Python Data Structures

Sowmya Vajjala

10/9/2020

Let us take a look at some common data structures in Python - strings, lists, dictionaries and tuples. Use of some form of data structures is inevitable in any program. When we are dealing with text files, there should be someway of storing the information we read, so that we can use it for processing later in the program. Think of these data structures as tools that enable that!

Standard notes hold: - I am using either Python console or the project interface in PyCharm. You are free to use what you want (Atom, Spyder, Mac/Unix terminal etc.).

- Instead of copy pasting, try to type the code in this document. There will be a video accompanying this text - I strongly recommend using both together to get a better understanding!
- For a more detailed treatment of this topic, take a look at a few chapters in a Python textbook (Chapters 6–10 in Py4e). Since this is not a Python course per se, I am just quickly going through the basics to get started.

1. Lists

A list is a sequence of values. "Values" can be integers, strings, mix of both, characters, whatever you want. These values in a list are called "elements" or "items". Lists in Python are identified by the presence of square brackets.

```
list1 = [500, 501, 502, 503] #List of integers
list2 = ["Macro Economics", "Labor Economics", "Organizational Economics"] #List of strings
list3 = ['spam', 2.0, 5, [10, 20]] #List containing elements of various data types.
```

Index of the items start at 0, not at 1. So, in the above example, the first item of list1 is accessed as list1[0], not list1[1]. A list can include another list too! Let us try to understand more about lists through examples. Type the following on your console and see the output.

```
complexList = [1, "complex list", ["nested", "item"]]
print(complexList[1])
print(complexList[2])
print(complexList[3])
print(complexList[2][0])
print(complexList[2][1])
print(len(complexList))
print(len(complexList[2]))
anotherList = [1, 2, 3]
print(complexList, anotherList)
```

Lists are "mutable". We can change individual items in the list separately. Try the following commands one by one on your console!

```
numbers = [1, 2, 3, 123]
print(numbers[1])
newNum = numbers[0]
numbers[0] = 2
print(numbers)
```

Traversing a list item by item:

```
numbers = [1, 2, 3, 123]
#One way:
for item in numbers:
  print(item)

#Another way:
for index in range(0,len(numbers)):
  print(index, numbers[i])
```

Why traverse using an "index" variable in the above example? Try to understand through this example below.

```
numbers = [1, 2, 3, 123]
print(numbers)
for index in range(0,len(numbers)):
  numbers[index] = numbers[index]*numbers[index]
print(numbers)
```

List operations:

- operator concatenates lists.

```
a = [1,2,3]
b = [4,5,6]
c = a+b
print(c)
#this gives you:
[1,2,3,4,5,6]

"*" operator repeats a list given number of times.
```

```
print(a*3)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List slicing: accessing and modifying parts of lists using ":" - the slice operator.

```
c = [1,2,3,4,5,6]
print(c[1:3])
c[1:3] = [8,9] #slice operator can be used to change list contents.
```

List "Methods"

- append(): takes one argument and adds it as a new element to the end of the list.
- extend(): takes a list as argument and appends all items in the list to the current list.
- sort(): sorts list elements from low to high. Important note: All these three methods change the list. They do not return the value into a new variable.

To understand this, let us try the following on the console and observe how the list changes after every line.

```
a = [] #empty list.
a.append(4)
a.append(1)
print(a)
print(a.append(4,5))
print(a)
print(a.append([1,2,3]))
print(a)
print(a.extend([1,2,3]))
print(a)
b = [1,4,5,8,0,3]
b.sort()
item print(b)
```

What is the difference between extend and + operator? Figure this out yourself later, by trying what you know so far, by searching online etc.

Deleting elements in a list: if you know the index of the element, use pop(index) or del operator. If you know the element you want to remove (but not the index), you can use remove(). Here is how this works:

```
a = [1, 2, 3, 4, 5, 6]
a.pop()
print(a)
a.pop(1)
print(a)
del(a[1])
print(a)
a.remove(1)
print(a)
```

Lists and Python's built in functions

```
nums = [3, 41, 12, 9, 74, 15]
print(len(nums))
print(max(nums))
print(min(nums))
print(sum(nums))
```

List "Aliasing"

Let us say you want to copy the contents of one list into another. How do you go about that? Let a = [1, 2, 3] be a list. Will it be sufficient if I say b = a in Python? Try these things:

```
a = [1,2,3]
b = a
b[0] = 9
print(a)
```

Did you see what you expected to see?

So, how should I copy from one list to another, then?

