String Handling Concept:

- A group/sequence of characters is called String.
- Python supports **str** data type to represent string type data.
- String objects are immutable objects that mean we can’t modify the existing string object.
- Insertion order is preserved in string objects.
- Every character in the string object is represented with unique index.
- Python supports both forward and backward indexes.
- Forward index starts with 0 and negative index starts with -1
- Python string supports both "concatenation" and "multiplication" of string objects.
- Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

Quotations in Python:

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- Generally triple quotes are used to write the string across multiple lines. For example, all the following are legal.

  For example1 :

  ```
  word  =  'word'
  sentence  =  "This is a sentence."
  Paragraph   =   """This is a paragraph. It is
           made up of multiple lines and sentences."""
  ```

Example :

```
      0    1     2    3    4    5    6    7
s =  S     r     i    n    i    v    a    s
     -8   -7    -6   -5   -4   -3   -2   -1
```

- If the given index is not available in the string we will get exception like **IndexError**

  ```
  >>> s[8]
  ```

  IndexError: string index out of range.

- If we try to modify the content of string object by using index we will get the **TypeError**

  >>> s[2] = 'x'

  TypeError: 'str' object does not support item assignment

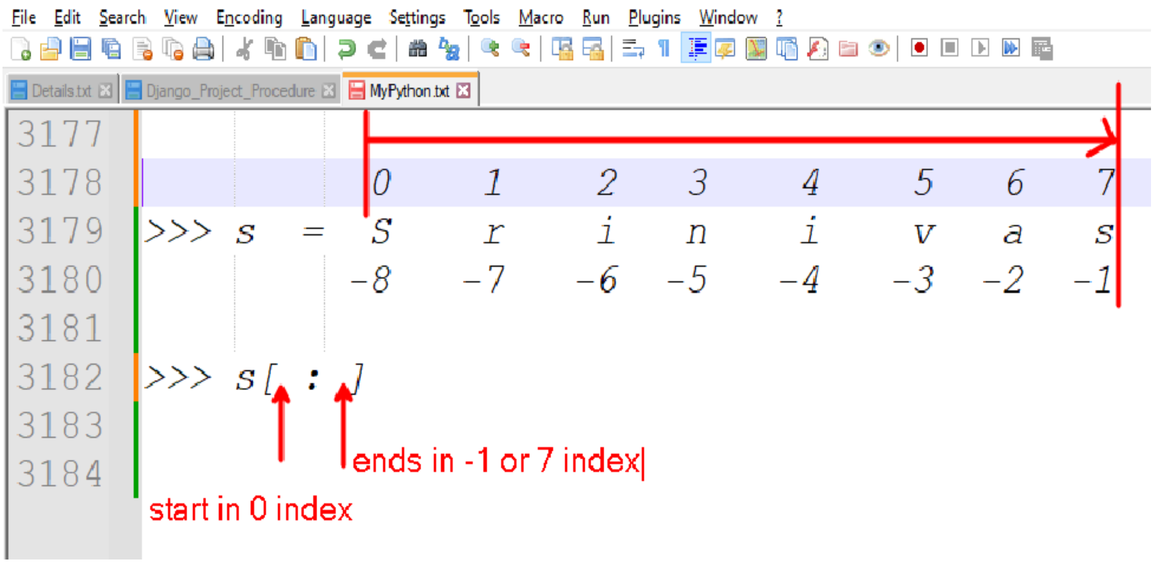
String Slicing :

- column(:) is a slice operator , which is used to extract the require content from the given string using given index values.
- [startIndex : endIndex] : Here, start index is 0 position and end index is -1 position.

Example:

>>> print(x)

Srinivas



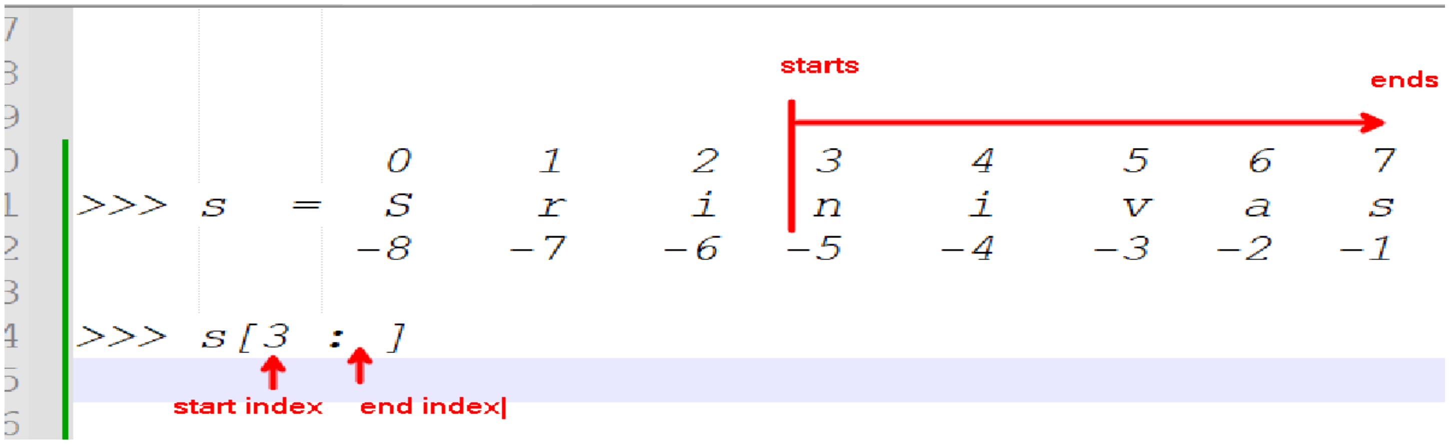
```
                0   1   2   3   4   5   6   7
>>> s  =        S   r   i   n   i   v   a   s
               -8  -7  -6  -5  -4  -3  -2  -1

>>> s[ : ]
```

>>> s[:]

Srinivas **# Here starts from 0 index and ends with available length**

>>> s[3 :]   # 'nivas'

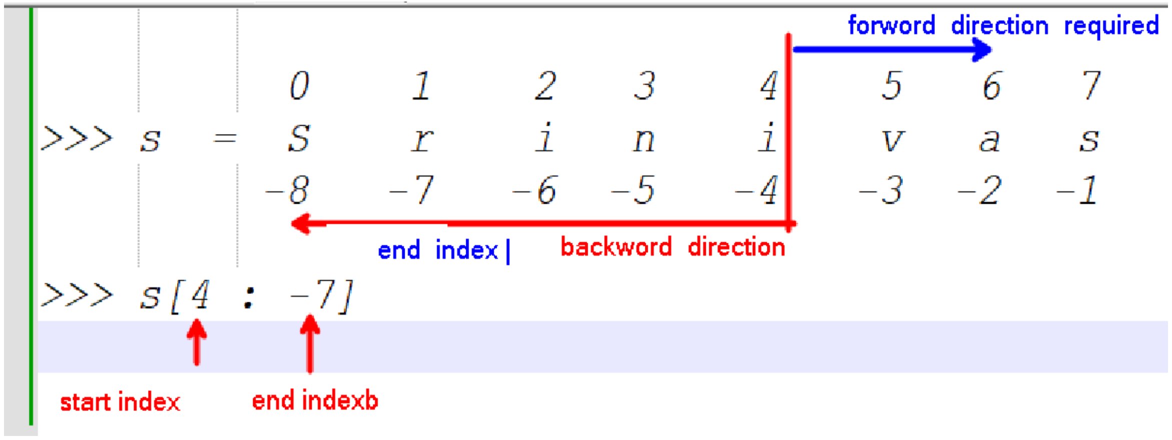


```
                0   1   2   3   4   5   6   7
>>> s  =        S   r   i   n   i   v   a   s
               -8  -7  -6  -5  -4  -3  -2  -1

>>> s[3 : ]
```

- All ways we can slicing the given string as a forward index position only otherwise it returns empty string.

```
>>> s[ 4 : -7 ]            # ' '
```



```
>>> s  =  0  1  2  3  4  5  6  7
          S  r  i  n  i  v  a  s
         -8 -7 -6 -5 -4 -3 -2 -1
>>> s[4 : -7]
```

String indexing:

To access specific value from a given string by using a given index value is called as indexing.

Syntax: **object[indexPosition]**

Accessing Values in Strings

```
>>> s[2]      # 'i'
```

Updating Strings

```
var1 = 'Hello World!'
print("Updated String :- ", var1[:6] + 'Python')
```

NOTE :

- We can access **individual characters** using "**indexing**" and a **range of characters** using "**slicing**".   s[3] , s[2:6]
- Index starts from 0. If we try to access a character out of index range will raise an **IndexError**.

```
>>> s[15]
IndexError: string index out of range
```

- The index must be an integer. We can't use float or other types, this will result into **TypeError**.

```
>>> s[1.0]
```

TypeError: string indices must be integers, not 'float'

- The index of -1 refers to the last item, -2 to the second last item and so on.
- We can access a range of items in a string by using the slicing operator (colon).

```
>>> s [ 2 : 7 ]
'iniva'
```

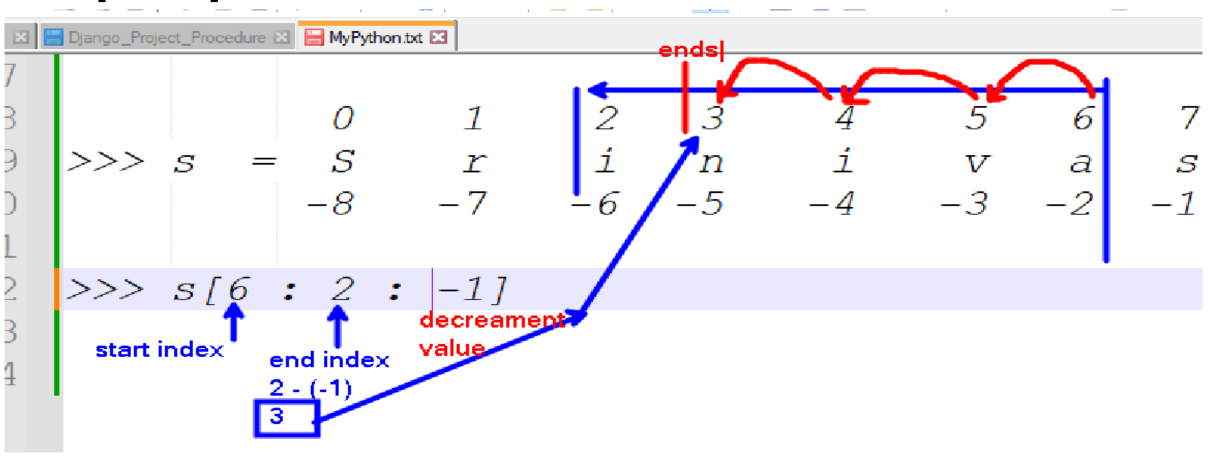
Slicing with step increment:

- Accessing every second character starting from 0 index to end index

```
>>> s[ 0 : : 2 ]       #  'Siia'
```

- Accessing every backword character starting from 4th index to start 0 index

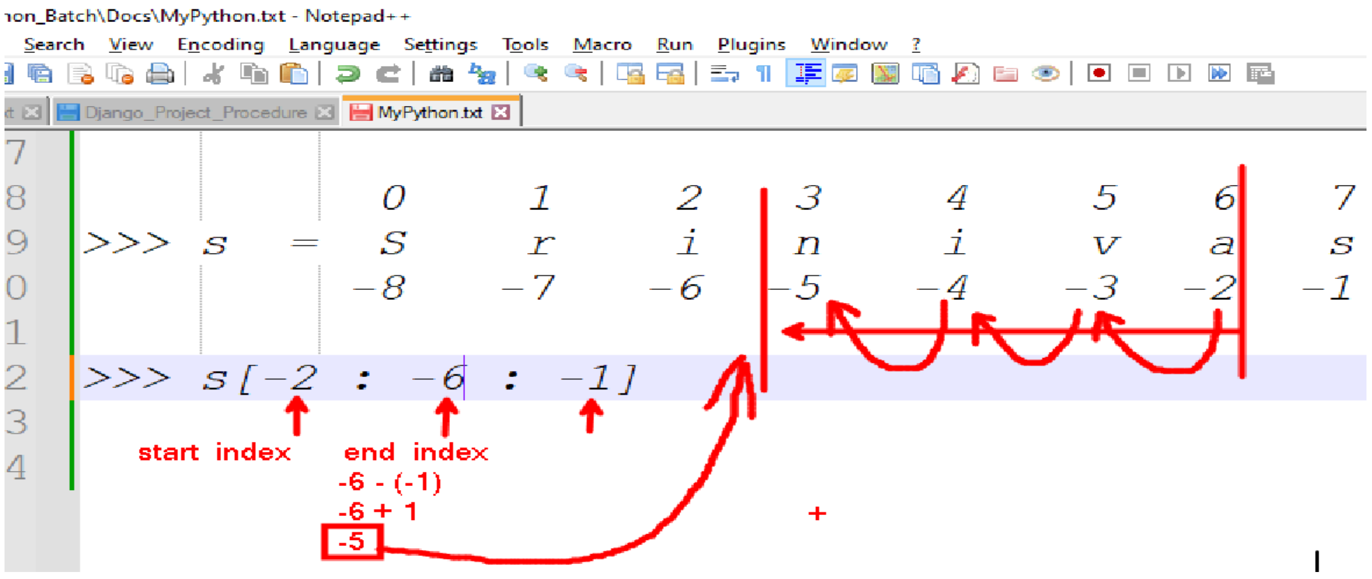
```
>>>  s[4::-1]          # 'inirS'
>>> s[6:2:-1]         #  'avin'
```



```
            0     1     2     3     4     5     6     7
>>> s  =    S     r     i     n     i     v     a     s
           -8    -7    -6    -5    -4    -3    -2    -1

>>> s[6  :  2  :  -1]
```

```
>>> s[-2:-6:-1]         #  'avin'
```

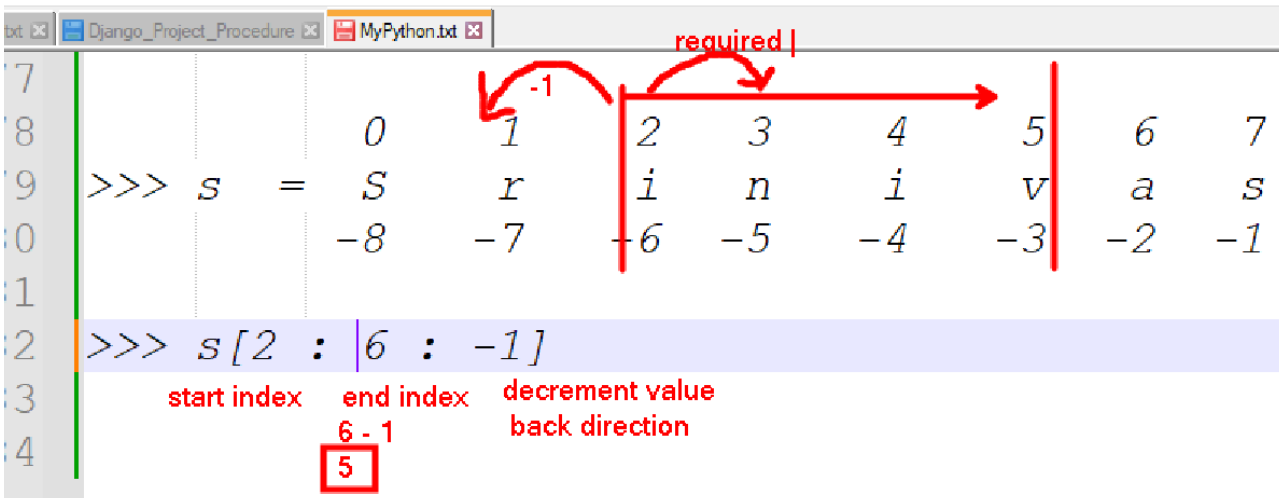


```
            0     1     2     3     4     5     6     7
>>> s  =    S     r     i     n     i     v     a     s
           -8    -7    -6    -5    -4    -3    -2    -1

>>> s[-2  :  -6  :  -1]
```

non_Batch\Docs\MyPython.txt - Notepad++
Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Django_Project_Procedure
MyPython.txt
start index
end index
-6 - (-1)
-6 + 1
-5
+

```
>>> s[ 2 : 6 : -1 ]     #  ' '
```



```
>>> s  =  S   r   i   n   i   v   a   s
          0   1   2   3   4   5   6   7
         -8  -7  -6  -5  -4  -3  -2  -1
>>> s[2 : 6 : -1]
```

Concatenation of two or more strings:

- We can concatenate two or more strings into a single one is called concatenation.
- The + operator is used in Python for concatenation.

Example:

```
>>> string1 = 'Python'
>>> string2 = 'Developer'
>>> print( 'String1 + string2 :  ',    string1 + '  ' + string2 )
```

Output : String1 + string2 : Python Developer

Multiplication of string:

- Python supports multiplying the given string into n number of times.
- The * operator can be used to repeat the string for a given number of times.

Example:

```
>>>string1 = 'Python'
>>>print(string1 * 3)
```

Output : PythonPythonPython

String Unpacking

- String unpacking allows extracting string elements automatically.
- String unpacking is the list of variables on the left has the same number of elements as the length of the string.

```
>>> str1="Python"
>>> print(str1)                  Python
>>> type(str1)                  <class 'str'>
>>> id(str1)                    23941472
>>> a,b,c,d,e,f = str1          # string unpacking
>>> print(a)                    P
>>> type(a)                     <class 'str'>
>>> print(b)                    y
>>> type(b)                     <class 'str'>
```

Membership ---->>> in

- It Returns True if a given character exists in the given string

```
>>> s = "Srinivas"
>>> 'r' in s  ----->> True
```

Membership ---->>>   not in

- Returns true if a character does not exist in the given string

```
>>> ‘S’  in  s       #  True
```