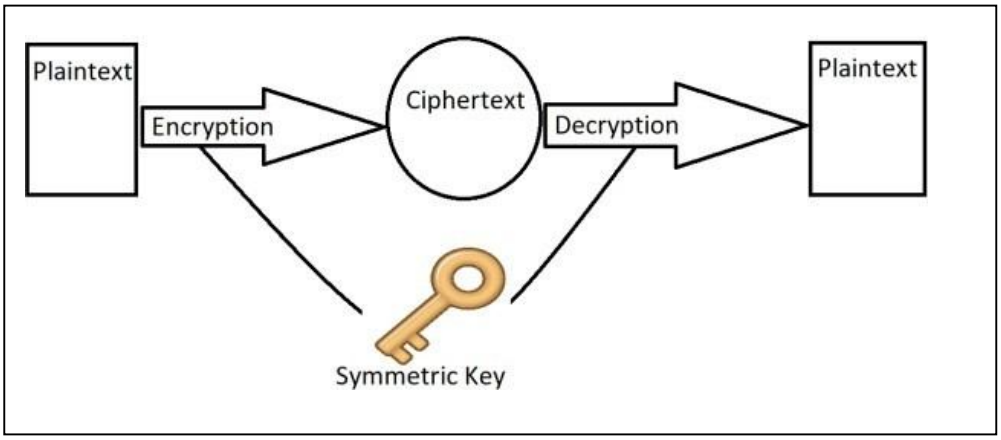


Fig : Encryption and Decryption Using Symmetric Key

5. A very simple encryption method that has been in use for thousands of years is called aCaesar cipher.
6. The next two Python scripts implement Caesar cipher methods for any strings

thatcontain lowercase letters and for any distance values between 0 and 26.

7. Here we are using the **ord** function returns the ordinal position of a character value inthe ASCII sequence, whereas **chr** is the inverse function.

Example-1:

```
“””
File:encrypt.py
Encrypts an input string of lowercase letters and
prints the result.The other input is the distance
value.
“””
```

Program:

```
plain_text=input('Enter Plain Text:')
key=int(input('Enter KEY Value:'))
Cipher_Text=""
 for i in plain_text:
      ordvalue=ord(i)
     ciphervalue=ordvalue+key
     Cipher_Text+=chr(ciphervalue)
print(Cipher_Text)
```

Output:

```
Enter Plain Text:
hello!(#Enter KEY
Value: 2 jgnnq#*%
```

Example-2:

```
“””
File: decrypt.py
Decrypts an input string of lowercase letters and
prints the result.The other input is the distance
value.
“””
```

Program:

```
ciphertext=input("enter cipher text: ")
key=int(input("enter key value: "))
plain=" "
for i in ciphertext:
      ordvalue=ord(i)
      plainvalue=ordvalue-key
     plain+=chr(plainvalue)
print(plain)
```

Output:

```
enter cipher text: jgnnq#*%
enter key value: 2
 hello!(#
```

Strings in Python

A single character or a sequence of characters identify as strings in Python. Strings are one of the primitive data structures in Python. A very important thing while working with strings is that strings are “immutable”.
That is, once we create a string, we cannot change its value. So to manipulate strings, we’ll need to create new strings as we go, while the original strings stay intact. We always enclose a string within **quotes**, either **single-quotes** or **double-quotes**.

Python String Indexing

Recall that a string is just a **sequence** of characters. And like any other sequence in Python, strings support **indexing** too. That is, we can access individual characters of a string using a **numeric index**.

Python starts counting from **zero** when it comes to numbering **index locations** of a sequence. So the first character of a string is at index **0**, the second character at index **1** and so on. We also have **negative indices** which count from **right to left**, starting with **-1**.
Indexing uses **square bracket notation**. So this is how we access a character at index of a string **my_string**:

```
>>> say = "Hi from Techvidvan!"
>>> say[0]
'H'
>>> say[3]
'f'
>>> say[5]
'o'
>>> say[-1]
'!'
>>> say[-3]
'a'
>>> say[-5]
'd'
```

Python String Slicing

We can extend the **square bracket notation** to access a **substring (slice)** of a string. We can specify **start**, **stop** and **step (optional)** within the **square brackets** as:

```
my_string[start:stop:step]
```

- **start:** It is the index from where the slice starts. The default value is **0**.
- **stop:** It is the index at which the slice stops. The character at this index is not included in the slice. The default value is the **length of the string**.
- **step:** It specifies the number of jumps to take while going from start to stop. It takes the default value of **1**.

```
>>> say = "Hi from Techvidvan!"
>>> say[3:6]
'fro'
>>> say[:10]
'Hi from Te'
>>> say[4:]
'rom Techvidvan!'
>>> say[1:11:3]
'ir c'
>>> say[::2]
'H rmTcvda!'
>>> say[::-1]
'!navdivhceT morf iH'
>>>
```

Python String Operations

1. Python + operator – String Concatenation operator

The **+ operator joins** all the operand strings and returns a **concatenated string**.

