Lists, Tuples and Dictionary

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Python

Lists

- Are the most versatile datatype available in Python.
- Which can be written as a list of comma-separated values (items) between square brackets.
- Important thing is that the items in a list need not be of the same type.
- Creating a list is as simple as putting different comma-separated values between square brackets.

```
list1= ['physics', 'chemistry', 1997, 2000]
```

Basic List Operations

- Lists respond to the + and * operators much like strings.
- Mean concatenation and repetition here too, except that the result is a new list, not a string.
- In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

Basic List Operations (Cont.)

Python Expression	Results	Description
`len([1,2,3])`	3	Length
`[1,2,3] + [4,5,6]`	[1,2,3,4,5,6]	Concatenation
`['Hi!' ] * 4`	[‘Hi!’, ‘Hi!’, ‘Hi!’, ‘Hi!’]	Repetition
`3 in [1,2,3]`	True	Membership
`for x in [1,2,3] : print (x ,end =' ')`	1 2 3	Iteration

Indexing, Slicing and Matrixes

- Since lists are sequences, indexing and slicing work the same way for lists as they do for strings.

```
L = ['C++', 'Java' , 'Python']
```

Python Expression	Results	Description
`L[2]`	'Python'	Offsets start at zero
`L[-2]`	'Java'	Negative : count from the right
`L[1:]`	['Java', 'Python']	Slicing fetches sections

Built-in List Functions and Methods

- `cmp(list1, list2)`
  - No longer available in Python 3.

Built-in List Functions and Methods (Cont.)

- `len(list)`
  - Gives the total length of the `list`.

Built-in List Functions and Methods (Cont.)

- **`max(list)`**
  - Returns item from the `list` with max value.

Built-in List Functions and Methods (Cont.)

- `min(list)`
  - Returns item from the `list` with min value.

Built-in List Functions and Methods (Cont.)

- `list(seq)`
  - Converts a tuple into list.

List Methods

- `list.append(obj)`
  - Appends object `obj` to list.

List Methods (Cont.)

- `list.count(obj)`
  - Returns count of how many times `obj` occurs in list.

List Methods (Cont.)

- `list.extend(seq)`
  - Appends the contents of `seq` to list.

List Methods (Cont.)

- `list.index(obj)`
  - Returns the lowest index in list that `obj` appears.

List Methods (Cont.)

- `list.insert(index, obj)`
  - Inserts object `obj` into list at offset `index`.

List Methods (Cont.)

- `list.pop(obj = list[-1])`
  - Removes and returns last object or `obj` from `list`.

List Methods (Cont.)

- `list.remove(obj)`
  - Removes object `obj` from list.

List Methods (Cont.)

- `list.reverse()`
  - Reverses objects of list in place.

List Methods (Cont.)

- `list.sort([func])`
  - Sorts objects of list, use compare `func` if given.

List - Lab

```
1 strdata = 'Time is money!!'
2 listdata = [1, 2, [1, 2, 3]]
3 print(strdata[5])      # ‘i'가 출력됨
4 print(strdata[-2])     # ‘!’가 출력됨
5 print(listdata[0])      # 1이 출력됨
6 print(listdata[-1])     # [1, 2, 3]이 출력됨
7 print(listdata[2][-1])  # 3이 출력됨
8
```

List – Lab (Cont.)

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성', '지구']
2 planet = '지구'
3 pos = solarsys.index(planet)
4 print('%s은(는) 태양계에서 %d번째에 위치하고 있습니다.' %(planet, pos))
5 pos = solarsys.index(planet, 5)
6 print('%s은(는) 태양계에서 %d번째에 위치하고 있습니다.' %(planet, pos))
7
```

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 planet = '화성'
3 pos = solarsys.index(planet)
4 solarsys [pos] = 'Mars'
5 print(solarsys)
6
```

List – Lab (Cont.)

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 rock_planets = solarsys[1:4]
3 gas_planets = solarsys[4:]
4 print('태양계의 암석형 행성: ', end='');print(rock_planets)
5 print('태양계의 가스형 행성: ', end='');print(gas_planets)
6
```

List – Lab (Cont.)

```
1 listdata = list(range(1, 21))
2 evenlist = listdata[1::2]
3 print(evenlist)
4
```

```
1 listdata = list(range(5))
2 listdata.reverse()
3 print(listdata)  # [4, 3, 2, 1, 0]이 출력됨
4
```

List – Lab (Cont.)

