What is String?

In [ ]:
```
String --> Any sequence of character either in double quotes or single quotes. are known as String.
       --> Strings are Immutable.
       --> Indexing is important in case of String(ordered collection of characters)
Example:
    x="Python"
    x="python"
    x="Python"
    x="10"
    x="10.5"
    x="True"
    x="10+2j"
    x='Hello'
    x='python'
```

How to define multiline literals in string?

In [7]:
```
Note--> If you want to represent a string in multiple lines then we can use triple quotes
x="""Good
Morning
India"""
print(x)
```
```
Good
Morning
India
```

Important Concept Related to String

In [ ]:
```
If you want to represent  a quotes inside a quotes then it mandatory then both the quotes are
differenet type.

Example:
    if you are represting a string with double quotes and inside that string if there is any
    requirement of another quote then it is necessary to use single quote inside the double quotes else
    you will get an error.

"" --> Inside that double quote again if you want to represent a quote then it is mandatory to use
       single quotes
' '--> Inside that single quote again if you want to represent a quote then it is mandatory to use
       double quotes
```

In [1]:
```
x="It's very hot today"
x
```
Out[1]:
```
"It's very hot today"
```

In [2]:
```
x="It's Sunday"
x
```
Out[2]:
```
"It's Sunday"
```

In [3]:
```
x="It's good to have a cup of tea"
x
```
Out[3]:
```
"It's good to have a cup of tea"
```

In [4]:
```
x="On 23rd of October 'India Will Play with Pakistan' "
x
```
Out[4]:
```
"On 23rd of October 'India Will Play with Pakistan' "
```

In [5]:
```
x='On 23rd of October "India Will Play with Pakistan"'
print(x)
```
```
On 23rd of October "India Will Play with Pakistan"
```

How we access the characters of a string:

In [ ]:
```
1.By indexing
2.By Slicing
```

Indexing in String(Python)

In [ ]:
```
Indexing --> We can access the character of a string based on the given index(key)
Python supports Two types of Indexing:
    1.Positive Indexing --> left to right (forward direction) -->first element index will be 0
    2.Negative Indexing --> Right to left (backward direction) -->first element index will be -1
```

Examples:

In [7]:
```
#by indexing --> Access the character:
x="Python is Easy Programming Language"
print(x[7])
print(x[-28])
print(x[-1])
print(x[-2])  #g
print(x[13])  #y
print(x[-6])  #n
#print(x[200]) #Error
#print(x[-200])  #error

Note --> In case of Indexing if the given index is not present in the string then indexing will give
         you an error
```
```
i
i
e
g
y
n
```

In [9]:
```
x="Python"
for i in range(len(x)):  # for i in range(6): #0,1,2,3,4,5
    print("Positive index of "+str(x[i])+" "+str(i)+" Negative index of "+str(x[i]+" "+str(i-len(x))))
```
```
Positive index of P 0 Negative index of P -6
Positive index of y 1 Negative index of y -5
Positive index of t 2 Negative index of t -4
Positive index of h 3 Negative index of h -3
Positive index of o 4 Negative index of o -2
Positive index of n 5 Negative index of n -1
```

Slicing in String(Python)

In [ ]:
```
#By SLicing --> access a piece of string

Syntax of Slicing:
    string_name[begin_index : ending_index-1:steps]

Begin_index --> From where we have to consider the slice(substring)
End_index --> last index of the string for termination
step --> Increment or decrement

Note: all these begin_index , ending_index, steps all are optional. if you are not giving anything
      then PVM will consider the whole string.
```

Example

In [10]:
```
x="Python is a High level programming Language"
x[::]
```
Out[10]:
```
'Python is a High level programming Language'
```

In [11]:
```
x[0:16]
```
Out[11]:
```
'Python is a High'
```

Important

In [ ]:
```
Note --> if you are not giving any value(begin_index,ending_index,step) then PVM will treat
all these valus as default  values of begin_index,ending_index,step.

Begin_index --> default value is 0
Ending_index --> length of the string
Step --> Value will be 1
```

Examples

In [12]:
```
#Examples:
x="learning python is easy"
x[:10]
```
Out[12]:
```
'learning p'
```

In [13]:
```
x="learning python is easy"
x[2:]
```
Out[13]:
```
'arning python is easy'
```

In [14]:
```
x="learning python is easy"
x[3:9]
```
Out[14]:
```
'rning '
```

In [15]:
```
x="learning python is easy"
x[7:]
```
Out[15]:
```
'g python is easy'
```

In [16]:
```
x="learning python is easy"
x[:]
```
Out[16]:
```
'learning python is easy'
```

In [18]:
```
x="learning python is easy"
x[10000:20000000000000000:9]
Note: In case of slicing you will never get an error weather you are doing slicing in forward
      direction or backward direction.
```
Out[18]:
```
''
```

Behaviour of Slicing

In [ ]:
```
Note --> if you are not giving any value(begin_index,ending_index,step) then PVM will treat
all these valus as default  values of begin_index,ending_index,step.

Begin_index --> default value is 0
Ending_index --> length of the string
Step --> Value will be 1

Step value is reponsible for forward slicing or backward slicing

if step value is +ve --> slicing will be done in forward direction(left to right)
if step value is -ve --> slicing will be done in backward direction (righ to left)
```

