Python Programming Basics

String Functions in Python

Contents

Introduction to Strings

Any value written within a pair of single quote or double quotes in Python is treated as a String.

'A character string using single quotes'

"A character string using double quotes"

Internally Python stores every string within single quotes, even when you create them with double quote.

We can insert single quotes in a string with double quotes, and vice versa:

"Welcome to 'Ask Analytics'"

'Welcome to "Ask Analytics"'

We cannot insert single quotes in a string with single quotes, neither we can insert double quotes in a string with double quotes:

"Welcome to "Ask" Analytics"

'Welcome to 'Ask' Analytics'

Concatenating Strings

```
a = "Hello"
b = 'How'
c = "are you?"

a+" "+b+" "+c
'Hello How are you?'

a+"-"+b+"-"+c
'Hello-How-are you?'

["y" + str(i) for i in range(1,5)]
['y1', 'y2', 'y3', 'y4']
```

In Python we can concatenate strings by using **+** and specify separator in between

In this case it's "**-**".

single character "**y**" is pasted with the sequence 1:4

Concatenating Strings – join()

- **`join()`** is called on a string with argument as a list value and returns a single string value which is a concatenation of each string in the argument list.

```
",".join(["orange", "blue", "green"])
'orange,blue,green'
```

Notice that **join()** is called on "," (can be customised) which is inserted between each string of the list..

- **`split()`** is called on a string value and returns a list of strings

```
"split does the opposite of join".split()
['split', 'does', 'the', 'opposite', 'of', 'join']
```

By default, it splits wherever the whitespace characters are found. You can specify a delimiter string to the **split()** function to split upon.

Formatting Strings

String Formatting Operator : %

- Unique to strings
- C's printf() like facility
- Formatting specifier is %

List of symbols which can be used along with %:

```
print("Name is %s and score is %d" %('Sana', 44))
```

```
Name is Sana and score is 44
```

```
print("Name is %s and grade is %c" %('Sana', 'A'))
```

```
Name is Sana and grade is A
```

%s gives string conversion via str() prior to formatting.
%d returns signed decimal integer.
%c returns character.

Formatting Strings - format()

String Formatting Function: format()

- Formatting specifier is { }

```
"This is a {}".format("new formatting style")
'This is a new formatting style'
```

- You can also use the positional index. This allows rearranging the order of display with no change in arguments.

```
"Learn {0} on {1}.".format("Python", "Sanaitics Kit")
'Learn Python on Sanaitics Kit.'
```

Extracting and Replacing

One common task when working with strings is the extraction and replacement of some characters. In Python extraction is same as subsetting. It can be done by simply using [ ] .

```
# Extract ‘THE’
a = "WELCOME TO THE WORLD OF PYTHON"
a[11:14]
```

```
'THE'
```

Characters from 12th to 14th position are extracted

Replacing Substrings – replace()

Within a string, if you want to replace one substring with another:

- Use **`replace()`** to replace the first instance of a substring

  `replace``(old,new,string)`

```
# Using replace()
string = "She is a data scientist. She works with an MNC"
string.replace("She","Sharon")
```

```
'Sharon is a data scientist. Sharon works with an MNC'
```

replace() finds the first instance of the ‘She’ within string and replaces it with ‘Sharon’

Replacing Substrings – replace()

```
# Replace a string with its first 3 letters
Location = ["Mumbai","Delhi","Mumbai","Kolkata","Delhi"]
Location = [word.replace(word,word[0:3]) for word in Location]
Location
```

```
['Mum', 'Del', 'Mum', 'Kol', 'Del']
```

- Here we are replacing ‘Mumbai’ with ‘Mum’, ‘Delhi’ with ‘Del’ and ‘Kolkata’ with ‘Kol’.
- **Location** is a vector of strings.
- **replace()** will replace a string with the new substring.

Splitting a String - split()

Besides the tasks of extracting and replacing substrings, another common tasks is splitting a string based on a pattern. In Python we have a function **`split()`** which splits the elements of a character vector into substrings.

If we want to break a string into individual components (.i.e. words), we can use **`split().`**

```
sentence = "break a string into individual components"
sentence.split()
```

```
['break', 'a', 'string', 'into', 'individual', 'components']
```

split() is a regular expression pattern used for splitting

```
# split each date component
```

```
dates = ["12-10-2014","01-05-2000","26-06-2015"]
dates = [date.split('-') for date in dates]
dates
```

```
['12 10 2014', '01 05 2000', '26 06 2015']
```

Here, date components joined by a dash “-”, are split

Other Basic String Manipulation Functions

```
#Converts any lower case characters into upper case
[x.upper() for x in ["ASK ANAlytics","data SCience"]]
['ASK ANALYTICS', 'DATA SCIENCE']
```

```
#Converts any upper case characters into lower case
[x.lower() for x in ["ASK ANAlytics","data SCience"]]
['ask analytics', 'data science']
```

```
#casefold function which is similar to lower function.
("all characters in LOWER case").casefold()
'all characters in lower case'
```

It converts all characters to lower case

```
#replace function which replaces a character
("This is a string").replace('a', 'A')
'This is A string'
```

replace() function replaces characters with specified character.

Quick Recap

In this session, we learnt how to manipulate strings with functions in base Python.

Here is the quick recap:

String Manipulation Functions

- `+` concatenates character strings
- `join()` concatenates list of strings with specified seperator
- `format()` formats numbers and strings to a specified style
- `replace()` replaces all instances of a substring
- `split()` which splits the elements of a character vector into substrings
- `upper()` Converts any upper case characters into lower case
- `lower()` Converts any lower case characters into upper case
- `casefold()` Converts upper case characters into lower case
.