UNIT-2:

Control Statement: 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.

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 sender encrypts 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 them to encrypt and decrypt messages.

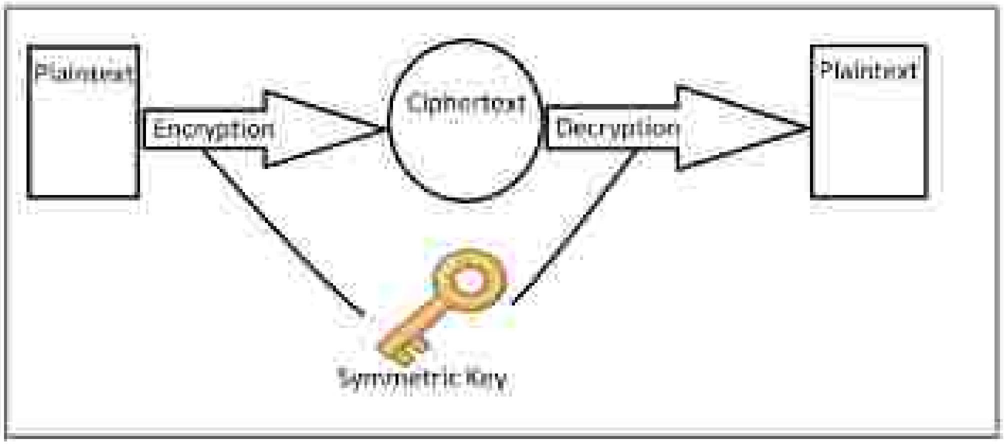


Fig.: Encryption and Decryption Using Symmetric Key

5. A very simple encryption method that has been in use for thousands of years is called a Caesar cipher.

6. The next two Python scripts implement Caesar cipher methods for any strings that contain 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 in the 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  alue:  ))
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!(#
```

String Methods

- Python has a set of built-in methods that you can use on strings.
- **Note:** All string methods returns new values. They do not change the original string.

S.No	Method	Description
1	capitalize()	Converts the first character to upper case
2	casefold()	Converts string into lower case
3	center()	Returns a centered string
4	count()	Returns the number of times a specified value occurs in a string
5	encode()	Returns an encoded version of the string
6	endswith()	Returns true if the string ends with the specified value
7	expandtabs()	Sets the tab size of the string
8	find()	Searches the string for a specified value and returns the position of where it was found
9	index()	Searches the string for a specified value and returns the position of where it was found
10	isalnum()	Returns True if all characters in the string are alphanumeric
11	isalpha()	Returns True if all characters in the string are in the alphabet
12	isdecimal()	Returns True if all characters in the string are decimals
13	isdigit()	Returns True if all characters in the string are digits
14	isidentifier()	Returns True if the string is an identifier
15	islower()	Returns True if all characters in the string are lower case
16	isnumeric()	Returns True if all characters in the string are numeric
17	isprintable()	Returns True if all characters in the string are printable

18	isspace()	Returns True if all characters in the string are whitespaces
19	istitle()	Returns True if the string follows the rules of a title
20	isupper()	Returns True if all characters in the string are upper case
21	join()	Joins the elements of an iterable to the end of the string
22	ljust()	Returns a left justified version of the string
23	lower()	Converts a string into lower case
24	replace()	Returns a string where a specified value is replaced with a specified value
25	split()	Splits the string at the specified seperator, and returns a list
26	startswith()	Returns true if the string starts with the specified value
27	strip()	Returns a trimmed version of the string
28	swapcase()	Swaps cases, lower case becomes upper case and vice versa
29	title()	Converts the first character of each word to upper case
30	upper()	Converts a string into upper case
31	rfind()	Searches the string for a specified value and returns the last position of where it was found
32	rindex()	Searches the string for a specified value and returns the last position of where it was found
33	rjust()	Returns a right justified version of the string
34	rsplit()	Splits the string at the specified separator, and returns a list
35	rstrip()	Returns a right trim version of the string
36	splitlines()	Splits the string at line breaks and returns a list
37	translate()	Returns a translated string
38	zfill()	Fills the string with a specified number of 0 values at the beginning

1. capitalize ():

- Converts the first character to upper case

Syntax: string.capitalize()

Example: 1

```
>>> str="python is a trending programming language"
>>>.str.capitalize()
```

Output:'Python is a trending programming language'

Example: 2

```
>>> txt='522616 is a pincode'
>>> txt.capitalize()
```

Output: '522616 is a pincode'

2. casefold ():

- Converts string into lower case
- This method is similar to the lower() method, but the casefold() method is stronger, more aggressive, meaning that it will convert more characters into lower case, and will find more matches when comparing two strings and both are converted using the casefold() method.

Syntax: *string*.casefold()

Example: 1

```
>>> str='PYTHON IS A TRENDING PROGRAMMING LANGUAGE'
>>> str.casefold()
```

Output: 'python is a trending programming language'

3. center ():

- The center() method will center align the string, using a specified character (space is default) as the fill character.

Syntax: string.center(length, character)

Here,

length	= Required. The length of the returned string.
Character	= Optional. The character to fill the missing space on each side. Default is " " (space)

Example-1: Print the word "Engineering", taking up the space of 128 characters, with " Engineering " in the middle:

```
str1="Engineering"
```

```
>>> str1.center(128)
```

Output: '______________________________Engineering______________________________'

```
>>> str1.center(28)
```

Output: '____Engineering____'

Example-2: Using the letter "O" as the padding character:

```
>>>str1="Engineering"
>>>str1.center(28,'o')
```

Output: 'ooooooooEngineeringooooooooo'

4. count ():

- The count() method returns the number of times a specified value appears in the string.

Syntax: string.count (value, start, end)

Here;

Value	=Required. A String. The string to value to search for
Start	= Optional. An Integer. The position to start the search. Default is 0
end	= Optional. An Integer. The position to end the search. Default is the end of the string

Example-1: Return the number of times the value "is" appears in the string:

```
>>> str2='my favourite subject is programming, python is a programming language'
>>> str2.count('is')
```

Output: 2

Example-2: Search from position 10 to 24:

```
>>> str2='my favourite subject is programming, python is a programming language'
>>>str2.count('is',20,40)
```

Output: 1

5. encode ():

- The encode() method encodes the string, using the specified encoding.
- If no encoding is specified, UTF-8 will be used.

Syntax: string.encode(encoding=encoding, errors=errors)

Here,

encoding	=	Optional. A String specifying the encoding to use. Default is UTF-8
errors	=	Optional. A String specifying the error method.

Legal values are:

- 'backslashreplace' - uses a backslash instead of the character that could not be encoded
- 'ignore'- ignores the characters that cannot be encoded
- 'namereplace'- replaces the character with a text explaining the character
- 'strict' - Default, raises an error on failure
- 'replace'- replaces the character with a questionmark
- 'xmlcharrefreplace' - replaces the character with an xml character

Example-1:

```
txt = "My name is Ståle"
print(txt.encode(encoding="ascii",errors="backslashreplace"))
print(txt.encode(encoding="ascii",errors="ignore"))
print(txt.encode(encoding="ascii",errors="namereplace"))
print(txt.encode(encoding="ascii",errors="replace"))
print(txt.encode(encoding="ascii",errors="xmlcharrefreplace"))
```

Output:

```
b'My name is St\\xe5le'
b'My name is Stle'
b'My name is St\\N{LATIN SMALL LETTER A WITH RING ABOVE}le'
b'My name is St?le'
b'My name is Ståle'
```

6. endswith():

- The endswith() method returns True if the string ends with the specified value, otherwise False.

Syntax: *string*.endswith(*value, start, end*)

Here,

value	Required. The value to check if the string ends with
start	Optional. An Integer specifying at which position to start the search
end	Optional. An Integer specifying at which position to end the search

Example-1:

```
txt = "Hello, welcome to my world."
x = txt.endswith("my world.")
print(x)
```

Output: True

Example-2:

```
txt = "Hello, welcome to my world."
x = txt.endswith("my world.", 5, 11)
print(x)
```

Output: False

7. expandtabs( ):

- The expandtabs() method sets the tab size to the specified number of whitespaces.

Syntax: string.expandtabs(tabsize)

Here,

tabsize	Optional. A number specifying the tabsize. Default tabsize is 8

Example-1:

```
txt = "H\te\tl\tl\to"
x =  txt.expandtabs(2)
print(x)
```

Output: H e l l o

Example-2:

```
txt = "H\te\tl\tl\to"
print(txt)
print(txt.expandtabs())
print(txt.expandtabs(2))
print(txt.expandtabs(4))
print(txt.expandtabs(10))
```

Output:

```
H       e       l       l       o
H       e       l       l       o
H e l l o
H   e   l   l   o
H         e         l         l         o
```

8. find():

- The find() method finds the first occurrence of the specified value.
- The find() method returns -1 if the value is not found.
- The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found. (See example below).

Syntax: string.find(value, start, end)

Here,

value	Required. The value to search for
start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

Example-1:

```
txt = "Hello, welcome to my world."
x = txt.find("welcome")
print(x)
```

Output: 7

Example-2:

```
txt = "Hello, welcome to my world."
x = txt.find("e", 5, 10)
print(x)
```

Output: 8

Example-3:

```
txt = "Hello, welcome to my world."
print(txt.find("q"))
print(txt.index("q"))
```

Output:

```
-1
Traceback (most recent call last):
  File "demo_ref_string_find_vs_index.py", line 4 in <module>
    print(txt.index("q"))
ValueError: substring not found.
```

9. index()

- The index() method finds the first occurrence of the specified value.
- The index() method raises an exception if the value is not found.
- The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not found. (See example below)

Syntax: string.index(value, start, end)

Here,

value	Required. The value to search for
start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

Example-1:

```
txt = "Hello, welcome to my world."
x = txt.index("welcome")
print(x)
```

Output: 7

Example-2:

```
txt = "Hello, welcome to my world."
x = txt.index("e")
print(x)
```

Output: 1

10. isalnum():

- The isalnum() method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).
- Example of characters that are not alphanumeric: (space)!#%&? etc.

Syntax: string.isalnum()

Example-1:

```
txt = "Company12"
x = txt.isalnum()
print(x)
```

Output: True

Example-2:

```
txt = "Company 12"
x = txt.isalnum()
print(x)
```

Output: False , because white space before 12.

11. isalpha():

- The isalpha() method returns True if all the characters are alphabet letters (a-z).
- Example of characters that are not alphabet letters: (space)!#%&? etc.

Syntax: string.isalpha()

Example-1:

```
txt = "CompanyX"
x = txt.isalpha()
print(x)
```

Output: True

Example-2:

```
txt = "Company10"
x = txt.isalpha()
print(x)
```

Output: False

12. isdecimal( ):

- The isdecimal() method returns True if all the characters are decimals (0-9).
- This method is used on unicode objects.

Syntax: string.isdecimal()

Example-1:

```
txt = "\u0033" #unicode for 3
x = txt.isdecimal()
print(x)
```

Output: True

Example-2:

```
a = "\u0030" #unicode for 0
b = "\u0047" #unicode for G
print(a.isdecimal())
print(b.isdecimal())
```

Output:

```
True
False
```

13. isdigit( ):

- The isdigit() method returns True if all the characters are digits, otherwise False.
- Exponents, like ², are also considered to be a digit.

Syntax: *string*.isdigit()

Example-1:

```
txt = "50800"
x = txt.isdigit()
print(x)
```

Output: True

Example-2:

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for ²
print(a.isdigit())
print(b.isdigit())
```

Output:

```
True
True
```

14. isidentifier( ):

- The isidentifier() method returns True if the string is a valid identifier, otherwise False.
- A string is considered a valid identifier if it only contains alphanumeric letters (a-z) and (0-9), or underscores (_).
- A valid identifier cannot start with a number, or contain any spaces.

Syntax: string.isidentifier()

Example-1:

```
txt = "Demo"
x = txt.isidentifier()
print(x)
```

Output: True

Example-2:

```
a = "MyFolder"
c = "2bring"
print(a.isidentifier())
print(c.isidentifier())
```

Output:

```
True
False
```

15. islower():

- The islower() method returns True if all the characters are in lower case, otherwise False.
- Numbers, symbols and spaces are not checked, only alphabet characters.

Syntax: string.islower()

Example-1:

```
txt = "hello world!"
x = txt.islower()
print(x)
```

Output: True

Example-2:

```
a = "Hello world!"
b = "hello 123"
c = "mynameisPeter"
print(a.islower())
print(b.islower())
print(c.islower())
```

Output:

```
False
True
False
```

16. isnumeric():

- The isnumeric() method returns True if all the characters are numeric (0-9), otherwise False.
- Exponents, like ² and ¾ are also considered to be numeric values.

