Strings in python

Data Types in Python:

Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories:

- Text Type
- Numeric Type
- Sequence Type
- Mapping Type
- Set Types
- Boolean Type
- Binary Type

Here, string belongs to the category text type.Now let's discuss the string data type further in detail.

Strings:

A string represents a group of characters. Strings are important because most of the data that we use in daily life will be in the form of strings. For example, the names of persons, their addresses, vehicle numbers, their credit card numbers, etc. are all strings. In Python, the str data type represents a string.

Creating strings:

We can create a string in Python by assigning a group of characters to a variable. The group of characters should be enclosed inside single quotes or double quotes as:

```
>>> s1= 'Hello World'
>>> s2="Hello World"
```

There is no difference between the single quotes and double quotes while creating the Strings. Both will work in the same manner.

Length of a String:

Length of a string represents the number of characters in a string. To know the length of a string, we can use the len() function. This function gives the number of characters including spaces in the string.

Ex:

```
1  str = 'Python'
2  n = len(str)
3  print(n)
```

Output:

```
6
>
```

Indexing in Strings:

Index represents the position number. Index is written using square braces [].By specifying the position number through an index, we can refer to the individual elements of a string.For example, str[0] refers to the 0th element of the string and
str[1] refers to the 1st element of the string. Thus, str[i| can be used to refer to ith element of the string. Here, 'i' is called the string index because it is specifying the position number of the element in the string.

When we use index as a negative number, it refers to elements in reverse order. Thus str[-1] refers to the last element and str[-2] refers to the second element from the last.

Slicing the strings:

A slice represents a part or piece of a string. The format of slicing is:
stringname[start: stop: stepsize]

If 'Start' and 'stop' are not specified, then slicing is done from 0th to n-1 th elements. If step size is not written, then it is taken to be 1.
Ex1:

```
1  Str='Python'
2  print(Str[0:2:1])
```

Output:

```
Py
>
```

Ex2:

```
1  Str='Python'
2  print(Str[0:5:2])
```

Output:

```
Pto
>
```

Ex3:

```
1  Str='Python'
2  print(Str[-1::-1])
```

Output:

```
nohtyP
>
```

Repeating the Strings:

The repetition operator is denoted by '*' symbol and is useful to repeat the string for several times. For example, str * n repeats the String for n times.

Ex:

```
1  str = 'Python'
2  print(str*2)
```

Output:

```
PythonPython
>
```

Concatenation of Strings:

We can use '+' on strings to attach a string at the end of another string. This operator '+' is called addition operator when used on numbers. But, when used on strings, it is called 'concatenation' operator since it joins or concatenates the strings.

```
1  s1='Hello'
2  s2='World'
3  s3=s1+s2
4  print(s3)
```

Output:

```
HelloWorld
>
```

Checking membership:

We can check if a string or character is a member of another string or not using ‘in’ and ‘not in’ operators. The ‘in’ operator returns True if the string or character is found in the main string. It returns False if the string or character is not found in the main String. The ‘not in’ operator returns False if a string or character is not found in the main string, otherwise True.

The operators ‘in’ and ‘not in’ make case sensitive comparisons. It means these operators, consider the upper case and lower case letters or strings differently while comparing strings.
Ex:

```
1  str = input('Enter main string: ')
2  sub = input('Enter substring: ')
3  if sub in str:
4      print(sub+' is found in main string')
5  else:
6      print(sub+' is not found in the main string')
7
```

Output:

```
Enter main string: This is Python
Enter substring: Python
Python is found in main string
>
```

Comparing Strings:

We can use the relational operators like >, >=, <, <=, == or != operators to compare strings. They return Boolean values, i.e. either True or False depending on the string being compared.
Ex:

```
s1='Toy'
s2='Boy'
if(s1==s2):
    print('Both are same')
else:
    print('Not same')
```

Output:

```
Not same
```

Removing Spaces from a String:

A space is also considered as a character inside a string. Sometimes, the unnecessary spaces in a string will lead to wrong results. For example, a person typed his name ‘Robert ’ (observe two spaces at the end of the string) instead of typing ‘Robert’. If we compare these two strings using ‘==’ operator as:

```
if 'Gru '=='Gru':
    print('Welcome')
else:
    print('Name not found')
```

The output will be ‘Name not found’. In this way, spaces may lead to wrong results. Hence such spaces should be removed from the strings before they are compared. This is possible using rstrip(), lstrip() and strip() methods. The rstrip() method removes the spaces which are at the right side of the string. The lstrip() method removes spaces which are at the left side of the string. strip() method removes spaces from both the sides of the strings. These methods do not remove spaces which are in the middle of the string. Ex:

```
txt=' Gru '
x=txt.strip()
print(x)
```

Output:

```
Gru
```

Finding Substrings:

The find(). rfind(), index() and rindex() methods are useful to locate sub strings in a string.These methods return the location of the first occurrence of the sub string in the main string.The find() and index() methods search for the sub string from the beginning of the main string.The rfind() and rindex() methods search for the substring from right to left i.e. in backward order.

mainstring.find(substring, beginning, ending)

Counting Substrings in a String:

The method count() is available to count the number of occurrences of a sub stil main string. The format of this method is:

stringname. count (substring)

This returns an integer number that represents how many times the substring is found in the main string.We can limit our search by specifying beginning and ending positions in the count() method so that the substring position is counted only in that range. Hence, the other form of count() method is:

stringname. count (substring, beg, end)

Strings are Immutable:

An immutable object is an object whose content cannot be changed. On the other hand, a a mutable object is an object whose content can be changed as and when required. In Python, numbers, strings and tuples are immutable. Lists, sets, dictionaries are mutable objects.