For Example

```
>>> x = "This is "
>>> y = "Python"
>>> z = x + y + '.'
>>> print(z)
```

Output

This is Python.

2. Python * operator – String Replication operator

The *** operator** takes **two** operands – a **string** and an **integer**. It then returns a **new string** which is a number of **repetitions** of the input string. The integer operand **specifies** the number of repetitions.

For Example

```
>>> x = 3 * "Hi!"
```

```
>>> print(x)
```
Output
Hi!Hi!Hi!

String Methods in Python

Everything in Python is an **object**. A **string** is an object too. Python provides us with various methods to **call on** the string object.
Let's look at each one of them.

Note that none of these methods **alters** the actual string. They instead **return a copy** of the string.This copy is the **manipulated version** of the string.

1. s.capitalize() in Python

Capitalizes a string

```
>>> s = "heLLo BuDdY"
>>> s2 = s.capitalize()
>>> print(s2)
```

Output
Hello buddy

2. s.lower() in Python

Converts all alphabetic characters to **lowercase**

```
>>> s = "heLLo BuDdY"
>>> s2 = s.lower()
>>> print(s2)
```
Output
hello buddy

3. s.upper() in Python

Converts all alphabetic characters to **uppercase**

```
>>> s = "heLLo BuDdY"
>>> s2 = s.upper()
>>> print(s2)
```
Output
HELLO BUDDY

4. s.title() in Python

Converts the **first letter** of each word to **uppercase** and remaining letters to lowercase

```
>>> s = "heLLo BuDdY"
>>> s2 = s.title()
>>> print(s2)
```

Output

Hello Buddy

5. s.swapcase() in Python

Swaps case of all alphabetic characters.

```
>>> s = "heLLo BuDdY"
>>> s2 = s.swapcase()
>>> print(s2)
```

Output

‘HEllO bUdDy’

6. s.count(<sub>[, <start>[, <end>]])

Returns the **number of time** <sub> occurs in s.

For Example

```
>>> s = 'Dread Thread Spread'
>>> s.count('e')
3
>>> s.count('rea')
3
>>> s.count('e', 4, 18)
2
>>>
```

7. s.find(<sub>[, <start>[, <end>]])

Returns the index of the **first** occurrence of <sub> in s. Returns -1 if the substring is not present in the string.

For Example

```
>>> s = 'Dread Thread Spread'
>>> s.find('Thr')
6
>>> s.find('rea')
1
>>> s.find('rea', 4, 18)
```

```
8
>>> s.find('x')
-1
>>>
```

8. s.isalnum() in Python

Returns True if all characters of the string are **alphanumeric**, i.e., **letters** or **numbers**.
For Example

```
>>> s = "Tech50"
>>> s.isalnum()
True
>>> s = "Tech"
>>> s.isalnum()
True
>>> s = "678"
>>> s.isalnum()
True
```

9. Python s.isalpha()

Returns True if all the characters in the string are **alphabets**.
For Example

```
>>> s = "Techvidvan"
>>> s2 = "Tech50"
>>> s.isalpha()
True
>>> s2.isalpha()
False
>>>
```

10. Python s.isdigit()

Returns True if all the characters in the string are **numbers**.
For Example

```
>>> s1 = "Tech50"
>>> s2 = "56748"
>>> s1.isdigit()
False
>>> s2.isdigit()
True
```

```
>>>
```

11. s.islower()

Returns True if all the alphabets in the string are **lowercase**.
For Example

```
>>> s1 = "Tech"
>>> s2 = "tech"
>>> s1.islower()
False
>>> s2.islower()
True
```

12. s.isupper()

Returns True if all the alphabets in the string are **uppercase**.
For Example

```
>>> s1 = "Tech"
>>> s2 = "TECH"
>>> s1.isupper()
False
>>> s2.isupper()
True
```

13. s.lstrip([<chars>])

Removes **leading characters** from a string. Removes **leading whitespaces** if you don't provide a **<chars> argument**.

14. s.rstrip([<chars>])

Removes **trailing characters** from a string. Removes **trailing whitespaces** if you don't provide a **<chars> argument**.

15. s.strip([<chars>])

Removes **leading and trailing characters** from a string. Removes **leading and trailing whitespaces** if you don't provide a **<chars> argument**.
Example of lstrip(), rstrip() and strip():

```
>>> s = " Techvidvan "
>>> s.lstrip()
'Techvidvan '
```

```
>>> s.rstrip()
' Techvidvan'
>>> s.strip()
'Techvidvan'
>>>
```

16. s.join(<iterable>)

Joins elements of an **iterable** into a **single string**. Here, **s** is the **separator** string.

Example

```
>>> s1 = ' '.join(['We', 'are', 'Coders'])
>>> print(s1)
We are Coders
>>> s2 = ':'.join(['We', 'are', 'Coders'])
>>> print(s2)
We:are:Coders
>>>
```

17. s.split(sep = None, maxspit = -1)

Splits a string into a list of **substrings** based on **sep**. If you don't pass a value to sep, it splits based on **whitespaces**.

Example

```
>>> l = 'we are coders'.split()
>>> l
['we', 'are', 'coders']
>>> l = 'we are coders'.split('e')
>>> l
['w', ' ar', ' cod', 'rs']
```

String Padding:

Inserting non-informative characters to a string is known as Padding. The insertion can be done anywhere on the string. String padding is useful for output formatting and alignment of strings while printing. Even for displaying values like a table string Padding is required.

Different Types of Padding

The three most commonly used String Padding techniques are :

- Inserting a character to the end of the String to make it of a specified length is known as **Left Padding.** It is mostly helpful when naming files that mostly start with numeric characters which are generated in sequence.
- In the case of **Center Padding**, the required character is inserted at both the ends of the strings with equal number of times until the newly formed string is of the specific length. This centers the string of the specific or required length. This returns a string centered of length width. Here, the default character is space.
- In **Right Padding**, the given character is inserted to the right side of the String which needs to be padded. The specified character is added to the end of the string until it attains the required length.

➢ ljust()

Using this method, the string is aligned to the left side by inserting padding to the right end of the string.

Syntax : string.ljust(width, char)
Parameters:
- **Width** -It is the length of the string after the padding is complete. Compulsory.
- **Char**– string to be padded, optional.

Returns: Returns the string which is remaining after the length of the specified width.

Example

```
# Creating a string variable
str = "Geeks for Geeks"

# Printing the output of ljust() method
print(str.ljust(20, '0'))
```

Output:
Geeks for Geeks00000

➢ rjust()

Using this method, the String is aligned to right by insertion of characters to the left end of the String.

Syntax: string.rjust(width, char)
Parameters:
- **Width** -It is the length of the string after the padding is complete. Compulsory.
- **Char**– string to be padded, optional.

Returns: returns a String which is placed right side with characters inserted to the beginning end.

Example:

```
# Creating a string variable
str = "Geeks for Geeks"

# Using rjust() method
print(str.rjust(20, "O"))
```

Output:

```
OOOOOGeeks for Geeks
```

center()

The **center()** method takes the string and aligns it in the center with the given width as parameter.
Syntax: string.center(width)

Parameter:
This method just takes one parameter which is the width and it adds white spaces to both the ends.

Returns:
Returns the string padded to the center.

Example:

```
# Creating a string variable
str = "Geeks for Geeks"

# Using center() method
print("Center String: ", str.center(20))
```

- **zfill()**

The **zfill()** method inserts 0s to the beginning of the String. This method does padding to the right by inserting characters to the left of the String with 0s until the string is of a specific length. In case of when the parameter passed is less than the length or size of the string, then no padding takes place.
Syntax: string.zfill(length)
Parameter: takes a length as its only parameter.

Example:

```
v1 = "Geeks"
v2 = "for"
v3 = "Geeks"

# Using zfill() method
print(v1.zfill(10))
print(v2.zfill(10))
print(v3.zfill(10))
```

Output:

```
00000Geeks

0000000for

00000Geeks
```

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.Because binary numbers canbe long strings of 0s and 1s.
- Computer scientists often use other number systems, such as octal (base eight) andhexadecimal (base 16) as shorthand for these numbers.

Example:

415 in binary notation 1100111112

415 in octal notation 6378

415 in decimal notation 41510

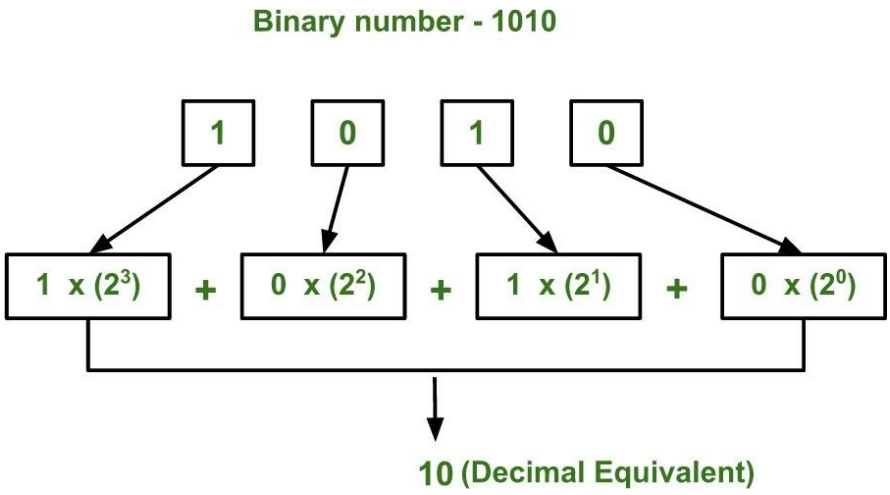
415 in hexadecimal

notation 19F16

Converting Binary to

Decimal

A binary number as a string of bits or a bit string. You determine the integer quantity that a string of bits represents in the usual manner: multiply the value of each bit (0 or 1) by its positional value and add the results.



Python Program to Converts a string of bits to a decimal integer.

```
bstring = input("Enter a string of bits: ")
decimal = 0
exponent = len(bstring) - 1
for i in bstring:
    decimal = decimal + int(i) * 2 ** exponent
    exponent = exponent - 1
print("The integer value is", decimal)
```

output:

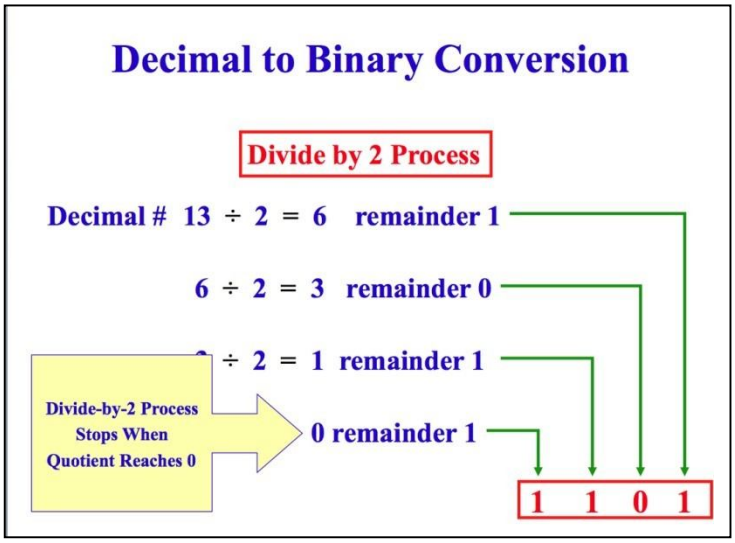
```
Enter a string of bits:
0101The integer value is 5
```

Converting Decimal to Binary

- This algorithm repeatedly divides the decimal number by 2. After each division, theremainder (either a 0 or a 1) is placed at the beginning of a string of bits.

- The quotient becomes the next dividend in the process. The string of bits is

initiallyempty, and the process continues while the decimal number is greater than 0.



Python Program to Converts a decimal integer into binary .

```
decimal = int(input("Enter a decimal integer: "))

if decimal == 0:

   print(0)

else:
     print("Quotient Remainder Binary")
    bstring = ""
    while decimal > 0:
            remainder = decimal % 2
            decimal = decimal // 2
            bstring = str(remainder) + bstring
            print("%5d%8d%12s" % (decimal, remainder, bstring))
print("The binary representation is", bstring)
```

output:

```
Enter a decimal integer: 10
Quotient Remainder Binary
```

5 0 0
2 1 10
1 0 010
0 1 1010
The binary representation is 1010

Text Files and Their Format

- Using a text editor such as Notepad or TextEdit, you can create, view, and save data in atext file. Your Python programs can output data to a text file.
- The data in a text file can be viewed as characters, words, numbers, or lines of text,depending on the text file's format and on the purposes for which the data are used.
- Python supports file handling and allows users to handle files i.e., to read and writefiles, along with many other file handling options, to operate on files.
- Python treats file differently as text or binary and this is important.
- Each line of code includes a sequence of characters and they form text file.
- Each line of a file is terminated with a special character, called the EOL or End of Linecharacters like comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun.
- The key function for working with files in Python is the open() function.
- The open() function takes two parameters; *filename*, and *mode*.There are four different methods (modes) for opening a file:

1. "r" - Read - Default value. Opens a file for reading, error if the file does not exist
2. "a" - Append - Opens a file for appending, creates the file if it does not exist
3. "w" - Write - Opens a file for writing, creates the file if it does not exist
4. "x" - Create - Creates the specified file, returns an error if the file exists

- In addition you can specify if the file should be handled as binary or text mode

1. "t" - Text - Default value. Text mode
2. "b" - Binary - Binary mode (e.g. images)

Python File Open:

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content ofthe file.

Syntax: f = open("filename",”mode”)
Example: f = open("demofile.txt", "rt")Here ,

"r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error. To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
print(f.read())
```

- If the file is located in a different location, you will have to specify the file path, like this:

Example:
```
f= open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Read Only Parts of the File: By default the read() method returns the whole text, but you can also specify how many characters you want to return. for example Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

- **Close Files:** It is a good practice to always close the file when you are done with it.

Example: Close the file when you are finish with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to afile may not show until you close the file.

- **Python File Write:** To write to an existing file, you must add a parameter to the open()function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Example:1

```
f = open("demofile.txt", "a")
# Write() =Writes the specified string into the file.
        f.write("Now the file has more content!")
        f.close()

#open and read the file after appending:
        f = open("demofile.txt", "r")
        print(f.read())
```

Example:2

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
f = open("demofile3.txt", "r")
print(f.read())
```

Note: The **"w"** mode will overwrite the entire file.

Create a New File: To create a new file in Python, use the open() method, with

"x" - Create - will create a file, returns an error if the file exist

Example: f = open("myfile.txt", "x")

Result: a new empty file is created!

- **Python Delete File:** To delete a file, you must import the OS module, and run its os.remove() function:

Example: Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Delete Folder: To delete an entire folder, use the os.rmdir() method

Example: Remove the folder "myfolder"

```
import os
os.rmdir("myfolder")
```

readline() Method:

- The readline() method returns one line from the file. You can also specified how

many bytesfrom the line to return, by using the size parameter.

Syntax: *file*.readline(*size*)

Here, size

Optional. The number of bytes from the line to return. Default -1, which means the whole line.

Example: 1

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Example: 2 Return only the five first bytes from the first line:

```
f = open("demofile.txt", "r")
print(f.readline(5))
```

readlines() Method:

- The readlines() method returns a list containing each line in the file as a list item.
- Use the hint parameter to limit the number of lines returned. If the total number of bytesreturned exceeds the specified number, no more lines are returned.

Syntax: file.readlines(hint)

Here,

Hint Optional. If the number of bytes returned exceeds the hint number, no more lines will bereturned. Default value is -1, which means all lines will be returned.

Example:

```
f = open("demofile.txt", "r")
print(f.readlines())
```

output:['Hello! Welcome to demofile.txt\n', 'This file is for testing purposes.\n', 'Good Luck!']

seek() **Method:**

- The seek() method sets the current file position in a file stream. It also returns the new postion.

Syntax: file.seek(offset)

Here, offset required. A number representing the position to set the current filestream position.

Example:

```
f = open("demofile.txt", "r")f.seek(4)
print(f.readline())
```

Output: o! Welcome to demofile.txt

tell() Method:

- The tell() method returns the current file position in a file stream.

Syntax: file.tell()

Example:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.tell())
```

Output: Hello! Welcome to demofile.txt
32

writelines() Method:

- The writelines() method writes the items of a list to the file.
- Where the texts will be inserted depends on the file mode and stream position.
- "a": The texts will be inserted at the current file stream position, default at the end of the file.
- "w": The file will be emptied before the texts will be inserted at the current file streamposition, default 0.

Syntax: file.writelines(list)

Here, list

The list of texts or byte objects that will be inserted.

Example:

```
f = open("demofile3.txt", "a")
f.writelines(["\nSee you soon!", "\nOver and out."])
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Output:

Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!
See you soon!
Over and out.

➢ Reading Numbers from a File

- The file input operation 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 manipulating them further.
- Python, the string representations of integers and floating-point numbers can be converted tothe numbers themselves by using the functions int and float, respectively.
- During input, these data can be read with a simple **for loop**. This loop accesses a line of text oneach pass. To convert this line to the integer.
- The programmer runs the **string method strip** to remove the newline and then runs the intfunction to obtain the integer value.

Example:

```
f=open("numbers.txt",'r') sum=0
for line in f:
    wordlist=line.split()
    for word in wordlist:
            number=int(word)
             sum+=number
print("The sum is",sum)
```

Output:

The sum is 55