String_manipulation_and_methods

February 4, 2023

0.1 6.2.1 String manipulation

Strings are used to record the text information such as name. In Python, Strings act as "Sequence" which means Python tracks every element in the String as a sequence. This is one of the important features of the Python language.

For example, Python understands the string "hello' to be a sequence of letters in a specific order which means the indexing technique to grab particular letters (like first letter or the last letter).

As we now know string contains set of characters, let's check how can we manipulate and take subset of string, how can we access charcters out of string and do some manipulation on strings.

Subset of string can be access by using slice operator [] or :. Lets look at the example.

[ ]:
```
# Single word
'hello'
```

[ ]:
```
'hello'
```

[ ]:
```
# Entire phrase
'This is also a string'
```

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

[ ]:
```
# We can also use double quote
"String built with double quotes"
```

[ ]:
```
'String built with double quotes'
```

[ ]:
```
# Be careful with quotes!
' I'm using single quotes, but will create an error'
```

```
  File "<ipython-input-26-3b26cff97c4c>", line 2
    ' I'm using single quotes, but will create an error'
       ^
SyntaxError: invalid syntax
```

The above code results in an error as the text "I'm" stops the string. Here, a combination of single quotes and double quotes can be used to get the complete statement.

[ ]:
```
"Now I'm ready to use the single quotes inside a string!"
```

[ ]:
```
"Now I'm ready to use the single quotes inside a string!"
```

We can automatically display the output strings using Jupyter notebook with just a string in a cell. But,the correct way to display strings in your output is by using a print function.

[ ]:
```
"iNeuron"
```

[ ]:
```
'iNeuron'
```

0.2 6.2.2 Python 3 Alert!

Note that, In Python 3, print is a function and not a statement. So you would print statements like this: print('Hello World')

If you want to use this functionality in Python2, you can import form the **future** module.

Caution: After importing this; you won't be able to choose the print statement method anymore. So pick the right one whichever you prefer depending on your Python installation and continue on with it.

[ ]:
```
# To use print function from Python 3 in Python 2
from __future__ import print_function

print('Hello World')
```

```
Hello World
```

[ ]:
```
string = "iNeuron"

print(string)
```

```
iNeuron
```

0.3 6.2.3 String operations

0.3.1 Accessing element from string

We know strings are a sequence, which means Python can use indexes to call all the sequence parts. Let's learn how String Indexing works.

- We use brackets `[]` after an object to call its index.
- We should also note that indexing `starts at 0` for Python.

Now, Let's create a new object called s and the walk through a few examples of indexing.

[ ]:
```
# Fetch first character of a string
print(string[0])
```

```
i
```

[ ]:
```
# Fetch last element of a string
print(string[-1])
```

```
n
```

[ ]:
```
# Fetch nth element of a string
print(string[4])
```

```
r
```

[ ]:
```
If length exceeds, then it will give index out of range as we are finding index
print(string[9])
```

```
  File "<ipython-input-76-7fb3b9347dc6>", line 1
    If length exceeds, then it will give index out of range as we are finding␣
 ↪index
            ^
SyntaxError: invalid syntax
```

We can use a : to perform *slicing* which grabs everything up to a designated point. For example:

[ ]:
```
# This slice operation will help us to fetch substring from a string.
# [ Index given before colon will be starting index and index given after colon␣
 ↪will be ending index and
# it is not considered to print]
#  And if no index is given it will consider till end of string index
print(string[1:])
```

```
Neuron
```

[ ]:
```
# This will give character starting from index 1 and ending index 2 (Last will␣
 ↪not consider for slice operation)
print(string[1:3])
```

```
Ne
```

Note the above slicing. Here we're telling Python to grab everything from 1 up to 3. It doesn't include the 3rd index. You'll notice this a lot in Python, where statements and are usually in the context of "up to, but not including".

[ ]:
```
# It will give all the characters but not the last three characeters
print(string[:-3])
```

```
iNeu
```

[ ]:
```
# It will give all the characters but not the first two characeters
print(string[2:])
```

```
euron
```

[ ]:
```
#It will give the last two char
print(string[-2:])
```

```
on
```

Index and slice notation is used to grab elements of a sequenec by a specified step size (where in 1 is the default size). For instance we can use two colons in a row and then a number specifying the frequency to grab elements. For example:

[ ]:
```
# Grab everything, but go in steps size of 1
s = "Hello world"
s[::1]
```

[ ]:
```
'Hello world'
```

[ ]:
```
# Grab everything, but go in step sizes of 2
s[::2]
```

[ ]:
```
'Hlowrd'
```

[ ]:
```
# We can use this to print a string backwards
s[::-1]
```

[ ]:
```
'dlrow olleH'
```

Immutability is one the finest string property whichh is created once and the elements within it cannot be changed or replaced. For example:

[ ]:
```
# Let's try to change the first letter to 'x'
s[0] = 'x'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-976942677f11> in <module>
      1 # Let's try to change the first letter to 'x'
----> 2 s[0] = 'x'

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

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

Something we can do is concatenate strings!

[ ]:
```
# We can reassign s completely though!
s = s+ " concatenate me!"
s
```

[ ]:
```
'Hello world concatenate me!'
```

[ ]:
```
# String can be concatenate using + sign operator
print(string + " Company ")
```

```
iNeuron Company
```

[ ]:
```
print(" Hello " + " World ")
```

```
 Hello  World
```

[ ]:
```
# Print string two times using * operator
#We can use the multiplication symbol to create repetition!
print(string * 2)
```

```
iNeuroniNeuron
```

0.3.2 String length

[ ]:
```
# To fing lengt                                    h of a string, we can use len
len(string)
```

[ ]:
```
7
```

Find character/word in a string

[ ]:
```
# To find a character in a string, use find and it will give index of that␣
 ↪character
string.find("n")
```

[ ]:
```
6
```

[ ]:
```
# If it is not able to find character , it will give index as a -1
string.find('x')
```

[ ]:
```
-1
```

0.3.3 Count characters in a string

[ ]:
```
# To count no. of characters in a string, can use count method
print(string.count(' '))
```

```
0
```

[ ]:
```
print(string.count('n'))
```

```
1
```

0.3.4 String split operation

[ ]:
```
# To split string at certain space/character, will return list of strings after splitting
print(string.split(' '))
```

```
['iNeuron']
```

[ ]:
```
string.split('u')
```

[ ]:
```
['iNe', 'ron']
```

0.3.5 Change strings to upper & lower case

[ ]:
```
# Changes to upper case
print(string.upper())
```

```
INEURON
```

[ ]:
```
# Changes to lower case
print(string.lower())
```

```
ineuron
```

[ ]:
```
# Swap case from lower to upper & upper to lower
print(string.swapcase())
```

```
InEURON
```

[ ]:
```
print(string.title())
```

```
Ineuron
```

[ ]:
```
print(string.capitalize())
```

```
Ineuron
```

0.3.6 Reverse string

[ ]:
```
# Can use reversed for reversing string
print(' '.join(reversed(string)))
```

```
n o r u e N i
```

[ ]:
```
# We can do reverse of a string by extended slice functionality [::1], so here the third one is the optional step size
# through which we are reversing by using step size as -1
print(string[::-1])
```

```
norueNi
```

0.3.7 Removing characters from the end of the string

[ ]:
```
string_a =" ineuron "
```

[ ]:
```
string_a
```

[ ]:
```
' ineuron '
```

[ ]:
```
# Strip will remove white space from both end of the strings
string_a.strip(" ")
```

[ ]:
```
'ineuron'
```

[ ]:
```
# removes leading character from a string
string_a.lstrip(" ")
```

[ ]:
```
'ineuron '
```

[ ]:
```
# removes trailing character from a string
string_a.rstrip(" ")
```

[ ]:
```
' ineuron'
```

0.3.8 Join operation in string

[ ]:
```
" ".join("Welcome to ineuron")
```

[ ]:
```
'W e l c o m e   t o   i n e u r o n'
```

0.3.9 Replace string

[ ]:
```
string_n = "greetings to ineuron"
string_n.replace("to","from")
```

[ ]:
```
'greetings from ineuron'
```

0.3.10 Formatting

The `center()` method allows you to place your string 'centered' between a provided string with a certain length.

[ ]:
```
string.center(20,'z')
```

[ ]:
```
'zzzzzziNeuronzzzzzzz'
```

`expandtabs()` will expand tab notations into spaces. Let's see an example to understand the concept.

[ ]:
```
'hello\thi'.expandtabs()
```

[ ]:
```
'hello    hi'
```

0.3.11 Checking string case

[ ]:
```
# Check if string are in upper case
string.isupper()
```

[ ]:
```
False
```

[ ]:
```
# Check if string are in lower case
string.islower()
```

[ ]:
```
False
```

[ ]:
```
# Check if string contains space
string.isspace()
```

[ ]:
```
False
```

[ ]:
```
# Check if string contains digit
string.isdigit()
```

[ ]:
```
False
```

[ ]:
```
# Check if string endswith character n
string.endswith('n')
```

[ ]:
```
True
```

[ ]:
```
# Check if string startswith character n
string.startswith('i')
```

[ ]:
```
True
```

[ ]:
```
#check if all char in string are alphanumeric
a = "abcd1234"
a.isalnum()
```

[ ]:
```
True
```

[ ]:
```
#test if string contains title words
a="Abcdef"
```

```
a.istitle()
```

[ ]: True

0.3.12 Iterate through a string

[ ]:
```
# Iterating through a string and count letters in a string
count = 0
for ch in 'Greetings from iNeuron':
    count += 1
print(count,'letters found')
```

```
22 letters found
```

[ ]:
```
# Same operation can be done using len
len("Greetings from iNeuron")
```

[ ]: 22

[ ]:
```
# Using range() to iterating through a string
string = "iNeuron"
for ch in range(len(string)):
   print(string[ch])
```

```
i
N
e
u
r
o
n
```

[ ]:
```
# We can use index to iterate string reverse direction
string = "iNeuron"
ch = len(string) - 1
while ch >= 0:
   print(string[ch])
   ch -= 1
```

```
n
o
r
u
e
N
i
```

[ ]:
```
Name = "ineuron"
vowels = "AaEeIiOoUu"
for ch in Name:
    if ch in vowels:
        print("{} is a vowel".format(ch))
    else:
        print("{} is not a vowel".format(ch))
```

```
i is a vowel
n is not a vowel
e is a vowel
u is a vowel
r is not a vowel
o is a vowel
n is not a vowel
```

[2]:
```
''' We will see famous palindrome program. String is called palindrom if we␣
 ↪reverse the string then
    original and reversed string are same or not.
    function to check string is
    palindrome or not
'''
def palindrome_check(str):

    # looping from 0 to len(str)/2
    for i in range(0, int(len(str)/2)):
        if str[i] != str[len(str)-i-1]:

            return False
    return True

    # Or for reversing we can use str == str[::-1] also.

s = "malayalam"
res = palindrome_check(s)

if (res):
    print("Yes string is palindrome")
else:
    print("No string is not palindrome")
```

```
Yes string is palindrome
```

[ ]:

[ ]: