Python Data Structures: Strings:

Programming for Data Science with Python

1. Overview

- In Python, ***strings*** are the objects of the class str that has the ***constructor str()***.
- Strings are one of the most popular data types/data structures in Python.
- We can create them simply by enclosing characters in quotes (single or double).
- Python treats single quotes the same as double-quotes.
- Creating strings is as simple as assigning a value to a variable.

  ### **Run the following 2 code blocks:**

In [1]:
```
aStr = "Hello"
print(aStr)
```
```
Hello
```

In [2]:
```
language = "English"
print(language)
```
```
English
```

In [ ]:
```
aStr2 = 'Hello'
print(aStr2)
```

In [3]:
```
language_2nd = 'Spanish'
print(language_2nd)
```
```
Spanish
```

1.1 Length of Strings

- The ***length*** of a string is the number of characters of the string.
- The ***length*** of a string can be obtained using the built-in function ***len()***.

IMPORTANT NOTES: ***len()*** is a ***built-in** function of Python, not a method of class str.

Run the following code block:

In [4]:
```
# Declare a string
aStr = "This is a string . "
print ("The length of this string - or the number of characters: ", len(aStr))
```

```
The length of this string - or the number of characters:  19
```

In [5]:
```
bStr = 'This is another string, but cooler.'
print('Length of bStr is : ', len(bStr))
```

```
Length of bStr is :  35
```

1.2 String Indices

- **String is a sequence data type/structure** in Python.
- Like any other sequence data type in Python, the **indices** of a string always start with 0.
- The range of indices of a string: 0 ... len(string) - 1

  ### **Run the following 5 code blocks:**

In [6]:
```
aStr = "This is a string . "
print("The length of this string: ", len(aStr))
```

```
The length of this string:  19
```

In [15]:
```
bStr = 'This is another string, but cooler!'
print('Length of bStr is : ', len(bStr))
```

```
Length of bStr is :  35
```

In [8]:
```
print(aStr[0])
```

```
T
```

In [9]:
```
print(bStr[5])
```

```
i
```

In [10]:
```
print(aStr[1])
```

```
h
```

In [11]:
```
print(bStr[32])
```

```
e
```

In [12]:
```
print(aStr[16])
# Notice: This will return a blank space.
```

In [13]:
```
print(bStr[4])
```

In [14]:
```
print (aStr[17])
#Notice:  This will return a period.
```

```
.
```

In [17]:
```
print(bStr[34])
```

```
!
```

2. Create Strings

2.1 Using String Literals

Run the following 4 code blocks:

In [18]:
```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)
```

```
Hello
```

In [19]:
```
his_string = "He/Him"
print(his_string)
```

```
He/Him
```

In [20]:
```
my_string = "Hello"
print(my_string)
```

```
Hello
```

In [21]:
```
her_string = "She/Her"
print(her_string)
```

```
She/Her
```

In [22]:
```
my_string = '''Hello'''
print(my_string)
```

```
Hello
```

In [23]:
```
their_str = '''They/Them'''
print(their_str)
```

```
They/Them
```

In [24]:
```
# triple quotes string can extend multiple Lines
my_string ="""Hello. Welcome to
Python World!"""
print(my_string)
```

```
Hello. Welcome to
Python World!
```

In [25]:
```
web_string = """With great power,
comes great responsibility."""
print(web_string)
```

```
With great power,
comes great responsibility.
```

2.2 Create Strings from Lists - Using join() method

Run the following 2 code blocks:

In [26]:
```
# VERSION 1: List of strings--> A string

alist = ["This", "is", "a", "string"]
print ("This is a list: ", alist)
```
```
This is a list:  ['This', 'is', 'a', 'string']
```

In [27]:
```
testList = ['Testing', 'one', 'two']
print(testList)
```
```
['Testing', 'one', 'two']
```

In [28]:
```
aString =" " . join(alist)
# aString is a string and so is alist
print(aString)
```
```
This is a string
```

In [29]:
```
bString =" ".join(testList)
print(bString)
```
```
Testing one two
```

2.3 Create Strings from Lists - Using str() and join ()

Run the following 2 code blocks:

