STRINGS

➔ **len** function is used to find the length of a string
message='Hello Gita'
len(message)
>>> 10

***Note that the indices start with 0(index for accessing the first characters from the left) and the end with one less than the length of the string (index for accessing the last character). The following instructions illustrate how to access individual characters in a string via indices. An index is specified within the square brackets, for example:

```
>>>message[0]
'H'
>>>message[6]
'G'
>>>index=len(message)-1
>>>message[index]
'a'
```

➔ The expression **message[0]** yields character '**H**' since '**H**' is stored at index 0. Similarly, the value of **len(message)** being **10**, when **index** is set equal to **len(message)-1**, **message[index]** yields the last character of the string **message** at index **9,** i.e., '**a'.** Sometimes it is more convenient to access the elements of a string from the right end. For this purpose, Python provides negative indices. The negative indices range from – *(length of the string)* to **-1**. For the string **message**, the negative indices would range from **-10** to **-1.**

➔ **>>>message[-1]**
'a'

➔ **>>>message[-index]**
'e'

➔ Strings in Python are immutable. i.e., a component of a string cannot be altered, and attempt to do so would lead to an error:

➔ >>>**message[6]=**'S' TypeError: 'str' object does not support item assignment

➔ As mentioned earlier, strings can be concatenated using concatenation operator **+** and can be repeated a multiple number of times using the operator *.

➔ >>>'Computer' + 'Science'

➔ 'Computer Science'

➔ >>>'Hi'*3

➔ ‘HiHiHi’

➔ Strings may also be compared using relational operators mentioned earlier. Also the functions **min** and **max** may be used to find the maximum and minimum respectively of several values, for example:
>>>**max(‘AZ’, ‘C’, ‘BD’, ‘BT’) ‘C’**
>>>min(‘BD’, ‘AZ’, ‘C’) ‘AZ’
>>>max(‘hello’, ‘How’, ‘Are’, ‘You’, ‘sir’) ‘sir’

Slicing

➔ Sometimes we need to retrieve a substring, also called a slice, from a string. This can be done by specifying an index range. To extract a substring comprising the character sequence having indices from **start** to **end-1,** we specify the range in the form **start:end**
➔ >>>message=’Hello Sita’
➔ Message[0:5]
➔ ‘Hello’
➔ If the value of **start** or **end** is not specified, Python assumes 0 as the default value of **start**, and length of the string as the default value of **end**,
>>>**message[:5]**
‘Hello’
>>>message[5:]
‘ Sita’
>>>message[:]
‘Hello Sita’
>>>message[:5]+message[5:]
‘Hello Sita’
>>>message[:15]
‘Hello Sita’
>>>message[15:]
‘’
>>>message[:15]+message[15:]
‘Hello Sita’
➔ Note that messge[:n]+message[n:] always yields message, irrespective of the value of n being negative, positive, or even out of range. Indeed, values used for defining a slice may be arbitrary integers or even None, for example:
➔ >>>message[8:20]
➔ ‘ta’
➔ >>>message[6:None]
➔ ‘Sita’

➔ Apart from extracting a consecutive subsequence of characters from a string, Python also allows us to extract a subsequence of the form ***start:end:inc***. This subsequence will include every **inc**th element of the sequence in the range ***start*** to ***end-1,*** for example:
➔ >>>message[0:len(message):2]
➔ 'HloSt'
➔ >>>message[0:len(message):3]
➔ 'HlSa'

Membership

➔ Python also allows us to check for membership of the individual characters or substrings in strings using **in** operator.
➔ >>>'h' in 'hello' yields **True**
➔ >>> 'h' in 'Hello' yields **False**

Built-in String Functions

- count

  to find the occurrence of a character in a string

  'Encyclopedia'.count('e')

  2

  vowels= 'aeiou'

  vowelCount= 0

  for ch in vowels:

  vowelCount+= 'Encyclopedia'.count(ch)

  vowelCount

- find and rfind

  To find whether a string is present as a substring in another string.

  >>>colors= 'green, red, blue, red, red, green'

  >>>colors.find('red')

  7

To find the last occurrence of a string, we use **rfind** that scans the string in reversed order from right of the string to the beginning:

>>>colors.rfind('red')

23

***If the function fails to find the desired substring, it returns -1:**

>>>colors.find('orange')

-1

- **capitalize, title, lower, upper** and **swapcase**
  - **capitalize** is used to convert the first letter of a string to uppercase character and converting the remaining letters in the string to lowercase(if not already so).

    >>>'python IS a Language'.capitalize()

    'Python is a language'
  - **title** is used to convert each first letter of the words in a string to uppercase and the remaining to lowercase(if not already so).

    >>>'python IS a PROGRAMMING Language'.title()

    'Python Is A Prorgramming Language'
  - **lower** and **upper** used to convert all letters in a string to lowercase and uppercase, as specified.
  - **swapcase** is used to convert lowercase letters into uppercase and vice versa.
- **islower, isupper, istitle**

  to check whether all letters are in lowercase or uppercase, respectively.

  'python'.islower() True

  'Python'.isupper() False

  istitle is used to check whether a strig is in titlecase

>>> ‘Basic Python Programming’.istitle()

True

>>> ‘Basic PYTHON Programmin’.istitle()

False

>>>’123’.istitle()

False

>>> ‘Book 123’.istitle()

True

- replace

  to replace a part of a string by another string. It takes two arguments as inputs. First one specifies the string to be replaced and the second argument the string to be replaced with.

  >>> message= ‘Amey my friend, Amey my guide’

  >>>message.replace(‘Amey’, ‘Rami’)

  ‘Rami my friend, Rami my guide’

- strip, lstrip and rstrip

  lstrip and rstrip is used to remove whitespaces from the beginning and end.

  strip is used to remove whitespaces both from beginning and end.

  *We may choose to remove any other character(s) from the beginning or end by explicitly passing the characters as an argument to the function.

  >>> ‘ Hello, how are you? ‘.lstrip()

  ‘Hello, how are you? ‘

  >>> ‘ Hello, kemcho! ‘.rstrip()

  ‘ Hello, kemcho!’

>>> ‘ Hello, kaa karru baba? ’.strip()

‘Hello, kaa karru baba?’

- **split** and **partition**

  to split a string into a list of strings based on a delimiter.

  >>>colors= ‘Green, Red, Blue, Yellow’

  >>> colors.split(‘,’)

  [‘Green’, ‘ Red’, ‘ Blue’, ‘ Yellow’]

  if no delimiter is used then it takes whitespace as default

  >>> colors.split()

  [‘Green,’, ‘Red,’, ‘Blue,’, ‘Yellow’]

  partition divides a string into two parts based on a delimiter and returns a tuple comprising string before delimiter, the delimiter itself, and the string after the delimiter, for example:

  >>> ‘Hello. How are you?’.partition(‘.’)

  (‘Hello’, ‘.’, ‘How are you?’)

- **join**

  returns a string comprising elements of a sequence separated by the specified delimiter

  >>> ‘ > ’.join([‘I’, ‘am’, ‘ok’])

  ‘I > am > ok’

  >>> ‘ ’.join([‘I’, ‘am’, ‘ok’])

  ‘I am ok’

  >>> ‘ > ’.join([“‘I’, ‘am’, ‘ok’”])

  “‘>I >’>, am > ok’

- **isspace, isalpha, isdigit** and **isalnum**

  enables us to check whether a value is of desired type.

```
>>> name= input('Enter your name :')
Enter your name : Babui
>>> name.isalpha()
True
>>> name= input('Enter your name :')
Enter your name : Babui Gamer
>>> name.isalpha()
False
```
cause the blank character in between is not an alphabet

```
>>>mobile=input('Enter mobile number :')
Enter mobile number : 1234567890
>>> mobile.isdigit() and len(mobile) == 10
True
```

isspace() is used to check if the string has any whitespaces

```
>>> '    \n\t  '.isspace()
True
```

isalnum() is used to check if a string comprises of alphabets and digits only

```
>>> name= 'WB54C7192'
>>>name.isalphanum()
True
>>> name= 'WB 54C 7192'
>>>name.isalphanum()
False
```
Cause of blank spaces in between

- startswith and endswith

  >>> name= ‘Naina Talwar’

  >>>name.startswith(‘Nai’)

  True

  >>> name= ‘Kabir Thapar’

  >>>name.endswith(‘ Thapar’)

  True

  >>> name= ‘ Babui’

  >>>name.startswith(‘Ba’)

  False

- encode and decode

  >>>str1= ‘message’.encode(‘utf32’)

  >>>str1