Python Programming

by Narendra Allam

Chapter 2

Strings

Topics Covering

- Strings
  - Commenting in python
  - Define a string - Multiple quotes and Multiple lines
  - String functions
  - String slicing - start, end & step
  - Negative indexing
  - Scalar multiplication
- Exercise Programs

Commenting in python

Comments are used in the code for describing the logic. This helps the new developers, understanding code better.

In python,

- Hash (#) is uded for single line comments
- Triple single quotes (''' ''') are used for multiline comments
- Triple double quotes (""" """) are used for doc strings (describing function parameters or class properties etc.,)

Check all the three types of comments in the below code snippet.

In [1]:

```
# s =  'John's Byke' # This gives an error
s =  "John's Byke" # Enclose with proper quotes
print(s)
```

```
John's Byke
```

In the below cell, a single line string spanned in multiple lines using a backslash(\ )

In [2]:

```
s = 'Apple is sweet. ' \
'But Orange is Sour.'
print(s)
```

```
Apple is sweet. But Orange is Sour.
```

Strings

- String is a collection of characters.
- Any pair of quotes can be used to represent a string.
- Strings are immutable, we cannot add, delete, modify individual characters in a string.
- Python 2 default character encoding is ASCII, in python 3 it is UNICODE

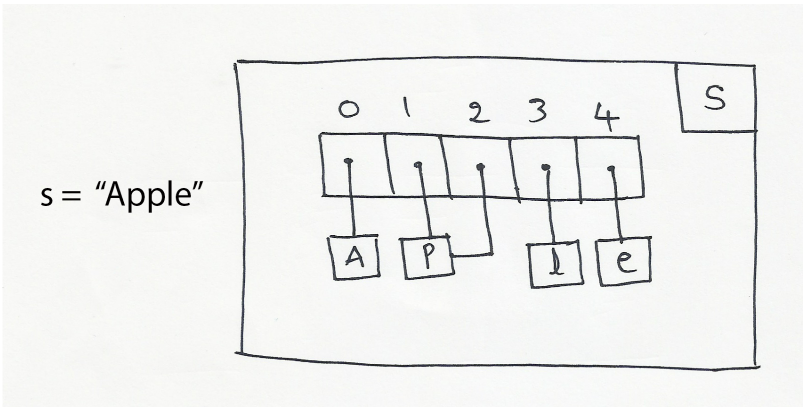
Individual characters in a string can be accessed using square brackets and indexing. Indexing starts from zero.
s[0] is 'A'
s[1] is 'p'
and so on.

In [3]:

```
s = 'Apple'
print(s[0], s[1], s[2])
```

```
A p p
```

Internal representation of a string



In [4]:

```
print(id(s[0]), id(s[1]), id(s[2]))
```

```
140154567352248 140154568108104 140154568108104
```

In the above example, 'p' is stored only once and its reference(address) is placed two times, at index 1 and 2, in the list of characters.

Finding length of the string - number of character in a string
len() function:

In [5]:

```
s = "Hello World!"
print(len(s)) # length of the string
```

```
12
```

Strings are immutable

- we cannot change individual characters
- We cannot add or delete characters

In [6]:

```
# **** Strings are immutable, we cannot change the characters
s = "Hello World!"
print(s)
```

```
Hello World!
```

ASCII and Unicode encoding

In python 3 characters are stored in Unicode encoding. We use prefix 'u' to define unicode strings in python 2

In [7]:

```
import sys
s = 'Apple'
print(type(s), sys.getsizeof(s))
```

```
<class 'str'> 54
```

String slicing

Slicing the technique of extracting sub string or a set of characters form a string.

syntax :

```
string[start:end:step]
```

- start - index start at which slicing is started
- end - index at which slicing is ended, end index is exclusive
- step - step value is with which start value gets incremented/decremented.

Note: Default step value is 1.