```
a = [1,2,3]
firstcopy = list(a)
secondcopy = a[:]
```

There are also other ways, by using "copy" module in python. Find out yourselves!

Lists and Functions - An exercise

Understand this piece of code.

```
def funct1(lst1):
   lst1.sort()
def funct2(lst1):
   lst2 = lst1[:]
   lst2.sort(reverse=True)
   return lst2
lst1 = [9,8,7,12,1,2]
print(funct1(lst1))
print(lst1)
print(funct2(lst1))
print(lst1)
```

Try to do the exercises under Chapter 8 in Py4e!

2. Strings

A string is a sequence of characters - they need not be alphabetic. For example, all the following are strings:

```
str1 = "abc111"
str2 = "abc 111"
str3 = "my name is red"
str4 = "a"
str5 = str1+str4
str6 = str3 + " :)"
```

Strings are "immutable" in Python i.e., it means that you cannot change individual characters in an existing string like we did for a list.

Traversing through a string is like traversing through a list.

```
course = "MGSE Guestcourse"
index = 0
while index<len(course):
    letter = course[index]
    print(letter)
    index = index+1
#Short exercise: Write the for loop equivalent of this while loop.
```

Reverse traversal of a string:

```
course = "MGSE"
i = len(course)
while i>0:
    print(course[i-1])
    i = i-1
```

What is those code below doing?:

```
def countChar(someString, someChar):
   count = 0
   for anyChar in someString:
       #Write your 2 lines of code here.
   return count
stringInput = input("Enter a string: ")
charInput  = input("Enter a char: ")
print("The number of times ", charInput, "occured in", stringInput,
    "is",countChar(stringInput,charInput))
```

Strings can be "sliced" like lists..try these:

```
mystr="Python programming"
print(mystr[5:])
print(mystr[:3])
print(mystr[9:9])
print(mystr[:])
print(mystr[9:3])
print(mystr[-1])
print(mystr[::-1])
```

Some string methods:

Try these on console and understand what they do.

```
example="Some Example String"
print(example.upper())
print(example.lower())
print(example.startswith("S"))
print(example.endswith("S"))
print(example.isdigit())
print(example.find("e"))
print(example.find("e",5))
print(example.find("tri")
```

in is a Python keyword that functions as a boolean operator that takes two strings and returns True if the first appears as a substring in the second".

```
"seed" in "banana" #returns FALSE
"ana" in "banana" #returns TRUE
"a" in "banana" #returns TRUE
```

We can also compare two strings using ==, <, > as with numbers. Try this out yourself, and keep in mind that lower case and upper case characters are treated differently!

What does the below code do?

```
import string
def remove_punctuation(s):
    s_without_punct = ""
    for letter in s:s
        if letter not in string.punctuation:
            s_without_punct += letter
    return s_without_punct
```

```
print(remove_punctuation('"Well, I never did!", said Alice.'))
print(remove_punctuation("Are you very, very, sure?"))
```

Try to understand what these new string methods do:

```
str = 'X-DSPAM-Confidence:  0.8475'
index = str.find(":")
required = str[index+3:]
#because there are two spaces before the number started.
print(float(required))
```

```
def findit(str,substr,num):
    ind = str.find(substr,num)
    return str[ind:]

findit("mississisipi", "ss", 4)

str = 'X-DSPAM-Confidence:  0.8475'
index = str.find(":")
required = str[index+1:].strip()
print(float(required))

str = 'X-DSPAM-Confidence:  0.8475'
newstr = str.replace("X","Y")
print(newstr)
Y-DSPAM-Confidence:  0.8475
```

What is happening here?:

```
def secretFunction(str1,str2):
  str1_lower = str1.lower()
  str2_lower = str2.lower()
  if str1_lower == str2_lower:
     return True
  else:
     return False
print(secretFunction("LaTex","latex"))
print(secretFunction("Nature","Nurture"))

def secretFunction2(str1,str2):
  result = ""
  result = str2[0:2] + str1[2:] + " " + str1[0:2] + str2[2:]
  return result
print(secretFunction2("suntan","sinner"))
print(secretFunction2("whats","that"))
```

Take a look at Python's documentation on strings. Since we are dealing with text in this course, these are going to be very useful data structures.

3. Strings and Lists

```
str = "Look at this code"
demoList = str.split(" ")
#Splits the string wherever there is a space.
```

```
print(demoList)
['Look', 'at', 'this', 'code']
print(len(demoList)
# prints the number of items in a demoList object. 4 here.
print(demoList[1])
# prints "at".
```

Strings as lists of Characters?

You can access individual characters in a string as if it is a list of characters. What do all these give you?

```
course = "MGSE ECONNLP"
course[0]
course[1]
course[1.9]
course[10]
course[-1]
course[-5]
course[-15]
```

Type the following in the Python console and observe the output:

```
string1 = "python"
alist = list(string1) \#converts string to a list
print(alist)
string2 = "this is a longer string"
blist = string2.split()
print(blist)
string3 = "there are strings, lists, int, and float"
clist = string3.split(",") \#, is called the delimiter.
print(clist)
comma = ","
string4 = comma.join(clist)
print(string4)
```

Try to understand this program below:

```
string = input("Enter a paragraph of text:\n")
splitString = string.split(" ")
varN = 5
result = ""
for loopvar in range(0,len(splitString)):
    if loopvar == varN and varN < len(splitString):
        result = result + " " + "____"
        varN = varN +5
    else:
        result = result + " " + splitString[loopvar]
#Question: What does result have now?
print(result)
```

It is very common to view text as a list of words in NLP. So, having a good grasp of strings and lists is very useful for the rest of this coruse!

Tips for using Lists:

- Be aware that some of the list methods manipulate the list and do not return anything lst1.sort() sorts the list lst1 directly..and not save the result as a new list!
- There are multiple ways of doing the same thing with minor differences (+ vs append, l1.sort() vs sorted(l1) etc.). Pick one idiom and stick to that to avoid confusion.
- Make a copy of the list before you use functions like sort(), to avoid aliasing.
- It is easy to fall to write a code that will break easily, especially when we are using split() etc. So, be careful (and patient). Handle possible exceptions, use print statements to do some debugging.

4. Dictionaries, Tuples and Sets

- Dictionaries are a way of storing data as pairs (called: "key-value" pair) in Python. Think of a telephone directory - we access it by names (keys) to get numbers (values).

Here is an Example dictionary structure:

```
mydict = {'Instructor': 'Sowmya', 'Course': 'EconNLP', 'Location': 'Germany', 'Institute':'MGSE'\} \paus
#In this, mydict['Institute'] returns me 'MGSE'.
```

A dictionary can have a dictionary embedded within itself too, just like a list can have a list in it. One good and bad thing about dictionaries: You don't access them one by one sequentially. i.e., you cannot do eg[1] etc. One example of where they are useful is to build a word frequency list from a corpus.

We will see more on how they work as we progress with other stuff. Here is python's official tutorial on dictionaries.

Tuples are another datastructure in Python, which in some ways has some similarities with both lists and dictionaries. More details here. Finally, another data structure you may come across is sets.

6. Connecting all this back to Files:

Lists and Files:

What is the potential problem with this code:

```
fhand = open('mbox-short.txt')
for line in fhand:
    words = line.split()
    if words[0] != 'From':
        continue
    print(words[2])
```

Strings and Files:

Searching through a file:

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

```
#One more example:
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1 :
        continue
    print(line)
#This will print only lines that have that string "@uct.ac.za"
```

We will see more examples of searching through text in the next topic (and the next).

I will end my python crash course here. We've roughly covered the first 10 chapters of Py4E in these two documents!! Try to go through these, spend some time and get familiar with Python, and prepare yourself for more python (and more fun?)!