In [30]:
```
# Version 2: List of numbers--> A string

# A List of numbers
alist = [20, 30, 40, 50, 60]

# Convert aList into a List of strings - Using the constructor str()
aStrList = [str(element) for element in alist]

print ("This is a list of strings: ", aStrList)
```
```
This is a list of strings:  ['20', '30', '40', '50', '60']
```

In [31]:
```
kBryant_pts = [81, 65, 62, 61, 60]
kBryantList = [str(pt) for pt in kBryant_pts]

print('Kobe Bryant Top 5 Highest scores', kBryantList)
```
```
Kobe Bryant Top 5 Highest scores ['81', '65', '62', '61', '60']
```

In [32]:
```
# Using join() to create a new string
aString =" " . join(aStrList)

# aString = "20 30 40 50 60"
print("This is a string : ", aString)
```
```
This is a string :  20 30 40 50 60
```

In [35]:
```
kBString = '   '.join(kBryantList)
print("Kobe Bryant's Top 5 Highest scores: ", kBString)
```
```
Kobe Bryant's Top 5 Highest scores:  81   65   62   61   60
```

2.4 Create Strings from Lists - Using map() and join()

Run the following code block:

In [36]:
```
# Generate the combination from the list
# Then transform each element of the list into a string

from itertools import combinations
L = [1, 2, 3, 4]

print(combinations(L, 3))

# Using map() and join() to convert each numeric combination into as string
# Thanks to this technique, we can display the List of combinations

[",".join(map(str, comb)) for comb in combinations(L, 3)]
```

```
<itertools.combinations object at 0x000002AB12A74450>
```

Out[36]:
```
['1,2,3', '1,2,4', '1,3,4', '2,3,4']
```

3. Access Characters in Strings

3.1 Access Single Characters

Run the following 4 code blocks:

In [37]:
```
# Python allows negative indexing for its sequences.
# The index of -1 refers to the last item, -2 to the second to the last item, and so o
# We can access a range of items in a string by using the slicing operator (colon).
str = 'programiz'
print('str = ', str)
```

```
str =  programiz
```

In [39]:
```
one_string = "to rule them all."
print('The One string...', one_string)
```

```
The One string... to rule them all.
```

In [40]:
```
# first character
print('str[0] = ', str[0])
```

```
str[0] =  p
```

In [43]:
```
print('one_string[5] = ', one_string[5]) #fifth character
```

```
one_string[5] =  l
```

In [44]:
```
# Third character
print('str[0] = ', str[2])
```

```
str[0] =  o
```

In [45]:
```
print('one_string[10] = ', one_string[10]) #tenth character
```

```
one_string[10] =  e
```

In [46]:
```
#Last character
print('str[-1] = ', str[-1])
```

```
str[-1] =  z
```

In [47]:
```
print('one_string[-2] = ', one_string[-2]) #second-to-last character
```

```
one_string[-2] =  l
```

3.2 Access a Slice of Strings

Run the following 6 code blocks:

In [48]:
```
#slicing 2nd to 5th character

str='programiz'

print('str[1:5]= ', str[1:5])
```

```
str[1:5]=  rogr
```

In [52]:
```
marvel_str = 'Assemble!'
print('marvel_str[2:8] =', marvel_str[2:8])
```

```
marvel_str[3:8] = emble
```

In [57]:
```
#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

```
str[5:-2] =  am
```

In [62]:
```
marvel_str = 'Assemble!'
print('marvel_str[2:-7] =', marvel_str[:-7])
```

```
marvel_str[2:-7] = As
```

In [63]:
```
sample_str = 'Python String'

# Print a range of character starting from index 3 to index 4
print (sample_str[3:5])
```

```
ho
```

In [65]:
```
star_str = "May the force be with you."
print(star_str[8:13])
```

```
force
```

In [66]:
```
# Print all characters from index 7
print (sample_str[7:])
```

```
String
```

In [69]:
```
print(star_str[22:])
```

```
you.
```

In [70]:
```
# Print all characters before index 6
print(sample_str[:6])
```
```
Python
```

In [72]:
```
print(star_str[:13])
```
```
May the force
```

In [73]:
```
#Print all characters from index 7 to the index -4 (count from)
print (sample_str[7:-4])
```
```
St
```

In [74]:
```
print(star_str[13:-5])
```
```
 be with
```

4. Modify Strings

IMPORTANT NOTES:

- ***Strings are immutable***, i.e. they cannot be changed after being created.
- Any attempt to change or modify the contents of strings will lead to errors.

Run the following code block:

In [75]:
```
sample_str = 'Python String'
sample_str[2] = 'a'
# Do you know why you have an error in your output?
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[75], line 2
      1 sample_str = 'Python String'
----> 2 sample_str[2] = 'a'

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

IMPORTANT NOTES:

Strings are immutable.

- This means that elements of a string cannot be changed once it has been assigned.
- But an existing string variable can be re-assigned with a brand new string.

Run the following 2 code blocks:

In [76]:
```
str2 = "This is a string . "
print ("str2: ", str2)
```
```
str2:  This is a string .
```

In [77]:
```
trek_str = "Live long, and prosper."
print(trek_str)
```
```
Live long, and prosper.
```

In [78]:
```
# Reassign a new tuple to tuple1
str2 = "This is a new string."
print("str2 after being re-assinged : ", str2)
```
```
str2 after being re-assinged :  This is a new string.
```

In [81]:
```
trek_str = "Live long,and prosper? That's absurd."
print('Albert Camus\' response to Star Trek\'s favorite quote:', trek_str)
```
```
Albert Camus' response to Star Trek's favorite quote: Live long,and prosper? That's a
bsurd.
```

5. Copy Strings

5.1 Shallow copy

- Shallow copy means that only the reference to the object is copied. No new object is created.
- Assignment with an = on string does not make a copy.
- Instead, assignment makes the two variables point to the same list in memory.

Run the following code block:

In [82]:
```
strl = "Hello"
str2 = str1
# Both the strings refer to the same object, i . e. , the same id value
id(str1), id(str2)
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[82], line 2
      1 strl = "Hello"
----> 2 str2 = str1
      3 # Both the strings refer to the same object, i . e. , the same id value
      4 id(str1), id(str2)

NameError: name 'str1' is not defined
```

5.2 Deep copy

Deep copy means that a new object will be created when the copying has done.

IMPORTANT NOTES: Strings are ***immutable sequence objects***. Strings **cannot be deep-copied***.

6. Delete Strings

To ***delete a string***, using the built-in function ***del()***.

Run the following 2 code blocks:

In [87]:
```
sample_str = "Python is the best scripting language."
del (sample_str)
```

In [88]:
```
bat_str = "I AM BATMAN!"
del (bat_str)
```

In [89]:
```
# to show that the string has been deleted, Let's print it
# --> ERROR
print (sample_str)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[89], line 3
      1 # to show that the string has been deleted, Let's print it
      2 # --> ERROR
----> 3 print (sample_str)

NameError: name 'sample_str' is not defined
```

In [90]:
```
print(bat_str)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[90], line 1
----> 1 print(bat_str)

NameError: name 'bat_str' is not defined
```

7. Operations on Strings

7 .1 Concatenate Strings

Using **+** to ***concatenate*** strings

Run the following code block:

In [91]:
```
str1 = 'Hello'
str2 = ' '
str3 ='World!'
#using+
print('strl + str2 + str3 = ', strl + str2 + str3)
```

```
strl + str2 + str3 =  Hello World!
```

In [94]:
```
_300 = 'This '
_301 = 'is '
_302 = 'Sparta!'
print ('_300 + _301 + _302: ', _300 + _301 + _302)
```
```
_300 + _301 + _302:  This is Sparta!
```

7.2 Replicate Strings

Using * to ***replicate*** a string

Run the following code block:

In [95]:
```
str = "Hello"
replicatedStr = str * 3
print ("The string has been replicated three times: ", replicatedStr)
```
```
The string has been replicated three times:  HelloHelloHello
```

In [97]:
```
pwr_rngr = 'It\'s Morphin time! '
pwr_rngr3 = pwr_rngr * 2
print("Guess what time it is?", pwr_rngr3)
```
```
Guess what time it is? It's Morphin time! It's Morphin time!
```

7.3 Test substrings with "in" & "not in"

Run the following 2 code blocks:

In [98]:
```
str1 = "Welcome"
print("come" in str1)
```
```
True
```

In [99]:
```
gd_brgr = "Welcome to Good Burger, home of the Good Burger"
print("Burger" in gd_brgr)
```
```
True
```

In [100…
```
print("come" not in str1)
```
```
False
```

In [101…
```
print("Mondo" not in gd_brgr)
```
```
True
```

7.4 Compare strings: <, >, <=, >=, ==, !=

Run the following 3 code blocks:

In [102…
```
# TRUE: "apple" comes before "banana"
print("apple" < "banana")
```
```
True
```

In [103...
```
print('cat' < 'dog') #for all the cat lovers
```
```
True
```

In [104...
```
print("apple" < "Apple")
```
```
False
```

In [105...
```
print('bill' < 'Bill')
```
```
False
```

In [106...
```
print("apple" == "Apple")
```
```
False
```

In [107...
```
print('dog' == 'dawg')
```
```
False
```

7.5 Iterate strings using for loops

Run the following 3 code blocks:

In [108...
```
aStr = "Hello"
for i in aStr:
    print(i)
```
```
H
e
l
l
o
```

In [109...
```
easy_str = "string"
for i in simple_str:
    print(i)
```
```
s
t
r
i
n
g
```

In [110...
```
aStr = "Hello"
for i in aStr:
    print(i, end="")
```
```
Hello
```

In [111...
```
med_str = 'string cheese'
for i in med_str:
    print(i, end = " ")
```
```
s t r i n g   c h e e s e
```

In [112...
```
aStr = "Hello"
for i in aStr:
    print(i, end="\n")
```

```
H
e
l
l
o
```

In [113…
```
hard_str = "string wire"
for i in hard_str:
    print(i, end="\n")
```

```
s
t
r
i
n
g
 
w
i
r
e
```

7.6 Test Strings

Method Name	Method Description
isalnum()	Returns "True" if string is alpha-numeric
isalpha()	Returns "True" if string contains only alphabets
isidentifier()	Returns "True" if string is valid identifier
isupper()	Returns "True" if string is in uppercase
islower()	Returns "True" if string is in lowercase
isdigit()	Returns "True" if string only contains digits
isspace()	Returns "True" if string only contains whitespace

****Run the following 7 code blocks:****

In [114…
```
s = "welcome to python"
s. isalnum()
```

Out[114]:
```
False
```

In [129…
```
disarm_str = "expelliarmus"
disarm_str.isalnum()
```

Out[129]:
```
True
```

In [116…
```
"Welcome".isalpha()
```

```
Out[116]: True

In [121…  "Alohomora".isalpha()

Out[121]: True

In [122…  "first Number".isidentifier()

Out[122]: False

In [120…  "Ronald Weasley".isidentifier()

Out[120]: False

In [124…  "WELCOME".isupper()

Out[124]: True

In [125…  "DOBBY".isupper()

Out[125]: True

In [126…  "Welcome".islower()

Out[126]: False

In [127…  "Voldemore".islower()

Out[127]: False

In [128…  s.islower()

Out[128]: True

In [130…  disarm_str.islower()

Out[130]: True

In [131…  " \t". isspace()

Out[131]: True

In [132…  "\b".isspace()

Out[132]: False
```

8. Class string

8.1 count (x)

count(x): return the number of elements of the tuple that are equal to x

****Run the following code block:****

In [133...
```
strl = "This is a string: Hello . . . Hello Python World!"
print (strl.count("Hello"))
```
```
2
```

In [134...
```
gd_brgr_song = "I'm a dude, she's a dude, he's a dude, we're all dudes. Hey!"
print(gd_brgr_song.count("dude"))
```
```
4
```

8.2 index (x)

index(x) returns the index of the first element that is equal to x

****Run the following code block:****

In [135...
```
strl = "This is a string: Hello ... Hello Python World!"
print (strl.index('s'))
```
```
3
```

In [137...
```
bad_joke = "Is it me or did this assignment just string me alone?"
print(bad_joke.index('me'))
```
```
6
```

9. Format Strings

Importatnt Notes:

See the exercises: Formatting Output in Python