```
1 listdata = list(range(5))
2 ret1 = reversed(listdata)
3 print('원본 리스트 ', end='');print(listdata);
4 print('역순 리스트 ', end='');print(list(ret1))
5
6 ret2 = listdata[::-1]
7 print('슬라이싱 이용 ', end='');print(ret2)
```

List – Lab (Cont.)

```
1 listdata = list(range(3))
2 ret = listdata*3
3 print(ret)   # [0, 1, 2, 0, 1, 2, 0, 1, 2]가 출력됨
4
```

```
1 listdata = []
2 for i in range(3):
3     txt = input('리스트에 추가할 값을 입력하세요[%d/3]: '%(i+1))
4     listdata.append(txt)
5     print(listdata)
6
```

List – Lab (Cont.)

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 pos = solarsys.index('목성')
3 solarsys.insert(pos, '소행성')
4 print(solarsys)
5
```

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 del solarsys[0]
3 print(solarsys)
4 del solarsys[-2]
5 print(solarsys)
6
```

List – Lab (Cont.)

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 solarsys.remove('태양')
3 print(solarsys)
4
```

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 del solarsys[1:3]
3 print(solarsys)
```

List – Lab (Cont.)

```
1 listdata = [2, 2, 1, 3, 8, 5, 7, 6, 3, 6, 2, 3, 9, 4, 4]
2 c1 = listdata.count(2)
3 c2 = listdata.count(7)
4 print(c1)    # 3이 출력됨
5 print(c2)    # 1이 출력됨
```

```
1 listdata = [2, 2, 1, 3, 8, 5, 7, 6, 3, 6, 2, 3, 9, 4, 4]
2 del listdata
3 print(listdata)
4
```

List – Lab (Cont.)

```
1 namelist = ['Mary', 'Sams', 'Aimy', 'Tom', 'Michale', 'Bob', 'Kelly']
2 namelist.sort()
3 print(namelist)
4
```

```
1 namelist = ['Mary', 'Sams', 'Aimy', 'Tom', 'Michale', 'Bob', 'Kelly']
2 ret1 = sorted(namelist)
3 ret2 = sorted(namelist, reverse=True)
4 print(namelist)
5 print(ret1)
6 print(ret2)
7
```

List – Lab (Cont.)

```
1 from random import shuffle
2
3 listdata = list(range(1, 11))
4 for i in range(3):
5     shuffle(listdata)
6     print(listdata)     # 출력 결과는 실행할 때마다 달라짐
7
```

List – Lab (Cont.)

```
1 solarsys = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 ret = list(enumerate(solarsys))
3 print(ret)
4
5 for i, body in enumerate(solarsys):
6     print('태양계의 %d번째 천체: %s' %(i, body))
7
```

```
1 listdata = [2, 2, 1, 3, 8, 5, 7, 6, 3, 6, 2, 3, 9, 4, 4]
2 ret = sum(listdata)
3 print(ret)     # 65가 출력됨
4
```

List – Lab (Cont.)

```
1 listdata1 = [0, 1, 2, 3, 4]
2 listdata2 = [True, True, True]
3 listdata3 = ['', [], (), {}, None, False]
4 print(all(listdata1))   # False가 출력됨
5 print(any(listdata1))  # True가 출력됨
6 print(all(listdata2))   # True가 출력됨
7 print(any(listdata2))  # True가 출력됨
8 print(all(listdata3))   # False가 출력됨
9 print(any(listdata3))  # False가 출력됨
10
```

List – Lab (Cont.)

```
1 solar1 = ['태양', '수성', '금성', '지구', '화성', '목성', '토성', '천왕성', '해왕성']
2 solar2 = ['Sun', 'Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
3 solardict = {}
4 for i, k in enumerate(solar1):
5     val = solar2[i]
6     solardict[k] = val
7
8 print(solardict)
9
```

List – Lab (Cont.)

```
1 list1 = [1, 2, 3, 4, 5]
2 list2 = ['a', 'b', 'c']
3 list3 = [1, 'a', 'abc', [1, 2, 3, 4, 5], ['a', 'b', 'c']]
4 list1[0] = 6
5 print(list1)       # [6, 2, 3, 4, 5]가 출력됨
6 def myfunc():
7     print('안녕하세요')
8 list4 = [1, 2, myfunc]
9 list4[2]()         # '안녕하세요' 가 출력됨
10
```

List – Lab (Cont.)

- [1,3,5,4,2] list를 [5,4,3,2,1]로 변경하기

List – Lab (Cont.)

- ['Life', 'is', 'too', 'short'] list를 Life is too short.문장으로 변경하기

Tuples

- Is a sequence of immutable Python objects.
- Tuples are sequences, just like lists.
- The main difference between the tuples and the lists is that the tuples cannot be changed unlike lists.
- Tuples use parentheses, whereas lists use square brackets.

Tuples (Cont.)

- Creating a tuple is as simple as putting different comma-separated values.
- Optionally, can put these comma-separated values between parentheses also.

