Session-15

September 13, 2019

1 Session-15

Dated 13-09-2019, 08:00 to 09:00 Hrs, RJN 302

2 Data types and data structures

We illustrate the use of strings, lists, tuples and dictionaries in this session

2.1 Strings

You can create a sequence of type string by enclosing the characters between quotes.

```
In [1]: s1 = "Here is a string"
```

The builtin function len gives the number of items in the sequence. For a string, it will be the number of characters.

```
In [2]: len(s1)

Out[2]: 16
```

Accessing the individual items in the sequence is by using square brackets. The numbering starts with 0.

```
In [3]: s1[3]

Out[3]: 'e'
```

One can take a slice of a sequence using two integers that indicate start and end positions within the sequence. Start is included and end is not. One can use negative numbers to indicate the position as referred from the end of the sequence.

```
In [4]: s1[0:4]

Out[4]: 'Here'

In [81]: s1[4:-2]

Out[81]: ' is a stri'
```

To see what are the functions available to work on the string, look it up using the dir function.

```
In [6]: dir(s1)
```

```
Out[6]: ['__add__',
         '__class__',
         '__contains__',
         '__delattr__',
         '__doc__',
         '__eq__',
         '__format__',
         '__ge__',
         '__getattribute__',
         '__getitem__',
         '__getnewargs__',
         '__getslice__',
         '__gt__',
         '__hash__',
         '__init__',
         '__le__',
         '__len__',
         '__lt__',
         '__mod__',
         '__mul__',
         '__ne__',
         '__new__',
         '__reduce__',
         '__reduce_ex__',
         '__repr__',
         '__rmod__',
         '__rmul__',
         '__setattr__',
         '__sizeof__',
         '__str__',
         '__subclasshook__',
         '_formatter_field_name_split',
         '_formatter_parser',
         'capitalize',
         'center',
         'count',
         'decode',
         'encode',
         'endswith',
         'expandtabs',
         'find',
         'format',
         'index',
         'isalnum',
```

```
          'isalpha',
          'isdigit',
          'islower',
          'isspace',
          'istitle',
          'isupper',
          'join',
          'ljust',
          'lower',
          'lstrip',
          'partition',
          'replace',
          'rfind',
          'rindex',
          'rjust',
          'rpartition',
          'rsplit',
          'rstrip',
          'split',
          'splitlines',
          'startswith',
          'strip',
          'swapcase',
          'title',
          'translate',
          'upper',
          'zfill']

In [8]: s1.lower()

Out[8]: 'here is a string'

In [9]: s1.upper()

Out[9]: 'HERE IS A STRING'

In [10]: s1.swapcase()

Out[10]: 'hERE IS A STRING'
```

The plus operator works to concatenate the sequence.

```
In [85]: s2 = "...This is also a string"
         s1 + s2

Out[85]: 'Here is a string...This is also a string'
```

2.2 Lists

One can create a list using square brackets. The items can be of any type and can be mixed up too.

```
In [11]: a1=[1,2,3,4,5]

In [12]: len(a1)

Out[12]: 5
```

The range function gives out a list of integers starting from 0 until the number provided. In newer versions of python, this works differently as range() returns a separate data type. You can pass it to the creator of list and get the same output as in older versions. If the following command returns an output that does not look like a list of numbers from 0 to 9, then use the command below that.

```
In [13]: range(10)

Out[13]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [90]: list(range(0,10,1))

Out[90]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [14]: help(range)

Help on built-in function range in module __builtin__:

range(...)
    range(stop) -> list of integers
    range(start, stop[, step]) -> list of integers

    Return a list containing an arithmetic progression of integers.
    range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.
    When step is given, it specifies the increment (or decrement).
    For example, range(4) returns [0, 1, 2, 3].  The end point is omitted!
    These are exactly the valid indices for a list of 4 elements.



In [88]: a1=list(range(5,100,5))

In [89]: a1

Out[89]: [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]

In [18]: r1=[1.1, 2.2, 3.3, 4.4, 5.5]

In [19]: r1

Out[19]: [1.1, 2.2, 3.3, 4.4, 5.5]
```

```
In [20]: c1=['a', 'b', 'c', 'd']

In [21]: c1

Out[21]: ['a', 'b', 'c', 'd']
```

Lists can be made of a mixture of different types of items.

```
In [22]: m1=[1, 1.1, '1']

In [23]: m1

Out[23]: [1, 1.1, '1']
```

Addition operator works to achieve concatenation of the lists. The behavior is different in other languages such as octave.

```
In [24]: j1 = r1 + c1

In [25]: j1

Out[25]: [1.1, 2.2, 3.3, 4.4, 5.5, 'a', 'b', 'c', 'd']

In [91]: type(j1)

Out[91]: list
```

You can check if a value is there in a list or not. The output is boolean.

```
In [26]: 'b' in j1

Out[26]: True

In [27]: 'x' in j1

Out[27]: False
```

You can remove items from a list using the function that comes along with the list object.

```
In [29]: j1.remove(4.4)

In [30]: j1

Out[30]: [1.1, 2.2, 3.3, 5.5, 'a', 'b', 'c', 'd']

In [31]: j1[3]

Out[31]: 5.5
```

An elegant error message is thrown out when you try to delete an item that does not exist in a list.

```
In [33]: j1.remove(3.1415)

         ---------------------------------------------------------------------------

         ValueError                                Traceback (most recent call last)

         <ipython-input-33-68d3e0d9725f> in <module>()
    ----> 1 j1.remove(3.1415)


         ValueError: list.remove(x): x not in list
```

The builtin functions min and max will pick items as per the ascii sequence or numberic value as appropriate.

```
In [34]: min(j1)

Out[34]: 1.1

In [35]: max(j1)

Out[35]: 'd'
```