Lets see some examples,

In [8]:

```
s = "Hello World!"
print(s[6:11]) # returns a substring of characters from 6 to 11, excluding 11
```

```
World
```

In [9]:

```
print(s)
```

```
Hello World!
```

In [10]:

```
s[:4] # assumes start as 0
```

Out[10]:

```
'Hell'
```

In [11]:

```
s[6:] # assumes end as the length of the string
```

Out[11]:

```
'World!'
```

In [12]:

```
s[1:9] # returns a substring of characters from 1 to 8, excluding 9
```

Out[12]:

```
'ello Wor'
```

Step count - Default step count is 1

In [13]:

```
s[1:9:3]
```

Out[13]:

```
'eoo'
```

In [14]:

```
s[:10:2]
```

Out[14]:

```
'HloWr'
```

In the above example,
start is 1,
end is 9 and
step is 3.

first it prints s[1],
then s[1 + step] => s[1 + 2] => s[3]
prints s[3]
thne s[3 + step] which is s[5] and so on,
until it crosses 8.

In [15]:

```
s[:] # Returns Entire string
```

Out[15]:

```
'Hello World!'
```

In [16]:

```
s[::] # Returns Entire string, same as above
```

Out[16]:

```
'Hello World!'
```

In [17]:

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

Out[17]:

```
'HloWrd'
```

In the above example, it takes entire string, but step is 2, default start value is 0. so indices produced are 0, 2, 4, 6, 8, and 10.

In [18]:

```
s[9:2]
```

Out[18]:

```
''
```

In [19]:

```
s[9:2:-1]
```

Out[19]:

```
'lroW ol'
```

-ve indexing

Python supports -ve indexing. Index of last character is -1, last but one is -2 and so on.

In [20]:

```
s = "Hello World!"
s[-1]
```

Out[20]:

```
'!'
```

In [21]:

```
s[-2]
```

Out[21]:

```
'd'
```

Slicing using -ve indexing:

In [22]:

```
s[-9:-3]
```

Out[22]:

```
'lo Wor'
```

default step value is 1,
-9 + 1 ==> -8
-8 + 1 ==> -7
start value -9 is goin towards -3,
-9 ==> -3, so s[-9:-3] is a valid slice.

In [23]:

```
s[-3: -10]
```

Out[23]:

```
''
```

Above is not a valid slice, because

step is 1, default.
-3 + 1 ==> -2
-2 + 1 ==> -1
so on
-3 <== -10
-3 is not going towards -10, it never reaches -10, so invalid slice.
It returns ''(null string)

Some more examples,

In [24]:

```
s[-3: -10:-1]
```

Out[24]:

```
'lroW ol'
```

In [25]:

```
s[-4:-1:1]
```

Out[25]:

```
'rld'
```

Reversing a string

In [26]:

```
s[::]
```

Out[26]:

```
'Hello World!'
```

In [27]:

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

Out[27]:

```
'!dlroW olleH'
```

In [28]:

```
s
```

Out[28]:

```
'Hello World!'
```

Unfortunately this is the only standard way we can reverse a string in python. There are other complicated ways but not used in production.

In [29]:

```
s[3::-1]
```

Out[29]:

```
'lleH'
```

In [30]:

```
s[:3]
```

Out[30]:

```
'Hel'
```

In [31]:

```
s[:3:-1]
```

Out[31]:

```
'!dlroW o'
```

String functions

There are some useful functions on strings, below is the listing.

In [32]:

```
s = "hello World! 123$"
```

capitalize(): Captilize the first character and make remaining characters small

In [33]:

```
print(s.capitalize()) # no effect on non-alphabets
```

```
Hello world! 123$
```

Note: String functions do not effect original string, instead they take a copy of original string, process it and returns.

count(): Counts number of chars/substrings it has

In [34]:

```
s
```

Out[34]:

```
'hello World! 123$'
```

In [35]:

```
s.count('l') # number of 'l's in the string
```

Out[35]:

```
3
```

In [36]:

```
s.count('hell') # number of 'hell's in the string
```

Out[36]:

```
1
```

upper() and lower(): changing case to upper and lower, no effect on numbers and other characters.