```
tup1 = ('physics', 'chemistry', 1997, 2000)
```

Updating Tuples

- Tuples are immutable, which means cannot update or change the values of tuple elements.

```
>>> tup1= (12, 34.56)
>>> tup2= ('abc', 'xyz')
>>>
>>> #Following action is not valid for tuples
>>> #tup1[0] = 100
>>>
>>> #So let's create a new tuple as follows
>>> tup3 = tup1 + tup2
>>> print (tup3)
(12, 34.56, 'abc', 'xyz')
>>>
```

Delete Tuple Elements

- Removing individual tuple elements is not possible.
- There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.
- To explicitly remove an entire tuple, just use the `del` statement.

Delete Tuple Elements (Cont.)

```
>>>
>>> tup = ('physics', 'chemistry', 1997, 2000)
>>>
>>> print (tup)
('physics', 'chemistry', 1997, 2000)
>>> del tup
>>> print ("After deleting tup : ")
After deleting tup :
>>> print (tup)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print (tup)
NameError: name 'tup' is not defined
>>>
```

Basic Tuples Operations

- Tuples respond to the + and * operators much like strings.
- Mean concatenation and repetition here too, except that the result is a new tuple, not a string.
- In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter.

Basic Tuples Operations (Cont.)

Python Expression	Results	Description
`len((1,2,3))`	3	Length
`(1,2,3) + (4,5,6)`	(1,2,3,4,5,6)	Concatenation
`('Hi!',) * 4`	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
`3 in (1,2,3)`	True	Membership
`for x in (1,2,3) : print (x ,end =' ')`	1 2 3	Iteration

Indexing, Slicing and Matrixes

- Since tuples are sequences, indexing and slicing work the same way for tuples as they do for strings.

```
T = ('C++', 'Java' , 'Python')
```

Python Expression	Results	Description
`T[2]`	'Python'	Offsets start at zero
`T[-2]`	'Java'	Negative : count from the right
`T[1:]`	('Java', 'Python')	Slicing fetches sections

Built-in Tuple Functions

- `cmp(tuple1, tuple2)`
  - Compares elements of both tuples.

Built-in Tuple Functions (Cont.)

- `len(tuple)`
  - Gives the total length of the tuple.

Built-in Tuple Functions (Cont.)

- **`max(tuple)`**
  - Returns item from the tuple with max value.

Built-in Tuple Functions (Cont.)

- `min(tuple)`
  - Returns item from the tuple with min value.

Built-in Tuple Functions (Cont.)

- `tuple(seq)`
  - Converts a list into tuple.

Tuples - Lab

- (1,2,3) tuple에 4 값을 추가하여 (1,2,3,4)로 만들기

Dictionary

- Each key is separated from its value by a colon (`:`), the items are separated by commas, and the whole thing is enclosed in curly braces.
- An empty dictionary without any items is written with just two curly braces, like this: `{}`.
- Keys are unique within a dictionary while values may not be.
- The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary

- To access dictionary elements, can use the familiar square brackets along with the key to obtain its value.

```
>>> dict = {'Name' : 'Zara', 'Age' : 7, 'Class' : 'First'}
>>> print ("dict['Name'] : ", dict['Name'])
dict['Name'] :  Zara
>>> print ("dict['Age'] : ", dict['Age'])
dict['Age'] :  7
>>>
```

Accessing Values in Dictionary

- If attempt to access a data item with a key, which is not a part of the dictionary, we get an error as follows :

```
>>> dict = {'Name' : 'Zara', 'Age' : 7, 'Class' : 'First'}
>>> print ("dict['Name'] : ", dict['Name'])
dict['Name'] :  Zara
>>> print ("dict['Age'] : ", dict['Age'])
dict['Age'] :  7
>>>
>>>
>>>
>>> dict = {'Name' : 'Zara', 'Age' : 7, 'Class' : 'First'}
>>> print ("dict['Alice'] : ", dict['Alice'])
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print ("dict['Alice'] : ", dict['Alice'])
KeyError: 'Alice'
>>>
```

Updating Dictionary

- Can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown in a simple example given below :

