Data science and Machine learning with Python

Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

Python collections:

Introduction

Collections in Python are containers that are used to store collections of data, for example, **list, dict, set, tuple** etc. These are built-in collections. Several modules have been developed that provide additional data structures to store collections of data.

Lists

In **Python**, List is the most flexible data type. It is a sequence of elements, which allows you to remove or add the elements to list and allows you to slice the elements.

In order to write a list, you need to put the elements into a pair of square brackets [] and separate them by the comma. The elements in the list are indexed from the left to right and started from the index 0.

Demo:

```
fruitList = ["apple", "apricot", "banana","coconut", "lemen"]
 otherList = [100, "one", "two", 3]

 print ("Fruit List:")
 print (fruitList)
 print (" -------------------------- ")
 print ("Other List:")
 print (otherList)
```

Output:

```
Fruit List:
['apple', 'apricot', 'banana', 'coconut', 'lemen']
 --------------------------
```

Other List:

```
[100, 'one', 'two', 3]
```

Access the elements

Use the **'for'** loop to access the elements of list:

Demo:

```
fruitList = ["apple", "apricot", "banana", "coconut", "lemen", "plum", "pear"]
for fruit in fruitList :
        print ("Fruit: ", fruit)
```


Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

Output:

```
Fruit:  apple
Fruit:  apricot
Fruit:  banana
Fruit:  coconut
Fruit:  lemen
Fruit:  plum
Fruit:  pear
```

Access via index:

You can also access the elements of list via index. The elements of list are indexed from left to right and started by 0.

Demo:

```
fruitList = ["apple", "apricot", "banana", "coconut", "lemen", "plum", "pear"]

print ( fruitList )
# Element count.
print ("Element count: ", len(fruitList) )

for i in range (0, len(fruitList) ) :
    print ("Element at ", i, "= ", fruitList[i] )

# A sub list of elements with index from 1 to 4 (1, 2, 3)
subList = fruitList[1: 4]

# ['apricot', 'banana', 'coconut']
print ("Sub List [1:4] ", subList )
```

Output:

```
['apple', 'apricot', 'banana', 'coconut', 'lemen', 'plum', 'pear']
Element count:  7
Element at  0 =  apple
Element at  1 =  apricot
Element at  2 =  banana
Element at  3 =  coconut
Element at  4 =  lemen
Element at  5 =  plum
Element at  6 =  pear
Sub List [1:4]  ['apricot', 'banana', 'coconut']
```

You can also access the elements of list by negative index, the elements are indexed from right to left with the value -1, -2,...

Demo:

```
fruitList = ["apple", "apricot", "banana", "coconut", "lemen", "plum", "pear"]

print ( fruitList )

print ("Element count: ", len(fruitList) )
print ("fruitList[-1]: ", fruitList[-1])
print ("fruitList[-2]: ", fruitList[-2])

subList1 = fruitList[-4: ]

print ("\n")
print ("Sub List fruitList[-4: ] ")
print (subList1)

subList2  = fruitList[-4:-2]

print ("\n")
print ("Sub List fruitList[-4:-2] ")
print (subList2)
```

Output:

```
['apple', 'apricot', 'banana', 'coconut', 'lemen', 'plum', 'pear']
Element count:  7
fruitList[-1]:  pear
fruitList[-2]:  plum


Sub List fruitList[-4: ]
['coconut', 'lemen', 'plum', 'pear']

Sub List fruitList[-4:-2]
['coconut', 'lemen']
```

Update list:

Demo:

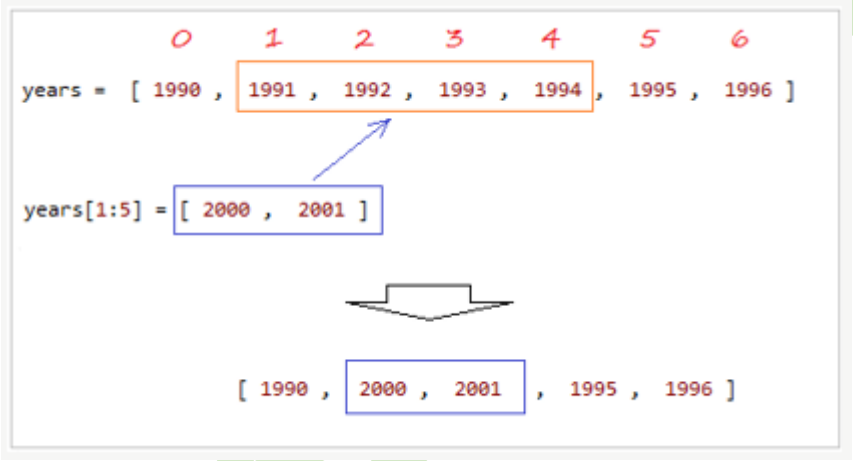
```
years = [1991,1995, 1992]

print ("Years: ", years)

print ("Set years[1] = 2000")

years[1] = 2000
```

```
print ("Years: ", years)

print ( years )

print ("Append element 2015, 2016 to list")

# Append an element to the end of the list.
years.append( 2015  )
years.append( 2016  )

print ("Years: ", years)
```

Output:

```
Years:  [1991, 1995, 1992]
Set years[1] = 2000
Years:  [1991, 2000, 1992]
[1991, 2000, 1992]
Append element 2015, 2016 to list
Years:  [1991, 2000, 1992, 2015, 2016]
```

You can also update the value to a slice of elements. This is a way helping you to update the multiple elements at a time.:



Demo:

```
years =  [ 1990 ,  1991 ,  1992 ,  1993 ,  1994 ,  1995 ,  1996 ]
print ("Years: ", years)
print ("Update Slice: years[1:5] = [2000, 2001]")
years[1:5] = [ 2000 ,  2001 ]
print ("Years: ", years)
```

Output:

```
Years:  [1990, 1991, 1992, 1993, 1994, 1995, 1996]
Update Slice: years[1:5] = [2000, 2001]
Years:  [1990, 2000, 2001, 1995, 1996]
```

Delete the elements in the list

In order to delete one or more elements in a list, you can use the **del** statement, or use the **remove()** method. The example below uses the **del** statement to delete one or more elements by index.

Demo:

```
years = [ 1990 , 1991 , 1992 , 1993 , 1994 , 1995 , 1996 ]
print ("Years: ", years)
print ("\n del years[6]")

# Delete element at index = 6.
del years[6]

print ("Years: ", years)
print ("\n del years[1:4]")

# Delete element at index = 1, 2,3
del years[1:4]

print ("Years: ", years)
```

Output:

```
Years:  [1990, 1991, 1992, 1993, 1994, 1995, 1996]

 del years[6]
Years:  [1990, 1991, 1992, 1993, 1994, 1995]

 del years[1:4]
Years:  [1990, 1994, 1995]
```

The **remove(value)** method removes the first element in list which has value the same as the value of parameter. The method can throw exception if no element is found to remove.

Demo:

```
years = [ 1990 , 1991 , 1992 , 1993 , 1994 , 1993 , 1993 ]

print ("Years: ", years)


print ("\n years.remove(1993)")

# Remove first occurrence of value.
years.remove(1993)

print ("Years: ", years)
```

Output:
Years: [1990, 1991, 1992, 1993, 1994, 1993, 1993]

years.remove(1993)
Years: [1990, 1991, 1992, 1994, 1993, 1993]

Like String, List has three operators including **+, *, in.**

Operator	Description	Example
+	The operator is used to concatenate 2 Lists to create a new List	[1, 2, 3[+ ["One","Two"] --> [1, 2, 3, "One", "Two"]
*	The operator is used to concatenate multiple copies of the same List. And create a new List	[1, 2] * 3 --> [1, 2, 1, 2, 1, 2]
in	Check whether an element is in a List, which returns True or False.	"Abc" in ["One", "Abc"] --> True

Ex:

```
list1 = [1, 2, 3]

list2 = ["One", "Two"]

print ("list1: ", list1)
print ("list2: ", list2)
print ("\n")

list12 = list1 + list2

print ("list1 + list2: ", list12)


list2x3 = list2 * 3

print ("list2 * 3: ", list2x3)


hasThree = "Three" in list2

print ("'Three' in list2? ", hasThree)
```

Re:

list1: [1, 2, 3]
list2: ['One', 'Two']

list1 + list2: [1, 2, 3, 'One', 'Two']
list2 * 3: ['One', 'Two', 'One', 'Two', 'One', 'Two']
'Three' in list2? False

Functions of List

Function	Description
len(list)	Gives the total length of the list.
max(list)	Returns item from the list with max value.
min(list)	Returns item from the list with min value.
list(seq)	Converts a tuple into list.

