

A Short and Incomplete Introduction to Python

Part 7: dicts and other data structures

Riccardo Murri <riccardo.murri@uzh.ch>,

Sergio Maffioletti <sergio.maffioletti@uzh.ch>

S3IT: Services and Support for Science IT,

University of Zurich

Dictionaries

Dictionaries

The `dict` type implements a key/value mapping:

```
>>> D = {}      # empty dict
>>> D['a'] = 1
>>> D[2] = 'b'
>>> D
{'a': 1, 2: 'b'}
```

Dictionaries can be created and initialized using the following syntax:

```
>>> D = { 'a':1, 2:'b' }
>>> D['a']
1
```

The `for` statement can be used to loop over keys of a dictionary:

```
>>> D = { 'a':1, 'b':2 }
>>> for val in D.keys():
...     print(val)
'a'
'b'
```

Loop over
dictionary *keys*.
*The `.keys()` part can
be omitted, as it's the
default!*

If you want to loop over dictionary *values*, you have to explicitly request it.

```
>>> D = { 'a':1, 'b':2 }  
>>> for val in D.values() :  
...     print(val)  
1  
2
```

Loop over
dictionary *values*
The .values ()
cannot be omitted!

Finally, you can request keys *and* corresponding values with the `.items()` iterator:

```
>>> D = { 'a':1, 'b':2 }                                Loop over
>>> for key, val in D.items():                         dictionary items.
...
print(key, '=>', val)                                 Note the syntax for
a=>1                                                 multiple-assignment!
b=>2
```

The 'in' operator (2)

Use the `in` operator to test for presence of an item in a collection.

`x in D`

`x in D.keys()`

Evaluates to `True` if `x` is equal to a *key* in the `D` dictionary.

`x in D.values()`

Evaluates to `True` if `x` is equal to a *value* in the `D` dictionary.

Exercise 7.A: Write a function `wordcount(filename)` that reads a text file and returns a dictionary, mapping words into occurrences (disregarding case) of that word in the text.

For example, using the `lipsum.txt` file:

```
>>> wordcount('lipsum.txt')
{'and': 3, 'model': 1, 'more-or-less': 1,
 'letters': 1, [...]}
```

For the purposes of this exercise, a “word” is defined as a sequence of letters and the character “-”, i.e., “e-mail” and “more-or-less” should both be counted as a single word.

Bar plots

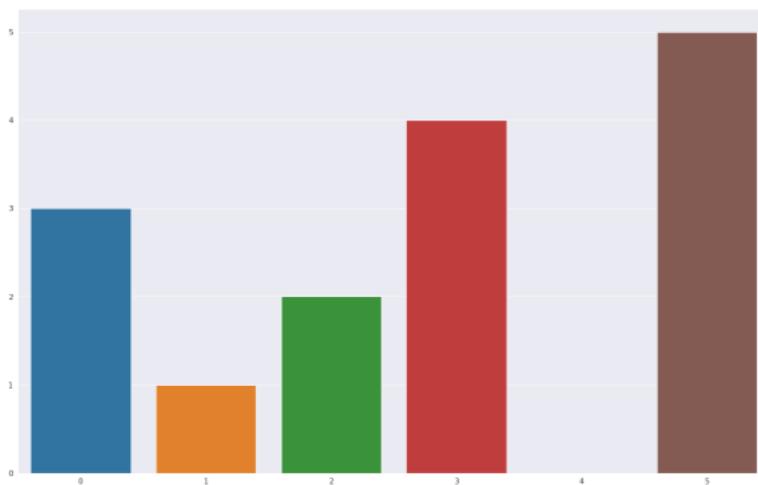
Bar plots, I

Seaborn's `barplot()` function can be used to draw bar plots.

In [1]: `x = [0, 1, 2, 3, 4, 5]`

In [2]: `y = [3, 1, 2, 4, 0, 5]`

In [3]: `seaborn.barplot(x, y)`



Bar plots, II

An additional argument `orient=...` can be used to give:

- ▶ vertical orientation (`orient='v'`, default), or
- ▶ horizontal (`orient='h'`).