Syntax: string.isnumeric()

Example-1:

```
txt = "565543"
x = txt.isnumeric()
print(x)
```

Output: True

Example-2:

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for ²
c = "10km2"
print(a.isnumeric())
print(b.isnumeric())
print(c.isnumeric())
```

Output:

```
True
True
False
```

17. isprintable():

- The isprintable() method returns True if all the characters are printable, otherwise False.
- Example of none printable character can be carriage return and line feed.

Syntax: string.isprintable()

Example-1:

```
txt = "Hello! Are you #1?"
x = txt.isprintable()
print(x)
```

Output: True

Example-2:

```
txt = "Hello!\nAre you #1?"
x = txt.isprintable()
print(x)
```

Output: False

18. isspace():

- The isspace() method returns True if all the characters in a string are whitespaces, otherwise False.

Syntax: string.isspace()

Example-1:

```
txt = "   "
x = txt.isspace()
print(x)
```

Output: True

Example-2:

```
txt = "  s   "
x = txt.isspace()
print(x)
```

Output: False

19. istitle():

- The istitle() method returns True if all words in a text start with a upper case letter, AND the rest of the word are lower case letters, otherwise False.
- Symbols and numbers are ignored.

Syntax: string.istitle()

Example-1:

```
txt = "Hello, And Welcome To My World!"
x = txt.istitle()
print(x)
```

Output: True

Example-2:

```
a = "HELLO, AND WELCOME TO MY WORLD"
b = "Hello"
c = "22 Names"
d = "This Is %'!?"
print(a.istitle())
print(b.istitle())
print(c.istitle())
print(d.istitle())
```

Output:

```
False
True
True
True
```

20. isupper()

- The isupper() method returns True if all the characters are in upper case, otherwise False.
- Numbers, symbols and spaces are not checked, only alphabet characters.

Syntax: string.isupper()

Example-1:

```
txt = "THIS IS NOW!"
x = txt.isupper()
print(x)
```

Output: True

Example-2:

```
a = "Hello World!"
b = "hello 123"
c = "MY NAME IS PETER"
print(a.isupper())
print(b.isupper())
print(c.isupper())
```

Output:

```
False
False
True
```

21. join():

- The join() method takes all items in an iterable and joins them into one string.
- A string must be specified as the separator.

Syntax: string.join(iterable)

Here,

iterable	Required. Any iterable object where all the returned values are strings

Example-1:

```
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print(x)
```

Output:

```
John#Peter#Vicky
```

Example-2:

```
myDict = {"name": "John", "country": "Norway"}
mySeparator = "TEST"
x = mySeparator.join(myDict)
print(x)
```

Output:

```
nameTESTcountry
```

22. ljust():

- The ljust() method will left align the string, using a specified character (space is default) as the fill character.

Syntax: string.ljust(length, character)

Here,

length	Required. The length of the returned string
character	Optional. A character to fill the missing space (to the right of the string). Default is " " (space).

Example-1:

```
txt = "banana"
x = txt.ljust(20)
print(x, "is my favorite fruit.")
```

Output:

```
banana               is my favorite fruit.
```

Example-2:

```
txt = "banana"
x = txt.ljust(20, "O")
print(x)
```

Output:

```
bananaOOOOOOOOOOOOOO
```

Example-3:

```
txt = "banana"
x = txt.ljust(20, "so")
print(x)
```

Output:

```
Traceback (most recent call last):
  File "./prog.py", line 3, in <module>
TypeError: The fill character must be exactly one character long
```

23. lower():

- The lower() method returns a string where all characters are lower case.
- Symbols and Numbers are ignored.

Syntax: string.lower()

Example-1:

```
txt = "Hello my FRIENDS"
x = txt.lower()
print(x)
```

Output: hello my friends

24. replace():

- The replace() method replaces a specified phrase with another specified phrase.
- Note: All occurrences of the specified phrase will be replaced, if nothing else is specified.

Syntax: string.replace(oldvalue, newvalue, count)

Here,

oldvalue	Required. The string to search for
newvalue	Required. The string to replace the old value with
count	Optional. A number specifying how many occurrences of the old value you want to replace. Default is all occurrences.

Example-1:

```
txt = "I like bananas"
x = txt.replace("bananas", "apples")
print(x)
```

Output: I like apples

Example-2:

```
txt = "one one was a race horse, two two was one too."
x = txt.replace("one", "three", 2)
print(x)
```

Output: "**three three** was a race horse, two two was **one** too."

25. split():

- The split() method splits a string into a list.
- You can specify the separator, default separator is any whitespace.
- Note: When maxsplit is specified, the list will contain the specified number of elements plus one.

Syntax: string.split(separator, maxsplit)

Here,

separator	Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator
maxsplit	Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

Example-1:

```
txt = "welcome to the jungle"
x = txt.split()
print(x)
```

Output:

```
['welcome', 'to', 'the', 'jungle']
```

Example-2:

```
txt = "apple#banana#cherry#orange"
x = txt.split("#")
print(x)
```

Output:

```
['apple', 'banana', 'cherry', 'orange']
```

26. startswith():

- The startswith() method returns True if the string starts with the specified value, otherwise False.

Syntax: string.startswith(value, start, end)

Here,

value	Required. The value to check if the string starts with
start	Optional. An Integer specifying at which position to start the search
end	Optional. An Integer specifying at which position to end the sear

Example-1:

```
txt = "Hello, welcome to my world."
x = txt.startswith("Hello")
print(x)
```

Output: True

Example-2:

```
txt = "Hello, welcome to my world."
x = txt.startswith("wel", 7, 20)
print(x)
```

Output: True

27. strip( ):

- The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)

Syntax: string.strip(characters)

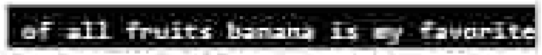
Here,

characters	Optional. A set of characters to remove as leading/trailing characters.

Example-1:

```
txt = "     banana     "
x = txt.strip()
print("of all fruits", x, "is my favorite")
```

Output:



```
of all fruits banana is my favorite
```

Example-2:

```
txt = ",,,,,rrttgg.....banana....rrr"
x = txt.strip(",.grt")
print(x)
```

Output:

```
banana
```

28. swapcase( ):

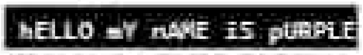
- The swapcase() method returns a string where all the upper case letters are lower case and vice versa.

Syntax: string.swapcase()

Example-1:

```
txt = "Hello My Name Is Purple"
x = txt.swapcase()
print(x)
```

Output:



```
hELLO mY nAME iS pURPLE
```

29. title() :

- The title() method returns a string where the first character in every word is upper case. Like a header, or a title.
- If the word contains a number or a symbol, the first letter after that will be converted to upper case.

Syntax: *string*.title()

Example-1:

```
txt = "Welcome to my world"
x = txt.title()
print(x)
```

Output: **Welcome To My World**

Example-2:

```
txt = "hello b2b2b2 and 3g3g3g"
x = txt.title()
print(x)
```

Output: **Hello B2B2B2 And 3G3G3G**

30. upper():

- The upper() method returns a string where all characters are in upper case.
- Symbols and Numbers are ignored.

Syntax: string.upper()

Example-1:

```
txt = "Hello my friends"
x = txt.upper()
print(x)
```

Output: **HELLO MY FRIENDS**

31. rfind():

- The rfind() method finds the last occurrence of the specified value.
- The rfind() method returns -1 if the value is not found.
- The rfind() method is almost the same as the rindex() method. See example below.

Syntax: string.rfind(value, start, end)

Here,

value	Required. The value to search for
start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

Example-1:

```
txt = "Mi casa, su casa."
x = txt.rfind("casa")
print(x)
```

Output: 12

Example-2:

```
txt = "Hello, welcome to my world."
x = txt.rfind("e", 5, 10)
print(x)
```

Output: 8

Example-3:

```
txt = "Hello, welcome to my world."
print(txt.rfind("q"))
print(txt.rindex("q"))
```

Output:

```
-1
Traceback (most recent call last):
  File "demo_ref_string_rfind_vs_rindex.py", line 4 in <module>
    print(txt.rindex("q"))
ValueError: substring not found
```

32. rindex():

- The rindex() method finds the last occurrence of the specified value.
- The rindex() method raises an exception if the value is not found.
- The rindex() method is almost the same as the rfind() method. See example below.

Syntax: `string.rindex(value, start, end)`

Here,

value	Required. The value to search for
start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

Example-1:

```
txt = "Mi casa, su casa."
x = txt.rindex("casa")
print(x)
```

Output: 12

Example-2:

```
txt = "Hello, welcome to my world."
x = txt.rindex("e", 5, 10)
print(x)
```

Output: 8

33. rjust():

- The rjust() method will right align the string, using a specified character (space is default) as the fill character.

Syntax: string.rjust(length, character)

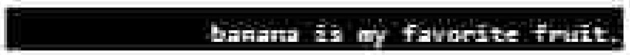
Here,

length	Required. The length of the returned string
character	Optional. A character to fill the missing space (to the left of the string). Default is " " (space).

Example-1:

```
txt = "banana"
x = txt.rjust(20)
print(x, "is my favorite fruit.")
```

Output:



```
              banana is my favorite fruit.
```

34. rsplit():

- The rsplit() method splits a string into a list, starting from the right.
- If no "max" is specified, this method will return the same as the split() method.
- Note: When maxsplit is specified, the list will contain the specified number of elements plus one.

Syntax: string.rsplit(separator, maxsplit)

Here,

separator	Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator
maxsplit	Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

Example-1:

```
txt = "apple, banana, cherry"
# setting the maxsplit parameter to 1, will return a list with 2 elements!
x = txt.rsplit(", ", 1)
print(x)
```

# Note that the result has only 2 elements "apple, banana" is the first element, and "cherry" is the last.

Output: **['apple, banana', 'cherry']**

35. rstrip():

- The rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.

Syntax: string.rstrip(characters)

Here,

characters Optional. A set of characters to remove as trailing characters.

Example-1:

```
txt = "     banana     "
x = txt.rstrip()
print("of all fruits", x, "is my favorite")
```

Output:

```
of all fruits      banana is my favorite
```

36. splitlines()

- The splitlines() method splits a string into a list. The splitting is done at line breaks.

Syntax: string.splitlines(keeplinebreaks)

Example-1:

```
txt = "Thank you for the music\nWelcome to the jungle"
x = txt.splitlines(True)
print(x)
```

Output:

```
['Thank you for the music\n', 'Welcome to the jungle']
```

37. translate():

- The translate() method returns a string where some specified characters are replaced with the character described in a dictionary, or in a mapping table.
- Use the maketrans() method to create a mapping table.
- If a character is not specified in the dictionary/table, the character will not be replaced.
- If you use a dictionary, you must use ascii codes instead of characters.

Syntax: string.translate(table)

Here,

table	Required. Either a dictionary, or a mapping table describing how to perform the replace

Example-1:

```
txt = "Hello Sam!";
mytable = txt.maketrans("S", "P");
print(txt.translate(mytable));
```

Output: Hello Pam!

38. zfill()

- The zfill() method adds zeros (0) at the beginning of the string, until it reaches the specified length.
- If the value of the len parameter is less than the length of the string, no filling is done.

Syntax: string.zfill(len)

Here,

len	Required. A number specifying the position of the element you want to remove.

Example-1:

```
txt = "50"
x = txt.zfill(10)
print(x)
```

Output:

```
0000000050
```

Example-2:

```
a = "hello"
b = "welcome to the jungle"
c = "10.000"
print(a.zfill(10))
print(b.zfill(10))
print(c.zfill(10))
```

Output:

```
00000hello
welcome to the jungle
000010.000
```

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

Example:

415 in binary notation 1100111112

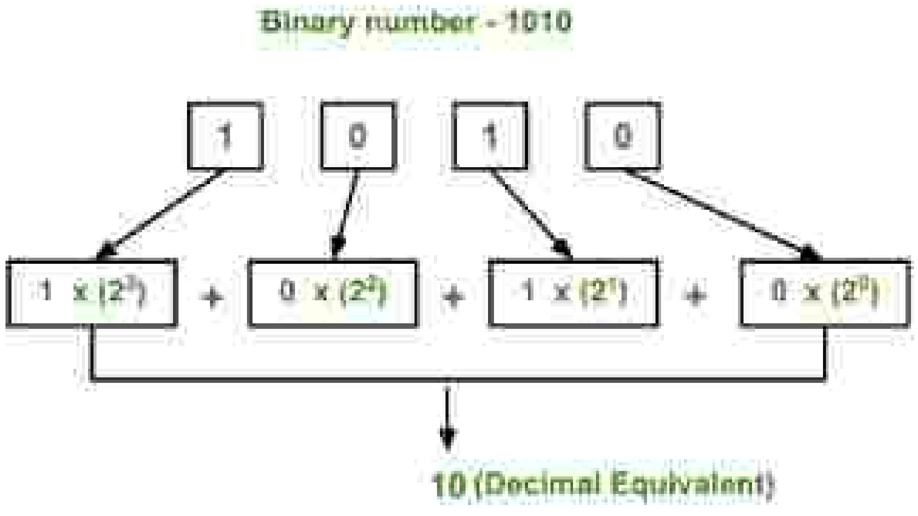
415 in octal notation 6378

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.

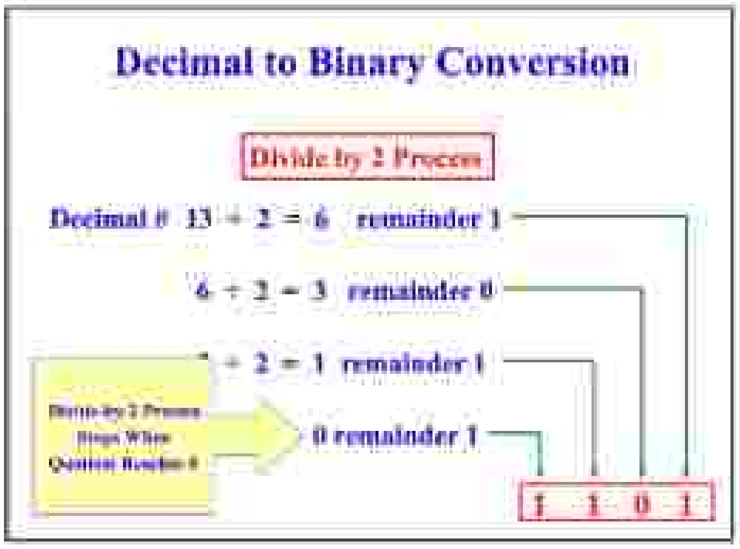
```
bstring = input("Enter a string of bits: ")
decimal = 0
exponent = len(bstring) - 1
for i in bstring:
    decimal = decimal + int(i) * 2 ** exponent
    exponent = exponent - 1
print("The integer value is", decimal)
```

output:

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

<u>Converting Decimal to Binary</u>

- This algorithm repeatedly divides the decimal number by 2. After each division, the remainder (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 initially empty, and the process continues while the decimal number is greater than 0.


Decimal to Binary Conversion
Divide by 2 Process
Decimal # 13 ÷ 2 = 6 remainder 1
6 ÷ 2 = 3 remainder 0
3 ÷ 2 = 1 remainder 1
0 remainder 1
Divide-by-2 Process Stops When Quotient Reaches 0
1 1 0 1

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 a text 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 write files, 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 Line characters 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 of the 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:

Example:
```
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:

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

Example: **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 a file 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")
```

➢ Python file methods:

Method	Description
detach()	Returns the separated raw stream from the buffer
fileno()	Returns a number that represents the stream, from the operating system's perspective
flush()	Flushes the internal buffer
isatty()	Returns whether the file stream is interactive or not
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size
writable()	Returns whether the file can be written to or not
writelines()	Writes a list of strings to the file

readline() Method:

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

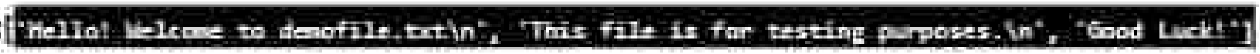
- The readlines() method returns a list containing each line in the file as a list item.
- Use the hint parameter to limit the number of lines returned. If the total number of bytes returned exceeds the specified number, no more lines are returned.

Syntax: file.readlines(hint)
Here,

hint	Optional. If the number of bytes returned exceed the hint number, no more lines will be returned. 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 file stream 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 stream position, 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 to the 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 on each pass. To convert this line to the integer.
- The programmer runs the **string method strip** to remove the newline and then runs the int function 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
```