Strings

Strings

- A string is a sequence of characters in order. A character is anything you can type on the keyboard in one keystroke, like a letter, a number, or a backslash.
- 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.
- Strings can have spaces: "hello world".
- An empty string is a string that has 0 characters.
- Python strings are immutable.

String Representation

- A **string** can be represented/created in one of the following ways.

```
word1='Python'
word2='Python Programming'
word3="Python Programming"
word4="""Python is a Programming.....
         Language"""

print(word1, word2, word3, word4, sep='\n')
```

Accessing String Characters

- The characters in a string are accessed by using the index of the string.
- Each character is associated with an index. Positive integers are used to index from the left and negative integers are used to index from the right end. Only integers are allowed to be passed as index.

P	Y	T	H	O	N
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Accessing String Characters

```
w='Python'

print(w[0], w[3], w[5])
print(w[-6], w[-3], w[-1])
w[2]='p'
print(w[6])
```

Escape Sequences in Strings

```
path="c:\new\text.dat"
path1="c:\raw\book.dat"
word1="\n is new line character"
word2="\t is tab space character"
print(path)
print(path1)
print(word1)
print(word2)

path="c:\\new\\text.dat"
path1="c:\\raw\\book.dat"
word1="\\n is new line character"
word2="\\t is tab space character"
print(path)
print(path1)
print(word1)
print(word2)
```

Raw Strings

- Raw strings suppresses or ignores escape sequences. Raw strings are represented by using r or R before a string.

```
path=r"c:\new\text.dat"
path1=R"c:\raw\book.dat"
word1=r"\n is new line character"
word2=R"\t is tab space character"
print(path)
print(path1)
print(word1)
print(word2)
```

Concatenation of Strings

- Joining of two or more strings into a single one is called concatenation. The + operator does this in Python.

```
a='Python'
b='Programming'
c=a+b
d=('python'  'programming')
e='Python' 'programming'
print(c, a+b)
print(a*5)
print((a+b)*3)
print(d)
print(e)
```

String Membership Test

```
a='Python'
print('t' in a)
print('t' in 'Python')
print('T' in a)
print('th' not in a)
```

Slicing

- To get set of characters from a string, we can use the slicing method like

  variable name[start : end]

- For example, word='Hello World'

```
word[start:end]   # items start through end-1
word[start:]      # items start through the rest of the list
word[:end]        # items from the beginning through end-1
word[:]           # a copy of the whole list
```

Slicing

word='Hello World'

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Slicing Operation	Output	Description
word[0:1]	H	get one char of the word
word[0:3]	Hel	get the first three char
word[:3]	Hel	get the first three char
word[3:]	lo World	get all except first three characters
word[3:10]	lo Worl	get all except first three characters
word[-3:]	rld	get the last three char
word[:-3]	Hello Wo	get all except last three characters

Extended Slicing

- Extended slicing facilitate more options to extract characters in a string. The syntax is

  variable name[start : end: step]

- By default the step value is +1(positive) and is optional. Start and end are similar to normal slicing.

- If the step value is negative, then extraction starts from the end and prints in the reverse order.

Extended Slicing

word='Hello World'

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Slicing Operation	Output	Description
word[0:4:1]	Hell	get first four characters of the word
word[::2]	HloWrd	get the alternate characters
word[::-1]	dlroW olleH	get all characters in reverse
word[-3:-8:-1]	roW o	get the last three characters
word[6:3:-1]	W o	get all except first three characters

Updating Strings

- Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. For example,

```
>>> S = ‘hello'
>>> S[0] = 'c'                # Raises an error!
TypeError: 'str' object does not support item
    assignment
```

- We can assign a different string to the same variable. For example,

```
>>>S=“world”
>>>print (S)
world
```

Updating Strings

- We can concatenate another string to the existing string. For example,

```
>>> S =“hello”
>>>S= S + ‘world!'
# To change a string, make a new one

>>> S
 ‘helloworld!'

>>> S = S[:5] + ‘vit' + S[−1]
>>> S
‘hellovit!'
```

Deleting Strings

- We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

```
>>> S='hello'
 >>> del S[1]
TypeError: 'str' object doesn't support item deletion

>>> del S
>>> S
...
Name Error: name 'S' is not defined
```

Sample Program 1

- Write a program to find number of letters in a string or calculate the length of a string.

```
word='Python Programming'
count=0
for i in word:
    count=count+1
print("Length of the string1 : ", count)
```

Sample Program 2

- Write a program to find number of repeated letters in a string.

```
word='Python Programming'
count=0
for i in word:
    if i=='o' or 'O':
        count=count+1
print(" 'o' is repeated", count, "times")
```

Sample Program 3

- Write a program to find number of vowels in a string.

```
string=input("Enter a string: ")
count=0
for i in string:
    if i=='a' or i=='e' or i=='i' or i=='o' or i=='u':
        count=count+1
print(" No. of vowels present : ", count)
```

Sample Program 4

- Write a program to find number of words in a string.