In [37]:

```
s.upper()
```

Out[37]:

```
'HELLO WORLD! 123$'
```

In [38]:

```
s.lower()
```

Out[38]:

```
'hello world! 123$'
```

In [39]:

```
s
```

Out[39]:

```
'hello World! 123$'
```

Validation functions

In [40]:

```
s = 'hello World! 123$'
```

In [41]:

```
s.endswith("3$") # does s ends with '3$'
```

Out[41]:

```
True
```

In [42]:

```
s.endswith("5$") # does s ends with '5$'
```

Out[42]:

```
False
```

In [43]:

```
s.startswith("Apple") # does s starts with 'Apple'
```

Out[43]:

```
False
```

In [44]:

```
s.startswith("hello") # does s starts with 'hello'
```

Out[44]:

```
True
```

In [45]:

```
s = 'Apple123'
s.isalpha() # check the string is having only alphabets are not
```

Out[45]:

```
False
```

In [46]:

```
s = 'Apple'
s.isalpha() # check the string is having only alphabets are not
```

Out[46]:

```
True
```

In [47]:

```
s = "2314"
s.isdigit() # check the string is having only digit chars are not
```

Out[47]:

```
True
```

replace(): replaces all the occurances of substring in target string

In [48]:

```
s = 'Apple'
s.replace('p', '$')
print(s)
```

```
Apple
```

As we discussed, original string doesn't get changed, we just have to capture the modified string if we want to, as below

In [49]:

```
s = 'Apple'
s1 = s.replace('App', '$Tupp')
print(s1, s)
```

```
$Tupple Apple
```

strip(): Strips spaces on both the sides of the string. We can pass any custom chars/substrings if we want to strip. Below are the examples.

In [50]:

```
s = ' Apple '
print (len(s), s)
s = s.strip()
print (len(s), s)
```

```
7  Apple 
5 Apple
```

In [51]:

```
s = ' Apple'
print(len(s))
s = s.lstrip() # lstrip() works only on start of the string
print(len(s))
```

```
6
5
```

In [52]:

```
s = 'Apple '
print(len(s))
s = s.rstrip() # rstrip() works only on end of the string
print(len(s))
```

```
6
5
```

stripping custom chars/substrings

In [53]:

```
s = '$$$Telangana'
s.strip('$')
```

Out[53]:

```
'Telangana'
```

In [54]:

```
s
```

Out[54]:

```
'$$$Telangana'
```

In [55]:

```
s = 'ApApTelangana'
s.strip('gaAn')
```

Out[55]:

```
'pApTel'
```

split(): Splits entire string into multiple words seperated by spaces. We can pass custom seperators if want to.

In [56]:

```
date = '12/02/1984'
l = date.split('/')
print(l, type(l))
print()
print(l[-1], type(l[-1]))
```

```
['12', '02', '1984'] <class 'list'>

1984 <class 'str'>
```

In [57]:

```
date = '12/02/1984'
l = date.split('/', 1) # splits one-time

print(l, type(l))
print()
print(l[-1], type(l[-1]))
```

```
['12', '02/1984'] <class 'list'>

02/1984 <class 'str'>
```

In [58]:

```
s = '''Once upon a time in India, there was a king called Tippu.
India was a great country.'''

print(s.find('India'))
print(s.find('America'))
```

```
20
-1
```

rfind(): searching from the end

In [59]:

```
s.rfind('India')
```

Out[59]:

```
58
```

Index:

In [60]:

```
s.index('India')
```

Out[60]:

```
20
```

In [61]:

```
s.index('America')
```

```
---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
 last)
<ipython-input-61-2e0ea1183f5f> in <module>
----> 1 s.index('America')

ValueError: substring not found
```

Note: Difference between find() and index() is, index() throws ValueError if word is not found, whereas find() returns -1.

Exercise: Guess the output

In [62]:

```
s = '''Once upon a time in India, there was a king called Tippu.
India was a great country.'''

print(s[s.find('great'):])
```

```
great country.
```

List of chars to string:

In [63]:

```
l = ['A', 'p', 'p', 'l', 'e']
print(''.join(l))
```

```
Apple
```

In [64]:

```
l = ['A', 'p', 'p', 'l', 'e']
print('|'.join(l))
```

```
A|p|p|l|e
```

In [65]:

```
s = 'Once upon a time in Inida.'
words = s.split()
print(words)
```

```
['Once', 'upon', 'a', 'time', 'in', 'Inida.']
```

In [66]:

```
' '.join(words)
```

Out[66]:

```
'Once upon a time in Inida.'
```

In [67]:

```
emp_data = ['1234', 'John', '23400.0', 'Chicago']

print(','.join(emp_data))
```

```
1234,John,23400.0,Chicago
```

Program: Reverse the word 'India' in-place in the below string.

In [68]:

```
s = '''Once upon a time in India, there was a king called Tippu. India was a gre
word = 'India'

print(s.replace(word, word[::-1]))
```

```
Once upon a time in aidnI, there was a king called Tippu. aidnI was a
great country.
```

Program: Count all the vowels in the given string.

In [69]:

```
s = '''once upon a time in india, there was a king called tippu. india was a gre

s.count('a')+ s.count('e') + s.count('i') + s.count('o') + s.count('u')
```

Out[69]:

```
29
```

Scalar multiplication

In [70]:

```
'Apple' * 5
```

Out[70]:

```
'AppleAppleAppleAppleApple'
```

Concatenating Strings

In [71]:

```
'Apple' + 'Orange'
```

Out[71]:

```
'AppleOrange'
```

Character encoding In python 2, a prefix 'u' is required to write unicode strings.

In [72]:

```
s = u'Apple'
print(s)
```

```
Apple
```

In [73]:

```
import math
```

In [74]:

```
math.sin(90)
```

Out[74]:

```
0.8939966636005579
```

In [75]:

```
from math import sin
sin(90)
```

Out[75]:

```
0.8939966636005579
```

In [76]:

```
help(sin)
```

```
Help on built-in function sin in module math:

sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

Exercise Programs

1. Add a comma between the characters. If the given woord is 'Apple', it should become 'A,p,p,l,e'
2. Remove the given word in all the places in a string?

Comprehension Quiz

Think you've got it? Here's a tiny quiz:

1. What's different about Python 2.x and Python 3.x in regards to string handling/Unicode/UTF-8?
2. What is the output when following statement is executed?

```
print ('Tech'  'Beamers')
    a. Beamers
    b. Tech
    c. TechBeamers
    d. Tech Beamers
```

3. What is the output when following statement is executed?

```
print (R'Tech\nBeamers')
    a. Tech Beamers
    b. Tech\nBeamers
    c. 'RTech' then 'Beamers' in a New line
    d. 'Tech' then 'Beamers' in a New line
```

4. Which of the following is the output of the below Python code?

```
str='Hello World'
print (str.find('o'))
    a. 4
    b. 4, 7
    c. 7
    d. 2
```

5. What is the output when following code will be executed?

```
str='Recurssion'
print  (str.rfind('s'))
    a. 5
    b. 6
    c. 4
    d. 2
```

Answers are below

```
1. Strings are UTF-8 by default in Python 3.x whereas strings are ascii by d
efault and a prefix 'u' is required
to write unicode strings in python 2.x
2. c (Note:- string literals when separated with space are written together
they get concatenated.)
3. b (Note:- ‘R’ stands for raw string this suppresses the meaning of escape
characters and they get printed in the String.)
4. a (Note:- Find method returns the lowest index at which the string is fou
nd.)
5. b (Note: rFind method returns the highest index at which the string is fo
und.)
```