Taking a slice of a list is same as in a string.

```
In [36]: j1[1:-2]

Out[36]: [2.2, 3.3, 5.5, 'a', 'b']
```

2.3 Tuples

Tuples are created using parantheses. The items inside a tuple are not mutable. Otherwise a tuple is very much like a list.

```
In [37]: t1 = (1.0, 2.0, 'a', 'b')

In [38]: t1

Out[38]: (1.0, 2.0, 'a', 'b')

In [39]: type(t1)

Out[39]: tuple
```

You can modify value of an item in a list but not of an item in a tuple.

```
In [42]: j1[1]=2.3

In [43]: j1
```

```
Out[43]: [1.1, 2.3, 3.3, 5.5, 'a', 'b', 'c', 'd']
```

If you try to change an item of a tuple, you will get an error as shown below.

```
In [44]: t1[1]=2.2


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-44-70bd7f27b4ea> in <module>()
    ----> 1 t1[1]=2.2


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

2.4 Dictionaries

Dictionaries are associative arrays or hashes. They are a bunch of key,value pairs with no particular sequence of arrangement. One can create a dictionary using flower brackets. The key,value pair can be given using a colon that separates the key and value. You can also give these values using the dict creator function.

```
In [46]: d1 = {'a':1.0, 'b':2.0}

In [47]: type(d1)

Out[47]: dict

In [95]: d2 = dict(c=3.0, d=4.0)

In [96]: d2

Out[96]: {'c': 3.0, 'd': 4.0}

In [97]: type(d2)

Out[97]: dict

In [48]: d1.values()

Out[48]: [1.0, 2.0]

In [49]: d1.keys()

Out[49]: ['a', 'b']

In [50]: type(d1.keys())
```

```
Out[50]: list

In [51]: type(d1.values())

Out[51]: list
```

New items can get added to the dictionary as you create new pairs of key, value.

```
In [52]: d1['pi'] = 3.1415

In [53]: d1

Out[53]: {'a': 1.0, 'b': 2.0, 'pi': 3.1415}
```

One can inquire if there is a particular key in the dictionary. This complements the "in" functionality of lists. In the newer version of python, this function has_key() is depracated. One can use the "in" functionality for dictionaries too.

```
In [103]: 'a' in d1

Out[103]: True

In [54]: d1.has_key('a')

Out[54]: True

In [55]: d1.has_key('z')

Out[55]: False

In [56]: d1['a']

Out[56]: 1.0
```

Dictionary can contain items that are lists too.

```
In [57]: d1['mylist']=[1.0, 2.0]

In [58]: d1

Out[58]: {'a': 1.0, 'b': 2.0, 'mylist': [1.0, 2.0], 'pi': 3.1415}
```

Items in a dictionary can be lists themselves. Such lists need not be of uniform length across items in the dictionary. We enclose range function inside a list constructor to avoid issues with new version of python.

```
In [101]: d1['mylonglist'] = list(range(5,25))

In [102]: d1
```

```
Out[102]: {'a': 1.0,
           'b': 2.0,
           'mylist': [1.0, 2.0],
           'mylonglist': [5,
            6,
            7,
            8,
            9,
            10,
            11,
            12,
            13,
            14,
            15,
            16,
            17,
            18,
            19,
            20,
            21,
            22,
            23,
            24],
           'pi': 3.1415}

In [62]: dir(d1)

Out[62]: ['__class__',
          '__cmp__',
          '__contains__',
          '__delattr__',
          '__delitem__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__gt__',
          '__hash__',
          '__init__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__ne__',
          '__new__',
          '__reduce__',
```

```
'__reduce_ex__',
'__repr__',
'__setattr__',
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'clear',
'copy',
'fromkeys',
'get',
'has_key',
'items',
'iteritems',
'iterkeys',
'itervalues',
'keys',
'pop',
'popitem',
'setdefault',
'update',
'values',
'viewitems',
'viewkeys',
'viewvalues']
```

The temporary variable to be used for iteration across all keys can be of any name. The function keys() provided along with the object of type dictionary returns a list that helps the for loop run over. The following may not work in newer version of python.

```
In [98]: for chabi in d1.keys():
             print chabi, "=>", d1[chabi]

a => 1.0
pi => 3.1415
b => 2.0
mylist => [1.0, 2.0]
mylonglist => [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

In the newer version of python, you can enclose the output of keys in list. The following should work for new version of python too.

```
In [99]: for chabi in list(d1.keys()):
             print chabi, "=>", d1[chabi]

a => 1.0
pi => 3.1415
b => 2.0
```

```
mylist => [1.0, 2.0]
mylonglist => [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

3 Looping

The variable used for iterating a for loop is temporary. We are enclosing the range function inside list constructor to avoid issues with new version of python.

```
In [100]: print("Number of items in the list j1:", len(j1))

          for i in list(range(0,len(j1))):
              print(i, "=>", j1[i])

          print("Now we are out of loop")
```

```
('Number of items in the list j1:', 8)
(0, '=>', 1.1)
(1, '=>', 2.3)
(2, '=>', 3.3)
(3, '=>', 5.5)
(4, '=>', 'a')
(5, '=>', 'b')
(6, '=>', 'c')
(7, '=>', 'd')
Now we are out of loop
```

4 Functions

Function definition starts with the keyword def followed by the name of the function, the arguments in parantheses and a colon. The body of the function shall be indented to convey that is a block of code that is part of the function definition.

```
In [72]: def myfunction(x):
             return x*x+5.0

In [73]: myfunction(2.0)

Out[73]: 9.0

In [ ]:
```