String Processing- Tips and Tricks in Python

1. Stripping Whitepsace

In Python we can remove leading and trailing blank spaces or unnecessary characters in a String using following function:

lstrip(): Strip leading whitespace.

rstrip(): Strip trailing whitespace.

strip(): both leading and trailing whitespace.

s = '   KIIT is the best Private University in India.   \n'

print('Strip leading whitespace: {}'.format(s.lstrip()))

print('Strip trailing whitespace: {}'.format(s.rstrip()))

print('Strip all whitespace: {}'.format(s.strip()))

Strip leading whitespace: KIIT is the best Private University in India.

Strip trailing whitespace: KIIT is the best Private University in India.

Strip all whitespace: KIIT is the best Private University in India.

Interested in stripping characters other than whitespace? The same methods are helpful, and are used by passing in the character(s) you want stripped.

s = 'This is a sentence with unwanted characters.AAAAAAAA'

print('Strip unwanted characters: {}'.format(s.rstrip('A')))

Strip unwanted characters: This is a sentence with unwanted characters.

Don't forget to check out the string format() documentation if necessary.

2. Splitting Strings

Splitting strings into lists of smaller substrings is often useful and easily accomplished in Python with the split() method.

s = 'KIIT Deemed to be University'

print(s.split())

['KIIT', 'Deemed', 'to', 'be', 'University']

By default, split() splits on whitespace, but other character(s) sequences can be passed in as well.

```
s = 'Rajesh,Amit,Sanjib,Rakesh'

print('\',\' separated split -> {}'.format(s.split(',')))

s = 'abacbdebfgbhhgbabddba'

print('\'b\' separated split -> {}'.format(s.split('b')))
```

',' separated split -> ['Rajesh', 'Amit', 'Sanjib', 'Rakesh']

'b' separated split -> ['a', 'ac', 'de', 'fg', 'hhg', 'a', 'dd', 'a']

3. Joining List Elements into a String

Need the opposite of the above operation? You can join list element strings into a single string in Python using the join() method.

```
s = ['KIIT', 'Deemed', 'to', 'be', 'University']

print(''.join(s))
```

KIIT Deemed to be University

if you want to join list elements with something other than whitespace in between? This thing may be a little bit stranger, but also easily accomplished.

```
s = ['Rajesh', 'Amit', 'Dilip', 'Sanjib', 'Rakesh']

print(' and '.join(s))
```

Rajesh and Amit and Dilip and Sanjib and Rakesh

4. Reversing a String

Python does not have a built-in string reverse method. However, given that strings can be sliced like lists, reversing one can be done in the same succinct fashion that a list's elements can be reversed.

```
s = 'Rajesh'

print('The reverse of Rajesh is {}'.format(s[::-1]))
```

The reverse of Rajesh is: hsejaR

5. Converting Uppercase and Lowercase

Converting between cases can be done with the upper(), lower(), and swapcase() methods.

```
s = 'KIIT University'

print('\'KIIT University \' as uppercase: {}'.format(s.upper()))

print('\'KIIT University \' as lowercase: {}'.format(s.lower()))

print('\'KIIT University \' as swapped case: {}'.format(s.swapcase()))
```

'KIIT University' as uppercase: KIIT UNIVERSITY

'KIIT University' as lowercase: kiit university

'KIIT University' as swapped case: kiit uNIVERSITY

6. Checking for String Membership

The easiest way to check for string membership in Python is using the in operator. The syntax is very natural language-like.

```
s1 = 'KIIT University'

s2 = 'IIT'

s3 = 'NIT'

print('\'IIT\' in \'KIIT University\' -> {}'.format(s2 in s1))

print('\'NIT\' in \'KIIT University\' -> {}'.format(s3 in s1))
```

'IIT' in ''KIIT University ' -> True

'NIT' in ''KIIT University ' -> False

If you are more interested in finding the location of a substring within a string (as opposed to simply checking whether or not the substring is contained), the find() string method can be more helpful.

```
s = 'Does this string contain a substring?'

print('\'string\' location -> {}'.format(s.find('string')))

print('\'spring\' location -> {}'.format(s.find('spring')))
```

'string' location -> 10

'spring' location -> -1

find() returns the index of the first character of the first occurrence of the substring by default, and returns -1 if the substring is not found. Check the documentation for available tweaks to this default behavior.

7. Replacing Substrings

What if you want to replace substrings, instead of just find them? The Python replace() string method will take care of that.

```
s1 = 'The theory of data science is of the utmost importance.'

s2 = 'practice'

print('The new sentence: {}'.format(s1.replace('theory', s2)))
```

The new sentence: The practice of data science is of the utmost importance.

An optional count argument can specify the maximum number of successive replacements to make if the same substring occurs multiple times.

8. Combining the Output of Multiple Lists

Have multiple lists of strings you want to combine together in some element-wise fashion? No problem with the zip() function.

```
countries = ['USA', 'Canada', 'UK', 'Australia']

cities = ['Washington', 'Ottawa', 'London', 'Canberra']

for x, y in zip(countries, cities):

print('The capital of {} is {}.'.format(x, y))
```

The capital of USA is Washington.

The capital of Canada is Ottawa.

The capital of UK is London.

The capital of Australia is Canberra.

9. Checking for Anagrams

Want to check if a pair of strings are anagrams of one another? Algorithmically, all we need to do is count the occurrences of each letter for each string and check if these counts are equal. This is straightforward using the Counter class of the collections module.

```
from collections import Counter

def is_anagram(s1, s2):

        return Counter(s1) == Counter(s2)
```

```
s1 = 'listen'

s2 = 'silent'

s3 = 'runner'

s4 = 'neuron'

print('\'listen\' is an anagram of \'silent\' -> {}'.format(is_anagram(s1, s2)))

print('\'runner\' is an anagram of \'neuron\' -> {}'.format(is_anagram(s3, s4)))
```

'listen' an anagram of 'silent' -> True

'runner' an anagram of 'neuron' -> False

10. Checking for Palindromes

How about if you want to check whether a given word is a palindrome? Algorithmically, we need to create a reverse of the word and then use the == operator to check if these 2 strings (the original and the reverse) are equal.

```
def is_palindrome(s):
        reverse = s[::-1]
        if (s == reverse):
                return True
        return False

s1 = 'racecar'

s2 = 'hippopotamus'

print('\'racecar\' a palindrome -> {}'.format(is_palindrome(s1)))

print('\'hippopotamus\' a palindrome -> {}'.format(is_palindrome(s2)))
```

'racecar' is a palindrome -> True

'hippopotamus' is a palindrome -> False

***