In [4]: `x = [0, 1, 2, 3, 4, 5]`

In [5]: `y = [3, 1, 2, 4, 0, 5]`

In [6]: `seaborn.barplot(y, x, orient='h')`



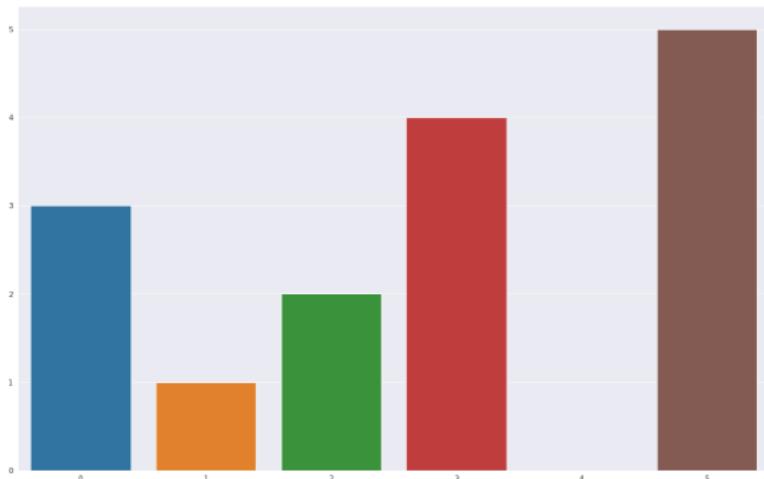
Bar plots, II

Note that the role of x and y parameters is *reversed* in horizontal plots: x is the size of bars, y is the labels.

```
In [7]: x = [0, 1, 2, 3, 4, 5]
```

```
In [8]: y = [3, 1, 2, 4, 0, 5]
```

```
In [9]: seaborn.barplot(y, x, orient='h')
```



Bar plots, III

Arguments `x` and `y` can be Python lists or NumPy arrays, as usual.

Another variant (available in all Seaborn plotting functions) is to pass a Pandas `DataFrame` object with the `data=...` argument, and then `x` and `y` become names of columns in the data frame.

```
seaborn.barplot(data=csv, x="word", y="frequency")
```

References and copies

Variables are names for values

Python variables are just “names” given to values.

This allows you to *reference* the string ‘Python’ by the *name* a. But also by another name b:



The *same* object can be given many names!

See also: <http://excess.org/article/2014/04/bar-foo/>

The `is` operator

The `is` operator allows you to test whether two names refer to the same object:

<code>>>> a = 1</code>	<code>>>> a = ''</code>	<code>>>> a = list()</code>
<code>>>> b = 1</code>	<code>>>> b = ""</code>	<code>>>> b = list()</code>
<code>>>> a == b</code>	<code>>>> a == b</code>	<code>>>> a == b</code>
<code>True</code>	<code>True</code>	<code>True</code>
<code>>>> a is b</code>	<code>>>> a is b</code>	<code>>>> a is b</code>
<code>True</code>	<code>True</code>	<code>False</code>

All variables are references

In Python, **all objects are ever passed by reference**.

In particular, **variables always store a reference to an object**, never a copy!

Hence, you have to be careful when modifying objects:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b.remove(2)
>>> print(a)
???
```

Q: How many items are in the a list now?

All variables are references

In Python, **all objects are ever passed by reference**.

In particular, **variables always store a reference to an object**, never a copy!

Hence, you have to be careful when modifying objects:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b.remove(2)
>>> print(a)
[1, 3]
```

Run this example in the [Online Python Tutor](#) to better understand what's going on.

This applies particularly for variables that capture the arguments to a function call!

All variables are references (demo)

www.pythontutor.com

```
→ 1 a = [1, 2, 3]
  2 b = a
  3 b.remove(2)
  4 print a
  5 print b
```

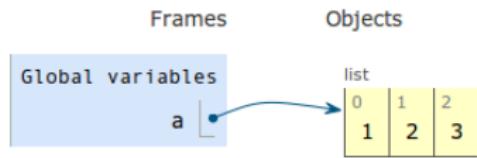
Frames

Objects

All variables are references (demo)

www.pythontutor.com

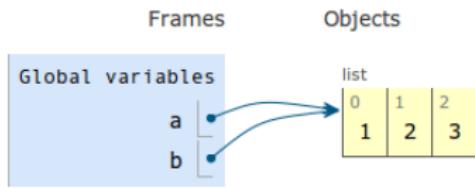
```
→ 1 a = [1, 2, 3]
→ 2 b = a
  3 b.remove(2)
  4 print a
  5 print b
```



All variables are references (demo)

www.pythontutor.com

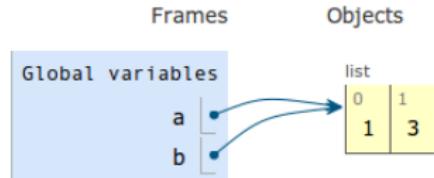
```
1 a = [1, 2, 3]
→ 2 b = a
→ 3 b.remove(2)
4 print a
5 print b
```



All variables are references (demo)

www.pythontutor.com

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



How to copy an object? (0)

```
>>> a = [1, 2]
>>> b = a
>>> b.remove(1)
>>> print(b)
[2]
>>> print(a)
[2]
```

How to copy an object? (1)

```
>>> from copy import copy
>>> a = [1, 2]
>>> b = copy(a)
>>> b.remove(1)
>>> print(b)
[2]
>>> print(a)
[1, 2]
```

How to copy an object? (2)

Note that `copy.copy` makes a *shallow* copy:

```
>>> D = { 'a':[1,2], 'b':3 }
>>> print(D['a'])
[1, 2]
>>> E = copy(D)
>>> print(E)
{ 'a':[1, 2], 'b':3 }
>>> E['a'].remove(1)
>>> print(D['a'])
[2]
```

How to copy an object? (3)

To make a copy of nested data structures,
you need `copy.deepcopy`:

```
>>> from copy import deepcopy
>>> D = { 'a':[1,2], 'b':3 }
>>> print(D['a'])
[1, 2]
>>> E = deepcopy(D)
>>> print(E)
{ 'a':[1, 2], 'b':3 }
>>> E['a'].remove(1)
>>> print(D['a'])
[1, 2]
>>> print(E['a'])
[2]
```

Appendix

Mutable vs Immutable

Some objects (e.g., tuple, int, str) are *immutable* and cannot be modified.

```
>>> S = 'UZH'  
>>> S[2] = 'G'  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'str' object does not support item assignment
```

list, dict, set and user-defined objects are *mutable* and can be modified in-place.

Dictionary, sets and mutable objects

Not all objects can be used as dictionary *keys* or items in a set:

- ▶ *Immutable* objects **can be** used as dict keys or set items.
- ▶ *Mutable* objects **cannot be** used as dict keys or set items.

(Explanation for the technically savvy: a dictionary is essentially a **Hash Table**, therefore keys of a dictionary must be *hashable* objects. If objects were allowed to mutate, their hash value would change too and we would lose the mapping.)

Other containers

Other containers

The following builtin containers are always available:

`dict` *mutable* key/value mapping.

`set` *mutable*, unordered set of *unique* elements.

`frozenset` *immutable*, unordered set of *unique* elements.

Other specialized containers are available in the `collections` module:

`deque` a generalization of stacks and queues

`namedtuple` similar to a tuple, but allows you to access the elements *by name*

`OrderedDict` dictionary that remembers the order that the items were inserted.

Other containers

The following builtin containers are always available:

`dict` *mutable* key/value mapping.

`set` *mutable*, unordered set of *unique* elements.

`frozenset` *immutable*, unordered set of *unique* elements.

Other specialized containers are available in the `collections` module:

`deque` a generalization of stacks and queues

`namedtuple` similar to a tuple, but allows you to access the elements *by name*

`OrderedDict` dictionary that remembers the order that the items were inserted.