session_one___02_strings

September 18, 2019

1 Strings

Strings are sequence of characters enclosed within single or double quotes.

"apple", "mango", 'apple is a fruit.', are all examples of strings in Python.

To create a string in Python, we just need to enclose a sequence of character(s) within sigle or double quotes.

[1]:
```
# Single word
"string"
```

[1]:
```
'string'
```

[2]:
```
# Phrase
"This is a string"
```

[2]:
```
'This is a string'
```

[3]:
```
'This is also a string but in single quotes!'
```

[3]:
```
'This is also a string but in single quotes!'
```

[4]:
```
'It's a string.'
```

```
      File "<ipython-input-4-7dab2ab4da3f>", line 1
    'It's a string.'
        ^
SyntaxError: invalid syntax
```

The reason for the error above is because the single quote in `It's` stopped the string. You can use combinations of double and single quotes to get the complete statement.

[5]:
```
"It's a string."
```

[5]: "It's a string."

To print a string in python (to the standard output i.e. your laptop screen/monitor), use the `print` function

We are going to discuss what a function is in the comming sessions. For now, just have a look at the examples below to understand how the `print` function is used.

You may know this function from the `Hello World` program that we wrote in the very beginning!

[6]:
```
print("This is a string!")
```

```
This is a string!
```

[7]:
```
print('This is also a string but in single quotes!')
```

```
This is also a string but in single quotes!
```

[8]:
```
print('string one')
print('string two')
print('use \n to print a new line')
print('\n')
print('see what I mean ?')
```

```
string one
string two
use
 to print a new line


see what I mean ?
```

You can also assign strings to variables just like we did with numbers.

[9]:
```
string_one = 'This is a string.'
```

[10]:
```
print(string_one)
```

```
This is a string.
```

`len()`: Python's in-built function is used to check the length of strings. It counts and reports the number of characters in a string including whitespaces and special characters.

[11]:
```
len(string_one)
```

[11]: 17

[12]:
```
len('apple')
```

[12]: 5

1.1 String Indexing

We know that strings are nothing but sequence of characters in Python. So, there must be some method of calling some parts of those sequences.

Here is where 'indexing' comes into play.

Every character in a string is assigned a numerical value as per its position in the string. The first character is assigned the integer 0, the second character is assigned the integer 1, the third character is assigned the integer 1, and so on.

Consider the string `"apple"`, the assignment of numerical value would be:

```
a -> 0
p -> 1
p -> 2
l -> 3
e -> 4
```

We can refer to a character present in the string by using the corresponding numerical value and a pair of square brackets - `[]`.

```
[13]: # Here, we grab the third character of the word "apple".
      # Note that the "indexing" starts with 0 and not 1. That's why we are using the␣
       ↪number '2' (to
      # grab the third character) within the square brackets instead of '3'.
      'apple'[2]
```

```
[13]: 'p'
```

You can also use square brackets with variables of type `string`.

Actually, this is more practical and it is how it is used in real life programming.

```
[14]: s = 'apple'
```

```
[15]: s[0]
```

```
[15]: 'a'
```

```
[16]: s[1]
```

```
[16]: 'p'
```

```
[17]: s[4]
```

```
[17]: 'e'
```

1.2 String Slicing

We can also grab subsections of strings using `:`.

This is how it works:

`string_variable_name[first:last:step]`

where `first` stands for the index from where we want to begin grabbing the characters,

`last` stands for the index upto which we want to grab the characters,

`step` stands for the step size in which we want to grab the characters.

Note that `first` is included while `last` is not included while slicing/grabbing!

A bunch of examples below will help you understand all this.

[18]:
```
s = 'supercalifragilisticexpialidocious'
```

[19]:
```
# We start grabbing the characters from index - 1 i.e. the second characters␣
↪(remember that indexing
# starts at 0 ?) upto index 9 (remeber that 'last' is not included while␣
↪slicing/grabbing)
# with a step size of 1
s[1:10:1]
```

[19]:
```
'upercalif'
```

Actually, the default value of step is 1. So, we can skip the `step` argument.

Have a look below.

[20]:
```
s[1:10]
```

[20]:
```
'upercalif'
```

[21]:
```
s[2:12]
```

[21]:
```
'percalifra'
```

We can also skip the `last` argument as well. In that case, the section of the string starting from the `first` index till the end of the string will be grabbed.

[22]:
```
s[2:]
```

[22]:
```
'percalifragilisticexpialidocious'
```

We can also skip the `first` argument as well. In that case, the section of the string from the beginning till the `last` index of the string will be grabbed.

Or in other words, first `first` characters will be grabbed!

```
[23]: # Grabbing first 2 characters
      s[:2]
```

```
[23]: 'su'
```

```
[24]: # Grabbing first 10 characters
      s[:10]
```

```
[24]: 'supercalif'
```

Python also supports negative indexing!

`-1` means first character from the last,

`-2` means second character from the last, and so on.

```
[25]: s[-1]
```

```
[25]: 's'
```

```
[26]: s[-2]
```

```
[26]: 'u'
```

```
[27]: # Grab everything but the last letter
      s[:-1]
```

```
[27]: 'supercalifragilisticexpialidociou'
```

```
[28]: # Grab the whole string
      s[:]
```

```
[28]: 'supercalifragilisticexpialidocious'
```

```
[29]: # Grab the whole string but in step size of 1
      s[::1]
```

```
[29]: 'supercalifragilisticexpialidocious'
```

```
[30]: # Grab the whole string but in step size of 2
      s[::2]
```

```
[30]: 'sprairglsiepaioiu'
```

```
[31]: # Reverse a string?
      # Grab the whole string but take 'step' in reverse direction!
      s[::-1]
```

```
[31]: 'suoicodilaipxecitsiligarfilacrepus'
```

1.3 String Properties

1. Strings are immutable i.e. you cannot do something like this:

```
[32]: s[0] = 'k'
```

```
                                                                             ␣
 ↪---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call␣
 ↪last)

        <ipython-input-32-97a0b231e717> in <module>()
    ----> 1 s[0] = 'k'


        TypeError: 'str' object does not support item assignment
```

Notice how the error tells us directly what we can't do, change the item assignment!

---

2. Something we can do is concatenate strings!

```
[33]: s
```

```
[33]: 'supercalifragilisticexpialidocious'
```

```
[34]: # concatenating!
      s + ' is actually a word!'
```

```
[34]: 'supercalifragilisticexpialidocious is actually a word!'
```

```
[35]: s + s
```

```
[35]: 'supercalifragilisticexpialidocioussupercalifragilisticexpialidocious'
```

```
[36]: s + " " + s
```

```
[36]: 'supercalifragilisticexpialidocious supercalifragilisticexpialidocious'
```

---

3. We can use the multiplication symbol to create repititions!

```
[37]: s * 4
```

```
[37]: 'supercalifragilisticexpialidocioussupercalifragilisticexpialidocioussupercalifr
      agilisticexpialidocioussupercalifragilisticexpialidocious'
```

```
[38]: t = 'cat'
```

```
[39]: t * 5
```

```
[39]: 'catcatcatcatcat'
```

1.3.1 Some built-in methods for strings in Python

1. `len()`

```
[40]: t
```

```
[40]: 'cat'
```

```
[41]: len(t)
```

```
[41]: 3
```

2. lower(): convert the characters in a string to lowercase.

```
[42]: t = 'CaT'
```

```
[43]: t
```

```
[43]: 'CaT'
```

```
[44]: t.lower()
```

```
[44]: 'cat'
```

This actually does not change `t`. We will need to reassign `t` to keep the changes.

```
[45]: t
```

```
[45]: 'CaT'
```

```
[46]: t = t.lower()
```

```
[47]: t
```

```
[47]: 'cat'
```

3. upper()

```
[48]: t = t.upper()
```

```
[49]: t
```

```
[49]: 'CAT'
```

That is all for strings as of now!

We will learn more about strings as the sessions continue.

Till then, Happy Coding!