04_Data_Structures2

April 11, 2020

0.1 4. Strings

Strings are ordered text based data which are represented by enclosing the same in *single/double/triple quotes*.

[1]:
```
String1 = 'Taj Mahal is beautiful'
String2 = "Taj Mahal is beautiful"
String3 = '''Taj Mahal
is
beautiful'''
```

[2]:
```
print (String1,type(String1))
print (String2,type(String2))
print (String3,type(String3))
```

```
Taj Mahal is beautiful <class 'str'>
Taj Mahal is beautiful <class 'str'>
Taj Mahal
is
beautiful <class 'str'>
```

String Indexing and Slicing are similar to Lists which was explained in detail earlier.

[3]:
```
print (String1[4])
print (String3[4:])
```

```
M
Mahal
is
beautiful
```

0.1.1 Built-in Functions

find( ) function returns the index value of the given data that is to found in the string. If it is not found it returns **-1**. Remember to not confuse the returned -1 for reverse indexing value.

[4]:
```
String1
```

[4]: 'Taj Mahal is beautiful'

[5]:
```
print (String1.find('al'))
print (String1.find('am'))
```

```
7
-1
```

The index value returned is the index of the first element in the input data.

One can also input **find( )** function between which index values it has to search.

[6]:
```
String1
```

[6]:
```
'Taj Mahal is beautiful'
```

[7]:
```
print (String1.find('j',1))
print (String1.find('a',6,9))
```

```
2
7
```

capitalize( ) is used to capitalize the first element in the string.

[8]:
```
String4 = 'a observe the first letter in this sentence.'
print (String4.capitalize())
```

```
A observe the first letter in this sentence.
```

center( ) is used to center align the string by specifying the field width.

[9]:
```
String1.center(40)
```

[9]:
```
'         Taj Mahal is beautiful         '
```

One can also fill the left out spaces with any other character.

[10]:
```
String1.center(40,'-')
```

[10]:
```
'---------Taj Mahal is beautiful---------'
```

zfill( ) is used for zero padding by specifying the field width.

[11]:
```
String1.zfill(40)
```

[11]:
```
'00000000000000000Taj Mahal is beautiful'
```

expandtabs( ) allows you to change the spacing of the tab character. '\t' which is by default set to 8 spaces.

[12]:
```
s = 'h\te\tl\tl\to'
print (s)
print (s.expandtabs(1))
```

```
print (s.expandtabs())
```

```
h       e       l       l       o
h e l l o
h       e       l       l       o
```

endswith( ) function is used to check if the given string ends with the particular char which is given as input.

[13]:
```
print (String1.endswith('ful'))
```

```
True
```

The start and stop index values can also be specified.

[14]:
```
String1
```

[14]:
```
'Taj Mahal is beautiful'
```

[15]:
```
print (String1.endswith('l',0))
print (String1.endswith('a',0,6))
```

```
True
True
```

count( ) function counts the number of char in the given string. The start and the stop index can also be specified or left blank. (These are Implicit arguments which will be dealt in functions)

[16]:
```
print (String1.count('a',0))
print (String1.count('a',5,10))
```

```
4
2
```

lower( ) converts any capital letter to small letter.

[17]:
```
print (String1)
print (String1.lower())
```

```
Taj Mahal is beautiful
taj mahal is beautiful
```

upper( ) converts any small letter to capital letter.

[18]:
```
String1.upper()
```

[18]:
```
'TAJ MAHAL IS BEAUTIFUL'
```

replace( ) function replaces the element with another element.

[19]:
```
String1.replace('Taj Mahal','Hyderabad')
```

[19]: 'Hyderabad is beautiful'

[20]:
```
String1
```

[20]: 'Taj Mahal is beautiful'

strip( ) function is used to delete elements from the right end and the left end which is not required.

[21]:
```
f = '    hello      '
```

If no char is specified then it will delete all the spaces that is present in the right and left hand side of the data.

[22]:
```
f.strip()
```

[22]: 'hello'

[23]:
```
String1.strip()
```

[23]: 'Taj Mahal is beautiful'

0.2 Dictionaries

Dictionaries are more used like a database because here you can index a particular sequence with your user defined string.

To define a dictionary, equate a variable to { } or dict()

[24]:
```
d0 = {}
d1 = dict()
print (type(d0), type(d1))
```

```
<class 'dict'> <class 'dict'>
```

Dictionary works somewhat like a list but with an added capability of assigning it's own index style.

[25]:
```
d0['name'] = "Ramesh"
d0['age'] = 24
print (d0)
```

```
{'name': 'Ramesh', 'age': 24}
```

That is how a dictionary looks like. Now you are able to access 'Ramesh' by the index value set at 'name'

[26]:
```
print (d0['name'])
```

```
Ramesh
```

0.2.1 Built-in Functions

values( ) function returns a list with all the assigned values in the dictionary.

[27]:
```
d0.values()
```

[27]:
```
dict_values(['Ramesh', 24])
```

keys( ) function returns all the index or the keys to which contains the values that it was assigned to.

[28]:
```
d0.keys()
```

[28]:
```
dict_keys(['name', 'age'])
```

items( ) is returns a list containing both the list but each element in the dictionary is inside a tuple. This is same as the result that was obtained when zip function was used.

[29]:
```
d0.items()
```

[29]:
```
dict_items([('name', 'Ramesh'), ('age', 24)])
```

pop( ) function is used to get the remove that particular element and this removed element can be assigned to a new variable. But remember only the value is stored and not the key. Because the is just a index value.

[30]:
```
d1 = d0.pop('age')
print (d0)
print (d1)
```

```
{'name': 'Ramesh'}
24
```

[31]:
```
d0.clear() #makes dict empty
```

[32]:
```
d0
```

[32]:
```
{}
```

[33]:
```
emp=dict()
emp['eid']=[101,102,103]
emp['ename']=['x','y','z']
emp
```

[33]:
```
{'eid': [101, 102, 103], 'ename': ['x', 'y', 'z']}
```

[34]:
```
emp['salary']=[10000,20000, 25000]
emp
```

[34]:
```
{'eid': [101, 102, 103],
 'ename': ['x', 'y', 'z'],
 'salary': [10000, 20000, 25000]}
```

[ ]: