

Pemograman 1 (python)

Asep Muhidin, S.Kom, M.Kom
asep.muhidin@gmail.com



www.facebook.com/asep.muhidin



www.twitter.com/asepmuhidin



www.github.com/asepmuhidin



asepmuhidin.blogspot.co.id



Alamat : Perum BCM Blok A.18 No.18
Cikarang - Selatan, Bekasi



Telp : 081316806705

Pertemuan #4

- List
- Tuple
- Dictionary

LIST

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that 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. For example -

```
list1 = ['physics', 'chemistry', 1997, 2000]  
list2 = [1, 2, 3, 4, 5 ]  
list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
list1 = ['physics', 'chemistry', 1997, 2000];  
list2 = [1, 2, 3, 4, 5, 6, 7 ];  
print ("list1[0]: ", list1[0])  
print ("list2[1:5]: ", list2[1:5])
```

When the above code is executed, it produces the following result –

```
list1[0]: physics  
list2[1:5]: [2, 3, 4, 5]
```

Update & Delete Lists

```
##Update
```

```
list = ['physics', 'chemistry', 1997, 2000];  
print ( "Value available at index 2 : ",print list[2])  
list[2] = 2001;  
print ("New value available at index 2 : ",print list[2])
```

```
##Update
```

```
del list[2];  
print ("After deleting value at index 2 : ",print list)
```

Basic List Operations

Lists respond to the `+` and `*` operators much like strings; they 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.

Python Expression	Results	Description
<code>len([1, 2, 3])</code>	3	Length
<code>[1, 2, 3] + [4, 5, 6]</code>	<code>[1, 2, 3, 4, 5, 6]</code>	Concatenation
<code>['Hi!'] * 4</code>	<code>['Hi!', 'Hi!', 'Hi!', 'Hi!']</code>	Repetition
<code>3 in [1, 2, 3]</code>	True	Membership
<code>for x in [1, 2, 3]: print x,</code>	1 2 3	Iteration

Built-in List Functions and Methods

Sr.No.	Function with Description
1	cmp(list1, list2) ↗ Compares elements of both lists.
2	len(list) ↗ Gives the total length of the list.
3	max(list) ↗ Returns item from the list with max value.
4	min(list) ↗ Returns item from the list with min value.
5	list(seq) ↗ Converts a tuple into list.

Sr.No.	Methods with Description
1	list.append(obj) ↗ Appends object obj to list
2	list.count(obj) ↗ Returns count of how many times obj occurs in list
3	list.extend(seq) ↗ Appends the contents of seq to list
4	list.index(obj) ↗ Returns the lowest index in list that obj appears
5	list.insert(index, obj) ↗ Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) ↗ Removes and returns last object or obj from list
7	list.remove(obj) ↗ Removes object obj from list
8	list.reverse() ↗ Reverses objects of list in place
9	list.sort([func]) ↗ Sorts objects of list, use compare func if given

Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

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

```
tup1 = ('physics', 'chemistry', 1997, 2000);  
tup2 = (1, 2, 3, 4, 5 );  
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing – `tup1 = ()`. To write a tuple containing a single value you have to include a comma, even though there is only one value – `tup1 = (50,)`

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing values in tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);  
tup2 = (1, 2, 3, 4, 5, 6, 7 );  
print ("tup1[0]: ", tup1[0])  
print ("tup2[1:5]: ", tup2[1:5])
```

When the above code is executed, it produces the following result –

```
tup1[0]:  physics  
tup2[1:5]:  [2, 3, 4, 5]
```

Update Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –

```
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;
```

When the above code is executed, it produces the following result –
(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. For example –

```
tup = ('physics', 'chemistry', 1997, 2000);  
print tup;  
del tup;  
print "After deleting tup : ";  
print tup;
```

Basic Tuple Operations

Tuples respond to the + and * operators much like strings; they 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 –

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

No Enclosing Delimiters






Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., default to tuples, as indicated in these short examples –

```
print ('abc', -4.24e93, 18+6.6j, 'xyz')  
x, y = 1, 2;  
print "Value of x , y : ", x,y;
```

When the above code is executed, it produces the following result –

```
abc -4.24e+93 (18+6.6j) xyz  
Value of x , y : 1 2
```

Built-in Tuple Functions

Sr.No.	Function with Description
1	cmp(tuple1, tuple2)  Compares elements of both tuples.
2	len(tuple)  Gives the total length of the tuple.
3	max(tuple)  Returns item from the tuple with max value.
4	min(tuple)  Returns item from the tuple with min value.
5	tuple(seq)  Converts a list into tuple.

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, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example –

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

```
print "dict['Name']: ", dict['Name']
```

```
print "dict['Age']: ", dict['Age']
```

When the above code is executed, it produces the following result –

```
dict['Name']: Zara
```

```
dict['Age']: 7
```


Updating Dictionary

You 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 below in the simple example –

```
#!/usr/bin/python
```

```
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']  
print "dict['School']: ", dict['School']
```

When the above code is executed, it produces the following result –

```
dict['Age']: 8  
dict['School']: DPS School
```

Deleting Dictionary Elements

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

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example –

```
#!/usr/bin/python

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']
print "dict['School']: ", dict['School']
```

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys –

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example –

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}  
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

```
dict['Name']: Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example –

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7}  
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

Traceback (most recent call last):

File "test.py", line 3, in <module>

dict = {'Name': 'Zara', 'Age': 7};

TypeError: list objects are unhashable

Built-in Dictionary functions and methods

Sr.No.	Function with Description
1	cmp(dict1, dict2) ↗ Compares elements of both dict.
2	len(dict) ↗ Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) ↗ Produces a printable string representation of a dictionary
4	type(variable) ↗ Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods –

Sr.No.	Methods with Description
1	dict.clear() ↗ Removes all elements of dictionary <i>dict</i>
2	dict.copy() ↗ Returns a shallow copy of dictionary <i>dict</i>
3	dict.fromkeys() ↗ Create a new dictionary with keys from seq and values set to <i>value</i> .

4	dict.get(key, default=None) ↗ For key <i>key</i> , returns value or default if key not in dictionary
5	dict.has_key(key) ↗ Returns <i>true</i> if key in dictionary <i>dict</i> , <i>false</i> otherwise
6	dict.items() ↗ Returns a list of <i>dict</i> 's (key, value) tuple pairs
7	dict.keys() ↗ Returns list of dictionary <i>dict</i> 's keys
8	dict.setdefault(key, default=None) ↗ Similar to <i>get()</i> , but will set <i>dict[key]=default</i> if <i>key</i> is not already in <i>dict</i>
9	dict.update(dict2) ↗ Adds dictionary <i>dict2</i> 's key-values pairs to <i>dict</i>
10	dict.values() ↗ Returns list of dictionary <i>dict</i> 's values