```
>>> dict = {'Name' : 'Zara', 'Age' : 7, 'Class' : 'First'}
>>> dict['Age'] = 8     # Update existing entry
>>> dict['School'] = 'DPS School'     # Add new entry
>>>
>>> print ("dict['Age'] : ", dict['Age'])
dict['Age'] :  8
>>> print ("dict['School'] : ", dict['School'])
dict['School'] :  DPS School
```

Delete Dictionary Elements

- Can either remove individual dictionary elements or clear the entire contents of a dictionary.
- Can also delete entire dictionary in a single operation.

```
>>> dict = {'Name' : 'Zara', 'Age' : 7, 'Class' : 'First'}
>>>
>>> del dict['Name']    # remove entry with key 'Name'
>>> dict.clear()        # remove all entries in dict
>>> del dict            # delete entire dictionary
>>>
>>> print ("dict['Age'] : ", dict['Age'])
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print ("dict['Age'] : ", dict['Age'])
TypeError: 'type' object is not subscriptable
```

Built-in Dictionary Functions and Methods

- `cmp(dict1, dict2)`
  - No longer available in Python 3.

Built-in Dictionary Functions and Methods (Cont.)

- **`len(dict)`**
  - Gives the total length of the dictionary.
  - This would be equal to the number of items in the dictionary.

Built-in Dictionary Functions and Methods (Cont.)

- **`str(dict)`**
  - Produces a printable string representation of a dictionary.

Built-in Dictionary Functions and Methods (Cont.)

- **`type(variable)`**
  - Returns the type of the passed variable.
  - If passed variable is dictionary, then it would return a dictionary type.

Dictionary Methods

- `dict.clear()`
  - Removes all elements of dictionary `dict`.

Dictionary Methods (Cont.)

- `dict.copy()`
  - Returns a shallow copy of dictionary `dict`.

Dictionary Methods (Cont.)

- `dict.fromkeys()`
  - Create a new dictionary with keys from seq and values set to value.

Dictionary Methods (Cont.)

- `dict.get(key, default = None)`
  - For key key, returns value or default if key not in dictionary.

Dictionary Methods (Cont.)

- `dict.has_key(key)`
  - Removed, use the in operation instead.

Dictionary Methods (Cont.)

- `dict.items()`
  - Returns a list of dict's (key, value) tuple pairs.

Dictionary Methods (Cont.)

- **`dict.keys()`**
  - Returns list of dictionary dict's keys.

Dictionary Methods (Cont.)

- `dict.setdefault(key, default = None)`
  - Similar to get(), but will set dict[key] = default if key is not already in dict.

Dictionary Methods (Cont.)

- `dict.update(dict2)`
  - Adds dictionary dict2's key-values pairs to dict.

Dictionary Methods (Cont.)

- `dict.values()`
  - Returns list of dictionary dict's values.

Dictionary - Lab

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 names['Aimy'] = 10000
4 print(names)
5
```

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 del names['Sams']
4 print(names)
5
```

Dictionary – Lab (Cont.)

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 names.clear()
4 print(names)
5
```

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 ks = names.keys()
4 print(ks)
5
6 for k in ks:
7     print('Key:%s \tValue:%d' %(k, names[k]))
8
```

Dictionary – Lab (Cont.)

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 vals = names.values()
4 print(vals)
5
6 vals_list = list(vals)
7 ret = sum(vals_list)
8 print('출생아수 총계: %d' %ret)
9
```

Dictionary – Lab (Cont.)

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 items = names.items()
4 print(items)
5
6 for item in items:
7     print(item)
8
```

Dictionary – Lab (Cont.)

```
1 names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,
2 'Michale':27115, 'Bob':5887, 'Kelly':7855}
3 k = input('이름을 입력하세요: ')
4 if k in names:
5     print('이름이 <%s>인 출생아수는 <%d>명 입니다.' %(k, names[k]))
6 else:
7     print('자료에 <%s>인 이름이 존재하지 않습니다.' %k)
8
```

Dictionary – Lab (Cont.)

- Dictionary a = {'A' : 90, 'B' : 80, 'C' : 70} 에서 'B'에 해당하는 값을 추출하고 삭제하기 (ex. pop() 함수 이용)