Examples

In [19]:
```
x="Python is Good"
x[1:9:2]
```
Out[19]:
```
'yhni'
```

In [20]:
```
x="Python is Good"
x[-3:-8:1]
```
Out[20]:
```
''
```

In [ ]:
```
In case of backward slicing --> default value for starting index is -1
                                default value for ending index is -len(x)
x=[::-1]
```

In [47]:
```
x="Python is Good"
x[::-1]
```
Out[47]:
```
'dooG si nohtyP'
```

Operators Used in String

In [ ]:
```
Operators that are used in String:
    a.Concatenation Operator(+ operator) --> if you want to add two strings then you can use +
      operator
    b.Repetition Opertaor (* Opertaor)--> if you want to repeat  a string multiple times then we can
    use * operator
    c.Membership Operator(in and not in) --> if the given character or substring is presnent in the
    given string or not if it is present then it will give you True else False
    d.Identity operator(is and not is)
    e.Logical Operator(and or and not)
```

Examples of Each Operator

In [28]:
```
a="Pratyush"
b="Srivastava"
c="BTECH"
d="hello"
a+b+c+d
#If you want to perform cancatenation of string with + operator then it is mandatory
#that both the operands are of string type only
```
Out[28]:
```
'PratyushSrivastavaBTECHhello'
```

In [23]:
```
a="Pratyush"
a*5
If you want to perform Repeaition of string with * operator then it is mandatory
that one operand is of string type and other operand is of int type.
```
Out[23]:
```
'PratyushPratyushPratyushPratyushPratyush'
```

In [29]:
```
x="Python is good"
"p" in x
```
Out[29]:
```
False
```

In [30]:
```
x="Python is good"
"p" not in x
```
Out[30]:
```
True
```

In [31]:
```
x="Python1"
y="Python2"
x is y
```
Out[31]:
```
False
```

In [27]:
```
x="Python1"
y="Python2"
x is not y
```
Out[27]:
```
True
```

In [32]:
```
"hello" and "world"
```
Out[32]:
```
'world'
```

Strip Function

In [ ]:
```
strip --> Inbuilt for string that is use to remove the extra spaces
Syntax:
    string_name.strip()
```

In [33]:
```
x="        Python is              good    "
x.strip()
```
Out[33]:
```
'Python is              good'
```

Find and Index Function

In [ ]:
```
finding the substring from a string
1.find()--> return the first occurance of the given substring if the element is not present then
            you will get -1
2.index() -->return thr first occurance of the sub string if the element is not presenet then you will
             get an error
```

In [66]:
```
#find -->s.find(substring , begin_index ,end_index)
x="Learning Python is very easy"
print(x.find("Python"))
print(x.find("Python",0,15)) #-1
print(x.find("Python",0,8)) #-1
```
```
9
9
-1
```

In [68]:
```
#index -->s.index(substring , begin_index ,end_index)
x="Learning Python is very easy"
print(x.index("Python"))  #9
print(x.index("Python",0,15)) #-1
print(x.index("Python",0,8)) #-1
```
```
9
9
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [68], in <cell line: 5>()
      3 print(x.index("Python"))  #9
      4 print(x.index("Python",0,15)) #-1
----> 5 print(x.index("Python",0,8))

ValueError: substring not found
```

upper(),lower(),title(),capitalize

In [34]:
```
x="pratyush"
x.upper() #Upper Function is used to convert whole string in Upper Case
```
Out[34]:
```
'PRATYUSH'
```

In [35]:
```
x="PRATYUSH"
x.lower() #lower Function is used to convert whole string in lower Case
```
Out[35]:
```
'pratyush'
```

In [36]:
```
x="pratyush srivastava"
x.title() #title Function is used to convert first letter of each word in Upper Case
```
Out[36]:
```
'Pratyush Srivastava'
```

In [37]:
```
x="pratyush srivastava"
x.capitalize() #capitalize Function is used to first letter of the string in upper case
```
Out[37]:
```
'Pratyush srivastava'
```

In [87]:
```
x="pratyush srivastava"
x[0].upper()+x[1:6]+x[7].upper()+x[8:len(x)-1]+x[-1].upper()
```
Out[87]:
```
'PratyuH srivastavA'
```

isupper(),islower() ,isalpha and isnumeric()

In [38]:
```
x="pratyusH srivastava"
x.islower() #return true if all content are of lower case
```
Out[38]:
```
False
```

In [39]:
```
x="PRATYUSH"
x.isupper() #return true if all content are of upper case
```
Out[39]:
```
True
```

In [40]:
```
x="pratyushSrivastava"
x.isalpha() #return true if all content is in alphabetical format
```
Out[40]:
```
True
```

In [80]:
```
x="123"
x.isnumeric() #return True if all content is in numeric format
```
Out[80]:
```
True
```

In [41]:
```
x="123"
x.isdigit() #return True if all content is in digit format
```
Out[41]:
```
True
```

In [ ]: