Python String

Till now, we have discussed numbers as the standard data types in python. In this section of the tutorial, we will discuss the most popular data type in python i.e., string.

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Consider the following example in python to create a string.

1. str = "Hi Python !"

Here, if we check the type of the variable str using a python script

1. **print**(type(str)), then it will **print** string (str).

In python, strings are treated as the sequence of strings which means that python doesn't support the character data type instead a single character written as 'p' is treated as the string of length 1.

Strings indexing and splitting

Like other languages, the indexing of the python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.

As shown in python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in python to access the substring. Consider the following example.

Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'python' is given, then str[1:3] will always include str[1] = 'p', str[2] = 'y', str[3] = 't' and nothing else.

Reassigning strings

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with a new string since its content can not be partially replaced. Strings are immutable in python.

Consider the following example.

Example 1

1. str = "HELLO"
2. str[0] = "h"
3. **print**(str)
   Output:
   Traceback (most recent call last):
   File "12.py", line 2, in <module>

```
    str[0] = "h";
```
TypeError: 'str' object does not support item assignment

However, in example 1, the string str can be completely assigned to a new content as specified in the following example.

Example 2

1. str = "HELLO"
2. **print**(str)
3. str = "hello"
4. **print**(str)

Output:

HELLO
hello

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Example

Consider the following example to understand the real use of Python operators.

1. str = "Hello"
2. str1 = " world"
3. **print**(str*3) # prints HelloHelloHello
4. **print**(str+str1)# prints Hello world
5. **print**(str[4]) # prints o
6. **print**(str[2:4]); # prints ll
7. **print**('w' **in** str) # prints false as w is not present in str
8. **print**('wo' **not in** str1) # prints false as wo is present in str1.
9. **print**(r'C://python37') # prints C://python37 as it is written

10. **print**("The string str : %s"%(str)) # prints The string str : Hello

Output:

HelloHelloHello
Hello world
o
ll
False
False
C://python37
The string str : Hello

Python Formatting operator

Python allows us to use the format specifiers used in C's printf statement. The format specifiers in python are treated in the same way as they are treated in C. However, Python provides an additional operator % which is used as an interface between the format specifiers and their values. In other words, we can say that it binds the format specifiers to the values.

Consider the following example.

1. Integer = 10;
2. Float = 1.290
3. String = "Ayush"
4. **print**("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s"%(Integer,Float,String));

Output:

Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Ayush

Built-in String functions

Python provides various in-built functions that are used for string handling. Many String fun

Functions	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3
casefold()	It returns a version of s suitable for case-less comparisons.
center(width ,fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
count(string,begin,end)	It counts the number of occurrences of a substring in a String between begin and end index.
decode(encoding = 'UTF8', errors = 'strict')	Decodes the string using codec registered for encoding.

encode()	Encode S using the codec registered for encoding. Default encoding is 'utf-8'.
endswith(suffix ,begin=0,end=len(string))	It returns a Boolean value if the string terminates with given suffix between begin and end.
expandtabs(tabsize = 8)	It defines tabs in string to multiple spaces. The default space value is 8.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
format(value)	It returns a formatted version of S, using the passed value.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found. It works same as find() method.
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false.
isalpha()	It returns true if all the characters are alphabets and there is at least one character, otherwise False.
isdecimal()	It returns true if all the characters of the string are decimals.
isdigit()	It returns true if all the characters are digits and there is at least one character, otherwise False.
isidentifier()	It returns true if the string is the valid identifier.
islower()	It returns true if the characters of a string are in lower case, otherwise false.
isnumeric()	It returns true if the string contains only numeric characters.
isprintable()	It returns true if all the characters of s are printable or s is empty, false otherwise.
isupper()	It returns false if characters of a string are in Upper case, otherwise False.
isspace()	It returns true if the characters of a string are white-space, otherwise false.
istitle()	It returns true if the string is titled properly and false otherwise. A title string is the one in which the first character is upper-case whereas the other characters are lower-case.
isupper()	It returns true if all the characters of the string(if exists) is true otherwise it returns false.
join(seq)	It merges the strings representation of the given sequence.
len(string)	It returns the length of a string.
ljust(width[,fillchar])	It returns the space padded strings with the original string left justified to the given width.
lower()	It converts all the characters of a string to Lower case.
lstrip()	It removes all leading whitespaces of a string and can also be used to remove particular character from leading.
partition()	It searches for the separator sep in S, and returns the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.
maketrans()	It returns a translation table to be used in translate function.

replace(old,new[,count])	It replaces the old sequence of characters with the new sequence. The max characters are replaced if max is given.
rfind(str,beg=0,end=len(str))	It is similar to find but it traverses the string in backward direction.
rindex(str,beg=0,end=len(str))	It is same as index but it traverses the string in backward direction.
rjust(width,[,fillchar])	Returns a space padded string having original string right justified to the number of characters specified.
rstrip()	It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.
rsplit(sep=None, maxsplit = -1)	It is same as split() but it processes the string from the backward direction. It returns the list of words in the string. If Separator is not specified then the string splits according to the white-space.
split(str,num=string.count(str))	Splits the string according to the delimiter str. The string splits according to the space if the delimiter is not provided. It returns the list of substring concatenated with the delimiter.
splitlines(num=string.count('\n'))	It returns the list of strings at each line with newline removed.
startswith(str,beg=0,end=len(str))	It returns a Boolean value if the string starts with given str between begin and end.
strip([chars])	It is used to perform lstrip() and rstrip() on the string.
swapcase()	It inverts case of all characters in a string.
title()	It is used to convert the string into the title-case i.e., The string **meEruT** will be converted to Meerut.
translate(table,deletechars = '')	It translates the string according to the translation table passed in the function .
upper()	It converts all the characters of a string to Upper Case.
zfill(width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
rpartition()	