Replacing a String with another String:

The replace() method is useful to replace a substring in a string with another substring. The format of using this method is:

stringname.replace(old, new)

This will replace all the occurrences of ‘old’ substring with ‘new’ sub string in the main string.

Ex:

```
str = 'That is a car'
s1 = 'car'
s2 = 'Bus'
str1 = str.replace(s1,s2)
print(str1)
```

Output:

```
That is a Bus
```

Splitting and Joining Strings:

The split() method is used to break a string into pieces. These pieces are returned in a list. For example, to break the string ‘str’ where a comma (,) is found, we can write:

str.split(‘,’)
Observe the comma inside the parentheses. It is called a separator that represents where to separate or cut the string.

When a group of strings are given, it is possible to join them all and make a single string. For this purpose, we can use join() method as:

separator. join(str)

where, the ‘separator’ represents the character to be used between the strings in the output. ‘str’ represents a tuple or list of strings.

Ex:

We want to join the three strings and form a single string. Also, we want to use hyphen (-) between the three strings in the output. The join() method can be written as:

```
str=('one', 'two', 'three')
```

Output:

```
one-two-three
```

Changing Case of a String:

Python offers 4 methods that are useful to change the case of a string. They are upper(), lower(), Swapcase(), title(). The upper() method is used to convert all the characters of a string into uppercase or capital letters. The lower() method converts the string into lowercase or into small letters. The swapcase() method converts the capital letters into small letters and vice versa. The title() method converts the string such that each word in the string will start with a capital letter and remaining will be small letters.

Ex:

```
str = 'Hello World'
print(str.upper())
```

Output:

```
HELLO WORLD
```

Checking Starting and Ending of a String:

The startswith() method is useful to know whether a string is Starting with a substring or not.The way to use this method is:

```
1  str.startswith(substring)
```

When the substring is found in the main string ‘str’, this method returns True. If the string is not found, it returns False. Similarly to check the ending of a string, we use endswith() method. It returns True if the string ends with the specified substring, otherwise it returns False.

```
2  str.endswith(substring)
```

String Testing Methods:

There are several methods to test the nature of characters in a string. These methods return either True or False.For example, if a string has only numeric digits, then isDigit() method returns True. These methods can also be applied to individual characters.

Method	Description
isalnum()	This method returns True if all characters in the string are alphanumeric (A to Z, a to z, 0 to 9) and there is at least one character otherwise it returns False.
isalpha()	Returns True if the string has at least one character and all characters are alphabetic (A to Z and a to 2); otherwise, it returns False.
isDigit()	Returns True if the string contains only numeric digits (0 to 9) and False otherwise.
islower()	Returns True if the string contains at least one letter and all characters are in lower case; otherwise, it returns False.
isupper()	Returns True if the string contains at least one letter and all characters are in upper case; otherwise, it returns False.
istitle()	Returns True if each word of the string starts with a capital letter and there is at least one character in the string; otherwise, it returns False.
isspace()	Returns True if the string contains only spaces; otherwise, it returns False.

Formatting the Strings:

Formatting a string means presenting the string in a clearly understandable manner. The format() method is used to format strings. This method is used as:

‘format string with replacement fields’ . format(values)

We should first understand the meaning of the first attribute, i.e. format string with replacement fields’. The replacement fields are denoted by curly braces { } that contain names or indexes. These names or indexes represent the order of the values. For example, let’s take an employee details like id number, name and salary in 3 variables ‘id’, ‘name’ and ‘sal’.

```
id=10
name='Lucy'
sal=10000.25
```

We want to create a format string by the name ‘str’ to display these 3 .These 3 values of variables should be mentioned inside the format() method as format(id,name,sal). The total format string can be written as:

```
str ='{},{},{}'.format(id,name,sal)
```

This string contains 3 replacement fields. The first field {} is replaced by ‘id’ value and the second field {} is replaced by the ‘name’ value and the third field is replaced by ‘sal’ value. So, if we display this string using print() method

```
print(str)
```

Output:

```
10,Lucy,10000.25
```

Working with Characters:

Characters are nothing but the individual elements of a string. As we know a string may contain 1 or more characters. When the programmer is interested in working with characters, he has to accept a string and then retrieve the characters from the string using indexing or slicing. Ex:

