Strings

A string in Python is a sequence of characters. For Python to recognize a sequence of characters, like *hello*, as a string, it must be enclosed in quotes. The string can be enclosed in single or double quotes.

Example:

```
>>> ‘‘Hello World’’
‘Hello World’
>>> ‘Hello World’
‘Hello World’
```

Note: Python interpreter displays the string with single quotes.

Having two sorts of quotes can be useful in certain circumstances. If you want text itself to include quotes of one type can define it surrounded by other type.

Example:

```
>>> print(‘‘This is ‘Use of Single Quotes’.’’)
This is ‘Use of Sungle Quotes’.
>>> print(‘This is ‘‘Use of Double Quotes’’.’)
This is ‘‘Use of Double Quotes’’.
```

Triple Quoted String Literals

Strings delimited by single or double quote character are required to lie within a single line. It is sometimes convenient to have a multi-line string, which can be delimited with triple quotes

Example:

```
>>> str1 = ’’’Hello
...  "Good Morning!!!"
...  Have a cup of coffee.’’’
>>> print(str1)
Hello
"Good Morning!!!"
Have a cup of coffee.
```

String Operations

Concatenation

The plus operation with strings means concatenate the strings. Python looks at the type of operands before deciding what operation is associated with the +.

The plus (+) sign is the string *concatenation operator* which is used to combine number of strings and returns the new string.

Example:

```
>>> str1 = "Hello World"
>>> print(str1 + " 'Can Be Joined'")   # Prints concatenated string
Hello World 'Can Be Joined'
```

Repetition Operator

The asterisk(*) sign is the *repetition operator* which is use to repeat the string as many times as specified.

Example:

```
>>> str1 = "Hello World"
>>> print(str1 * 3)   # Prints string 3 times
Hello WorldHello WorldHello World
```

Accessing Characters In String

We can access a character from the string by specifying the index of the character. Index starts from '0' indicates the beginning of the string and working their way from -1 at the end.

For Example:

```
>>> str1 = "Strings In Python"
>>> print(str1)       # Prints complete string
Strings In Python
>>> print(str1[0])    # Prints first character of the string
```

```
S
>>> print(str1[5])    # Prints sixth character of the string
g
>>> print(str1[-1])   # Prints the last character of the string
n
>>> print(str1[-3])   # Prints the third character from the last
h
>>> print(str1[-8])   # Prints the eigth character from the last
n
```

Slicing the String

We can access the subsets of a string using slice operator ([:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

For Example:

```
>>> str2 = "Slicing the String"
>>> print(str2[4:])   # Prints string starting from the 5th character
ing the String
>>> print(str2[:4])   # Prints the first four characters
Slic
>>> print(str2[2:8])  # Prints characters starting from 3rd to 7th
icing
>>> printt(str2[4:-3]) # Prints characters from 5th to 3rd character from last
ing the Str
```

String Methods

string methods performs operations on strings.

Strings have their own set of functions. In this section we will go through few of them:

len()

The *len()* function returns the length of a string as an integer. `len("String")`

Example

```
>>> len("Hello World") # This returns the value as 11.
>>>
>>> name = "TalentSprint"
>>> len(name) # 12
```

lower() Converts all uppercase letters in string to lowercase and return the new string.

```
>>> s1 = "PYTHON"
>>> s1.lower()
'python'
```

upper() Converts lowercase letters in string to uppercase and return the new string.

```
>>> s2 = "python"
>>> s2 = name.upper()
>>> print(s2)
'PYTHON'
```

replace() The function *replace()* returns a copy of the string with all occurrences of sub-string old replaced by new.

Syntax:

```
str.replace(old, new)
old - This is the old substring to be replaced
new - This is new substring, which would replace old substring.
```

Example:

```
>>> str = "This is example for replace function"
>>> str.replace('is', 'was')
'Thwas was example for replace function'
```

split() The method *split()* is used to split on the whitespaces (blanks, newline) and returns the list of sub strings as items.

Example:

```
>>> str1 = "Engineering Student"
>>> str1.split()
['Engineering', 'Student']
>>> str1.split("i")
['Eng', 'neer', 'ng Student']
```

strip() By using *strip()* function, It returns a copy of the string with the leading and trailing characters removed.

Example: As it treats the argument as a set of characters. In this example, we specify all digits, and some punctuation chars.

```
>>> value = "543210=Data,123"
# strip all digits
# Also remove equals sign and comma.
>>> result = value.strip("0123456789=,")
>>> print(result)
Data
```