Ex:

```
list1 = [1991, 1994, 1992]

list2 = [1991, 1994, 2000, 1992]

print ("list1: ", list1)
print ("list2: ", list2)

# Return element count.
print ("len(list1): ", len(list1) )
print ("len(list2): ", len(list2) )

# Maxinum value in the list.
maxValue = max(list1)

print ("Max value of list1: ", maxValue)


# Minimum value in the list
minValue = min(list1)

print ("Min value of list1: ", minValue)
```

```
# Tuple
tuple3 = (2001, 2005, 2012)

print ("tuple3: ", tuple3)

# Convert a tuple to a list.
list3 = list (tuple3)

print ("list3: ", list3)
```

RE:

```
list1:  [1991, 1994, 1992]
list2:  [1991, 1994, 2000, 1992]
len(list1):  3
len(list2):  4
Max value of list1:  1994
Min value of list1:  1991
tuple3:  (2001, 2005, 2012)
list3:  [2001, 2005, 2012]
```

Methods of List

Method	**Description**
list.append(obj)	Appends object obj to list
list.count(obj)	Returns count of how many times obj occurs in list
list.extend(seq)	Appends the contents of seq to list
list.index(obj)	Returns the lowest index in list that obj appears
list.insert(index, obj)	Inserts object obj into list at offset index
list.pop([index])	If has index parameter, remove and return the element at index position. Else, remove and return the last element of the list.
list.remove(obj)	Removes object obj from list
list.reverse()	Reverses objects of list in place
list.sort(key=None, reverse=False)	Sorts elements of list by key in parameters

Ex:

```
years = [ 1990 ,  1991 ,  1992 ,  1993 ,  1993 ,  1993 ,  1994 ]

print ("Years: ", years)

print ("\n - Reverse the list")

# Reverse the list.
years.reverse()

print ("Years (After reverse): ", years)

aTuple = (2001, 2002, 2003)

print ("\n - Extend: ", aTuple)
years.extend(aTuple)

print ("Years (After extends): ", years)

print ("\n - Append 3000")
years.append(3000)

print ("Years (After appends): ", years)

print ("\n - Remove 1993")
years.remove(1993)

print ("Years (After remove): ", years)


print ("\n - years.pop()")

# Remove last element of list.
lastElement = years.pop()

print ("last element: ", lastElement)

print ("\n")

# Count
print ("years.count(1993): ",years.count(1993) )
```

Re:
Years: [1990, 1991, 1992, 1993, 1993, 1993, 1994]

- Reverse the list
Years (After reverse): [1994, 1993, 1993, 1993, 1992, 1991, 1990]

- Extend: (2001, 2002, 2003)
Years (After extends): [1994, 1993, 1993, 1993, 1992, 1991, 1990, 2001, 2002, 2003]

- Append 3000
Years (After appends): [1994, 1993, 1993, 1993, 1992, 1991, 1990, 2001, 2002, 2003, 3000]

- Remove 1993
Years (After remove): [1994, 1993, 1993, 1992, 1991, 1990, 2001, 2002, 2003, 3000]

- years.pop()
last element: 3000

years.count(1993): 2

Tuples
In **Python**, Tuples is a sequence of values containing many elements, which is similar to **List**. Unlike List, **Tuples** is an immutable data type, all updates on the Tuple create a new entity on the memory.

In order to write a Tuple, you need to put the elements into parentheses ( ) and separated by comma. These elements of the list are indexed with the starting index 0.

Ex:
```
fruitTuple = ("apple", "apricot", "banana","coconut", "lemen")

otherTuple = (100, "one", "two", 3)

print ("Fruit Tuple:")

print (fruitTuple)

print (" ------------------------- ")

print ("Other Tuple:")

print (otherTuple)
```

Re:
Fruit Tuple:
('apple', 'apricot', 'banana', 'coconut', 'lemen')

Other Tuple:
(100, 'one', 'two', 3)

List vs Tuple
List and Tuple are a sequence of elements. There are some following differences between them:

Writing a list, you need to use square brackets [] while writing a Tuple, you use parentheses ( ).

```
# This is a Tuple.
aTuple = ("apple", "apricot", "banana")

# This is a List.
aList =  ["apple", "apricot", "banana"]
```

List is a mutable data type, you can use the **append()** method to add the element to List, or use the **remove()** method to remove elements from List without creating another "List" entity in memory.

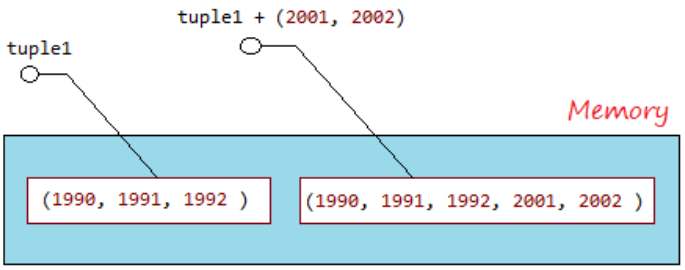
Ex:

```
list1 =  [1990, 1991, 1992]

print ("list1: ", list1)

# Address of list1 in the memory.
list1Address = hex ( id(list1) )

print ("Address of list1: ", list1Address )

print ("\n")
print ("Append element 2001 to list1")

# Append a element to list1.
list1.append(2001)

print ("list1 (After append): ", list1)

# Address of list1 in the memory.
list1Address = hex ( id(list1) )

print ("Address of list1 (After append): ", list1Address )
```

Re:
list1: [1990, 1991, 1992]
Address of list1: 0x1bda849f648

Append element 2001 to list1
list1 (After append): [1990, 1991, 1992, 2001]
Address of list1 (After append): 0x1bda849f648

Tuple is an immutable object, it does not have the methods **append (), remove ()**,... like list. You may think that some methods, or operators are used to update **Tuple** but the fact that it is based on the original Tuple to create a new Tuple.



Ex:

tuple1 **=** (1990, 1991, 1992)

Address of tuple1 in the memory.
tuple1Address **=** hex (id(tuple1))

print ("Address of tuple1: ", tuple1Address)

Concatenate a tuple to tuple1.
tuple1 **=** tuple1 **+** (2001, 2002)

Address of tuple1 in the memory.
tuple1Address **=** hex (id(tuple1))

print ("Address of tuple1 (After concat): ", tuple1Address)

Re:

Address of tuple1: 0x21fb266b2c8

Address of tuple1 (After concat): 0x21fb2685fa8

Access the elements of Tuples

Access the elements

Use the **'for'** loop to access the elements of Tuple:

Ex:

```
fruits = ("apple", "apricot", "banana", "coconut", "lemen", "plum", "pear")

for fruit in fruits :

    print ("Fruit: ", fruit)
```

re:

```
Fruit:  apple
Fruit:  apricot
Fruit:  banana
Fruit:  coconut
Fruit:  lemen
Fruit:  plum
Fruit:  pear
```

Access via index:

You can also access the elements of Tuple via index. The elements of Tuple are indexed from left to right, starting at 0.

Ex:

```
fruits = ("apple", "apricot", "banana", "coconut", "lemen", "plum", "pear")

print ( fruits )

# Element count.
print ("Element count: ", len(fruits) )


for i in range (0, len(fruits) ) :

    print ("Element at ", i, "= ", fruits[i] )
```

```
# A sub Tuple of elements with index from 1 to 4 (1, 2, 3)
subTuple = fruits[1: 4]

# ('apricot', 'banana', 'coconut')
print ("Sub Tuple [1:4] ", subTuple )
```

Re:

```
('apple', 'apricot', 'banana', 'coconut', 'lemen', 'plum', 'pear')
Element count:  7
Element at  0 =  apple
Element at  1 =  apricot
Element at  2 =  banana
Element at  3 =  coconut
Element at  4 =  lemen
Element at  5 =  plum
Element at  6 =  pear
Sub Tuple [1:4]  ('apricot', 'banana', 'coconut')
```

You can also access the elements of Tuple via negative index, the elements are indexed from right to left with the values -1, -2,...

Ex:

```
fruits = ("apple", "apricot", "banana", "coconut", "lemen", "plum", "pear")

print ( fruits )

print ("Element count: ", len(fruits) )


print ("fruits[-1]: ", fruits[-1])
print ("fruits[-2]: ", fruits[-2])


subTuple1 = fruits[-4: ]

print ("\n")
print ("Sub Tuple fruits[-4: ] ")
print (subTuple1)


subTuple2  = fruits[-4:-2]

print ("\n")
print ("Sub Tuple fruits[-4:-2] ")
print (subTuple2)
```

Re:
('apple', 'apricot', 'banana', 'coconut', 'lemen', 'plum', 'pear')
Element count: 7
fruits[-1]: pear
fruits[-2]: plum

Sub Tuple fruits[-4:]
('coconut', 'lemen', 'plum', 'pear')

Sub Tuple fruits[-4:-2]
('coconut', 'lemen')

Update Tuples
Note that Tuple is immutable, so it only has methods or operators to access, or create a new **Tuple** from the original **Tuple**.

Ex:

```
tuple1 = (1990, 1991, 1992)

print ("tuple1: ", tuple1)


print ("Concat (2001, 2002) to tuple1")


tuple2 = tuple1 + (2001, 2002)


print ("tuple2: ", tuple2)

# A sub Tuple, containing elements from index 1 to 4 (1,2,3)
tuple3 = tuple2[1:4]

print ("tuple2[1:4]: ", tuple3)

# A sub Tuple, containing elements from index 1 to the end.
tuple4 = tuple2[1: ]

print ("tuple2[1: ]: ", tuple4)
```

Re:
tuple1: (1990, 1991, 1992)
Concat (2001, 2002) to tuple1
tuple2: (1990, 1991, 1992, 2001, 2002)
tuple2[1:4]: (1991, 1992, 2001)
tuple2[1:]: (1991, 1992, 2001, 2002)

Operators for Tuples

Like String, Tuple has 3 operators including **+, * , in.**

Operator	**Description**	**Example**
+	The operator is used to concatenate 2 Tuples to create a new Tuple	(1, 2, 3) + ("One","Two") --> (1, 2, 3, "One", "Two")
*	The operator is used to concatenate multiple copies of the same Tuple. And create a new Tuple	(1, 2) * 3 --> (1, 2, 1, 2, 1, 2)
in	Check whether an element is in a Tuple, which returns True or False.	"Abc" in ("One", "Abc") --> True

Ex:

```
tuple1 = (1, 2, 3)

tuple2 = ("One", "Two")

print ("tuple1: ", tuple1)
print ("tuple2: ", tuple2)
print ("\n")

tuple12 = tuple1 + tuple2

print ("tuple1 + tuple2: ", tuple12)

tuple2x3 = tuple2 * 3

print ("tuple2 * 3: ", tuple2x3)

hasThree = "Three" in tuple2

print ("'Three' in tuple2? ", hasThree)
```

Re:
tuple1: (1, 2, 3)
tuple2: ('One', 'Two')

tuple1 + tuple2: (1, 2, 3, 'One', 'Two')
tuple2 * 3: ('One', 'Two', 'One', 'Two', 'One', 'Two')
'Three' in tuple2? False

Functions for Tuples:

Function	**Description**
len(list)	Gives the total length of the Tuple.
max(list)	Returns item from the Tuple with max value.
min(list)	Returns item from the Tuple with min value.
tuple(seq)	Converts a List into Tuple.

Ex:

```
tuple1 = (1991, 1994, 1992)

tuple2 = (1991, 1994, 2000, 1992)

print ("tuple1: ", tuple1)
print ("tuple2: ", tuple2)

# Return element count.
print ("len(tuple1): ", len(tuple1) )
print ("len(tuple2): ", len(tuple2) )

# Maxinum value in the Tuple.
maxValue = max(tuple1)

print ("Max value of tuple1: ", maxValue)

# Minimum value in the Tuple.
minValue = min(tuple1)

print ("Min value of tuple1: ", minValue)

# (all)
# List
list3 = [2001, 2005, 2012]

print ("list3: ", list3)

# Convert a List to a tuple.
```

```
tuple3 = tuple (list3)

print ("tuple3: ", tuple3)
```

Re:

```
tuple1:  (1991, 1994, 1992)
tuple2:  (1991, 1994, 2000, 1992)
len(tuple1):  3
len(tuple2):  4
Max value of tuple1:  1994
Min value of tuple1:  1991
list3:  [2001, 2005, 2012]
tuple3:  (2001, 2005, 2012)
```

Methods

Method	**Description**
tuple.count(obj)	Returns count of how many times obj occurs in Tuple
tuple.index(obj, [start, [stop]])	Returns the lowest index in Tuple that obj appears. Raises **ValueError** if the value not found. If there are parameters **start**, **stop**, just search from **start** to **stop** index (And not included **stop**).

Ex:

```
years = ( 1990 ,  1991 ,  1993 ,  1991 ,  1993 ,  1993 ,  1993 )

print ("Years: ", years)


print ("\n")

# return number of occurrences of 1993
print ("years.count(1993): ",years.count(1993) )

# Find the index that appears 1993
print ("years.index(1993): ", years.index(1993) )

# Find the index that appears 1993, from index 3
print ("years.index(1993, 3): ", years.index(1993, 3) )

# Find the index that appears 1993, from index 4 to 5 (Not included 6)
print ("years.index(1993, 4, 6): ", years.index(1993, 4, 6) )
```

Re:
Years: (1990, 1991, 1993, 1991, 1993, 1993, 1993)

years.count(1993): 4
years.index(1993): 2
years.index(1993, 3): 4
years.index(1993, 4, 6): 4