To retrieve the 0th character, we can write str[0] and to retrieve the 1st character, we can write str[1].

```
str='Apple'
```

```
2  ch=str[0]
3  print(ch)
```

Output:

A

Sorting Strings:

We can sort a group of strings into alphabetical order using the sort() method and sorted() function.The sort() method is used in the following way:
str.sort()

Here, ‘str’ represents an array that contains a group of strings. When the sort() method is used, it will sort the original array, i.e. ‘str’. So, the original order of strings will be lost and we will have only one sorted array. To retain the original array even after sorting, we can use sorted() function as:

```
1  str1 = sorted(str)
```

Ex:

```
# sorting a group of strings
# take an empty array
str = []
n=int(input('How many strings? '))
# append strings to str array
for i in range(n):
    print('Enter string: ', end='')
    str.append(input())
# sort the array
# str.sort()
str1 = sorted(str)
# display the sorted array
print('Sorted list: ')
for i in str1:
    print(i)
```

Output:

```
How many strings? 3
Enter string: Hyd
Enter string: Mumbai
Enter string: Delhi
Sorted list:
Delhi
Hyd
Mumbai
```

Searching in the Strings:

The easiest way to search for a string in a group of ‘n’ strings is by using sequential search or linear search technique.This is done by comparing the searching string ‘s’ with every string in the group. For this purpose, we can use a for loop that iterates from 0th to n-1th string in the group.By comparing the searching string ‘s’ with every one of these strings, we can decide whether ‘s’ is available in the group or not. The logic looks like this:

```
>>> for i in range(len(str)):
        if s==str[i]:
```

Ex:

```
# sorting a group of strings
# take an empty array
str = []
n = int(input('How many strings? '))
# append strings to str array
for i in range(n):
    print('Enter string: ', end='')
    str.append(input())
# sort the array
# str.sort
s=input('Enter string to search: ')
flag=False
for i in range(len(str)):
    if s==str[i]:
        print('Found at: ',i+1)
        flag=True
if flag==False:
    print('Not Found')
```

Output:

```
How many strings? 3
Enter string: Hyderabad
Enter string: Banglore
Enter string: Chennai
Enter string to search: Hyderabad
Found at:  1
```

Finding Number of Characters and Words:

We have the len() function that returns the number of characters in a string. Suppose we want to find the length of a string without using len() function, we can use a for loop as: >>> i=0

```
>>> for s in str:
        i+=1
```

This for loop repeats once for each character of the string ‘str’. So, if the string has 10 characters, the for loop repeats for 10 times. Hence, by counting the number of repetitions, we can find the number of characters. This is done by simply incrementing a counting variable ‘i’ inside the loop.

Ex:

```
# to find no. of characters in a string
str = input('Enter a string: ')
i=0
for s in str:
    print(str[i], end='')
    i=i+1
print('\n no. of characters: ', i)
```

Output:

```
Enter a string: Hello World
Hello World
 no. of characters:  11
```

Inserting Substring into a String:

Let's take a string with some characters. In the middle of the string, we want to insert a substring. This is not possible because of the immutable nature of strings. But, we can find a way for this problem. Let's assume the main string is 'str' and the sub string is 'sub'. After inserting 'sub' into 'str', the total string will be 'str1'. To represent this total string, we will declare an empty list 'str1' as:

str1=[]

If n is the position where the sub string to be inserted, we will append the first n-1 characters from str into str1. Then the entire sub string will be appended to str1. In the final step, we will append the remaining characters (from n till the end) from str to strl. Thus, the total string will be available in str1 as a list. Figure 8.3 shows the insertion of a substring in a particular position into a main string.

Since a list contains characters as individual elements, we have to convert the list into a String format so that we will have a continuous flow of characters. See the difference between elements of a list and of a string:
['H','e','l','l','o',' ','M','y','W','o','r','l','d']

Hello MyWorld

Above, the first line represents a list and the second line represents & String, We need the result in the form of a string, hence we have to convert the list into a String. For this purpose, we can use join() method with empty string as separator as:

str1=' '.join(str1)
Since the separator is an empty string, the elements of str1 will be joined without gaps in between and we will have the final string into 'str1'.

Ex:

```
str=input('Enter a string: ')
sub=input('Enter a sub string: ')
n=int(input('Enter position no: '))
# decrease n by 1 to insert in correct position
n-=1
# find the number of characters in str, sub
l1=len(str)
l2=len(sub)
# take another string as a list
str1 = []
# append o to n-1 characters from str to stri
for i in range(n):
    str1.append(str[i])
# append sub string to strl
for i in range(l2):
    str1.append(sub[i])
#append the remaining characters from str to strl
for i in range(n,l1):
    str1.append(str[i])
# convert the individual characters of list into
# a string using join() method with empty string as separator
str1=''.join(str1)
# display the total string
print(str1)
```

Output:

```
Enter a string: Hello
Enter a sub string: h
Enter position no: 2
Hhello
```