Python for scientists

Lesson 2

Data structures: lists and dictionaries

```
def complementary(seq):
    nt_comp = {
    'A' : 'T',
    'C' : 'G',
    'G' : 'C',
    'T' : 'A',
    }
    for nt in seq:
        compseq += nt_comp[nt]
    return compseq
```

Alvaro Sebastián Yagüe

www.sixthresearcher.com

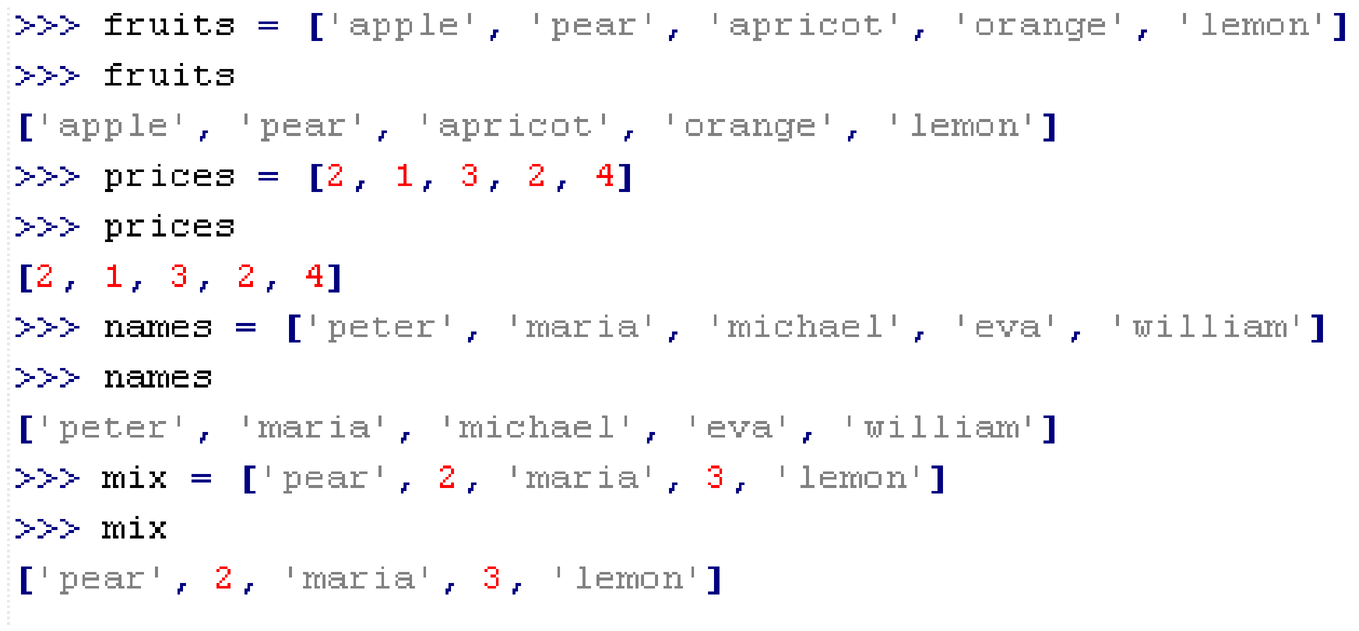
@SixthResearcher

Lists

A new data type: lists

A list is a comma-separated set of values between square brackets.

These values are usually numbers or strings. Lists might contain items of different types. Is the equivalent Python type for a Perl array.

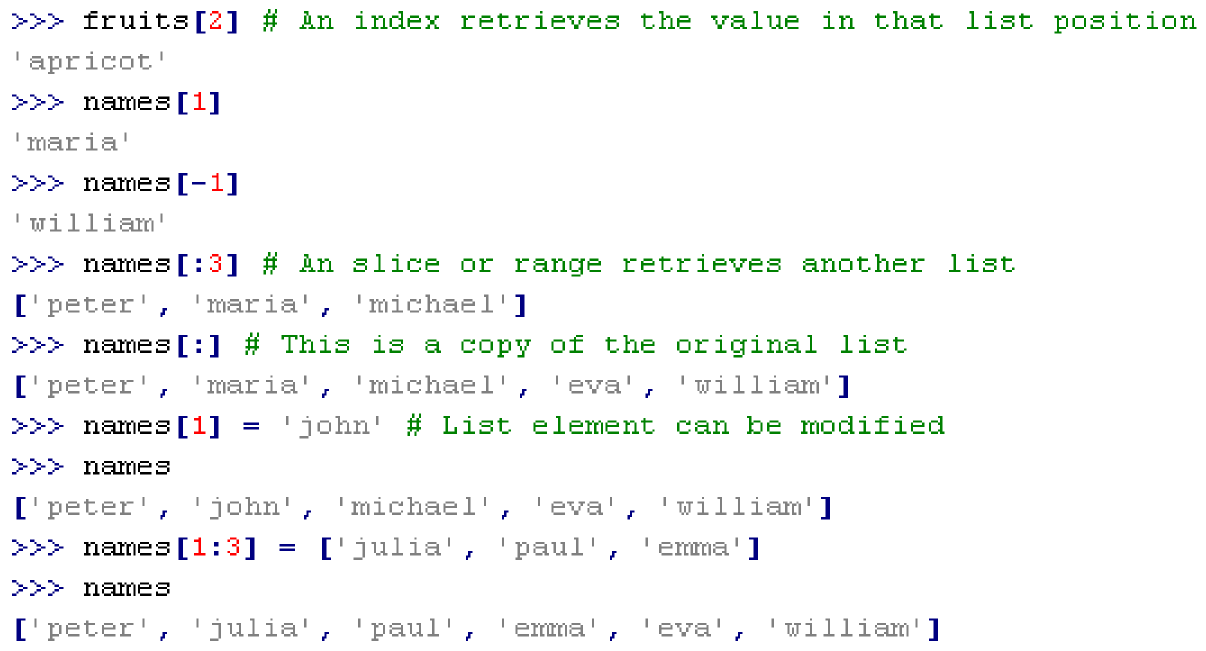


```
>>> fruits = ['apple', 'pear', 'apricot', 'orange', 'lemon']
>>> fruits
['apple', 'pear', 'apricot', 'orange', 'lemon']
>>> prices = [2, 1, 3, 2, 4]
>>> prices
[2, 1, 3, 2, 4]
>>> names = ['peter', 'maria', 'michael', 'eva', 'william']
>>> names
['peter', 'maria', 'michael', 'eva', 'william']
>>> mix = ['pear', 2, 'maria', 3, 'lemon']
>>> mix
['pear', 2, 'maria', 3, 'lemon']
```

A new data type: lists

We can refer to a item from the list by its position, also called **index**.

Lists can be indexed and sliced like we did with strings:



```
>>> fruits[2] # An index retrieves the value in that list position
'apricot'
>>> names[1]
'maria'
>>> names[-1]
'william'
>>> names[:3] # An slice or range retrieves another list
['peter', 'maria', 'michael']
>>> names[:] # This is a copy of the original list
['peter', 'maria', 'michael', 'eva', 'william']
>>> names[1] = 'john' # List element can be modified
>>> names
['peter', 'john', 'michael', 'eva', 'william']
>>> names[1:3] = ['julia', 'paul', 'emma']
>>> names
['peter', 'julia', 'paul', 'emma', 'eva', 'william']
```

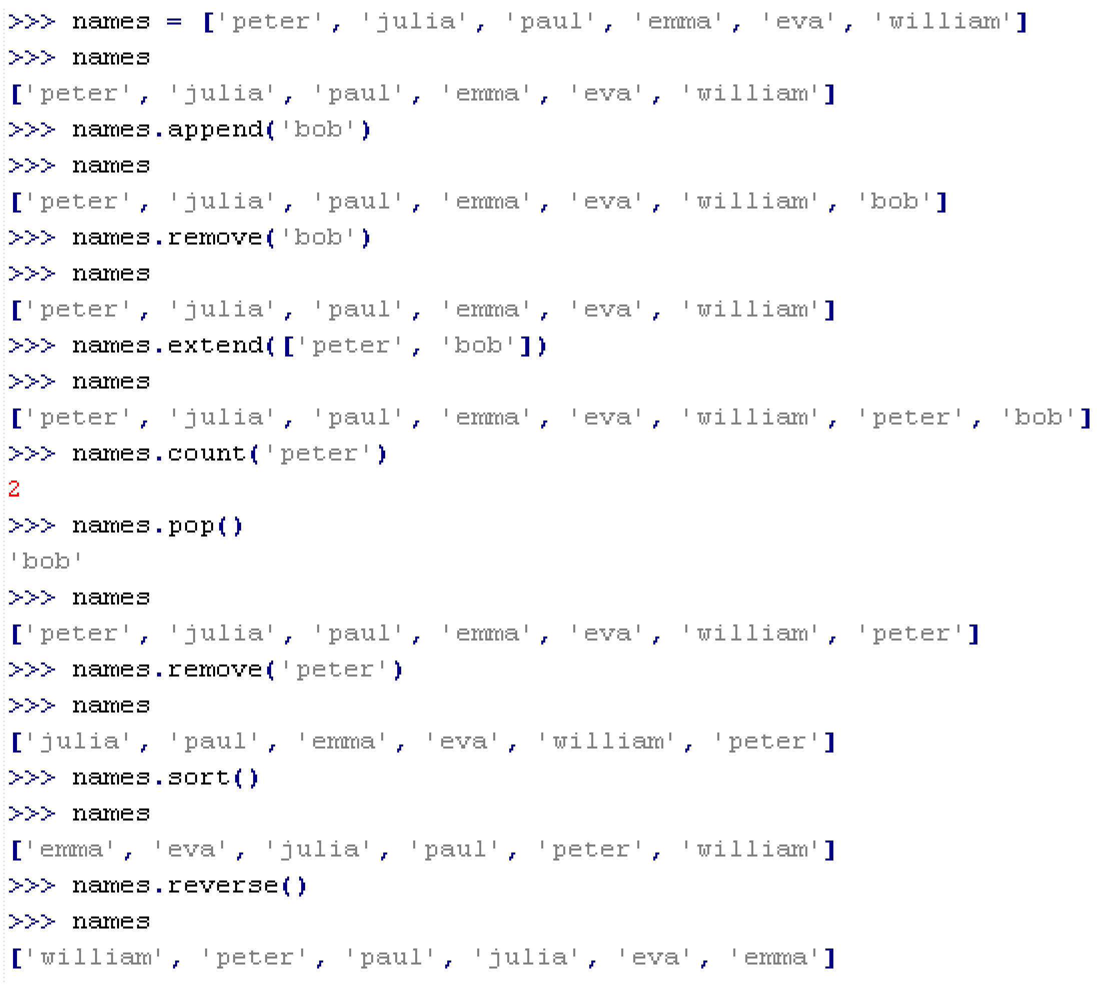
In contrast to strings, lists can be modified

List methods

list.append(*x*)	Adds an item to the end of the list.
list.extend(L)	Extends the list by appending all the items in the given list.
list.insert(i, x)	Inserts an item at a given position.
list.remove(x)	Removes the first item from the list whose value is x.
list.pop(i)	Removes the item at the given position in the list, and return it.
list.clear()	Remove all items from the list. Equivalent to del a[:].
list.index(x)	Returns the index in the list of the first item whose value is x.
list.count(x)	Returns the number of times x appears in the list.
list.sort()	Sorts the items of the list in place.
list.reverse()	Reverses the elements of the list in place.
list.copy()	Returns a shallow copy of the list. Equivalent to a[:].

x is a data value, **L** is a list and **i** is a list position or index

List methods

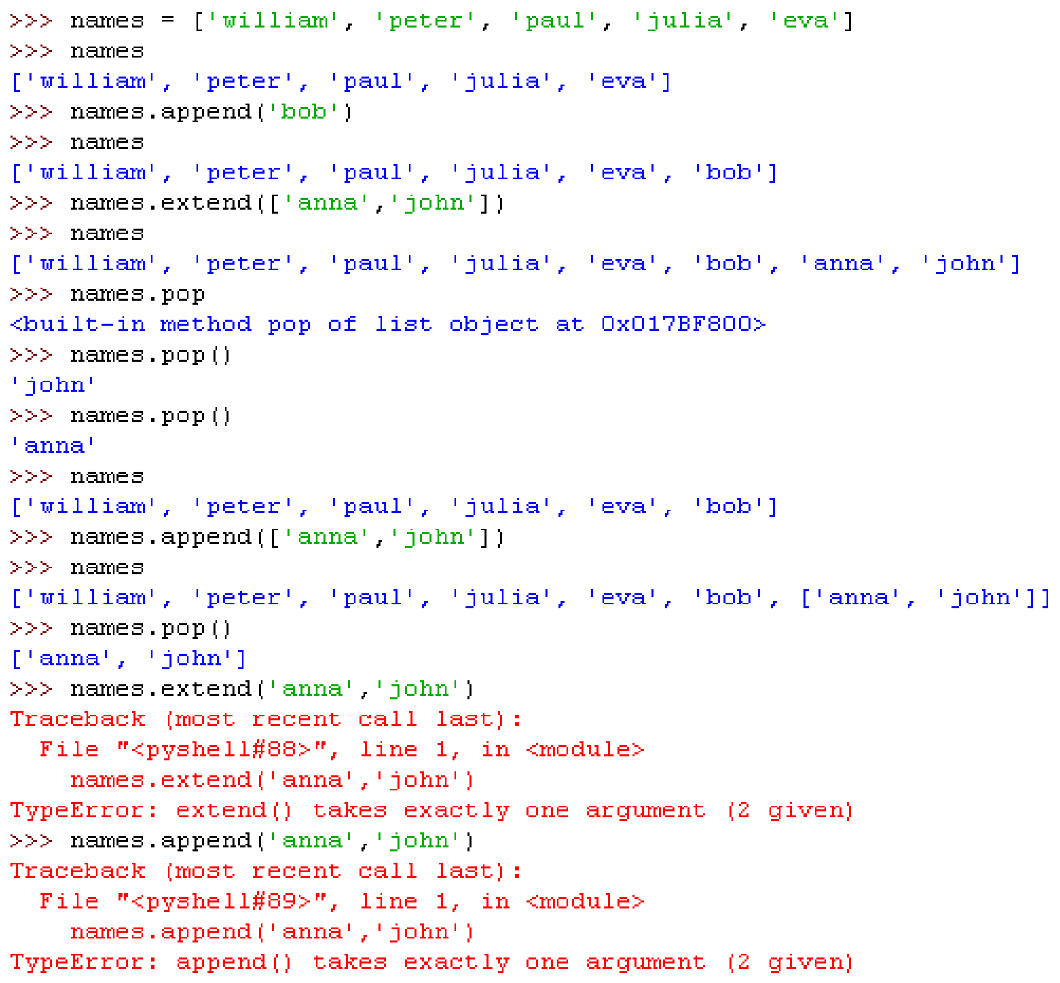


```
>>> names = ['peter', 'julia', 'paul', 'emma', 'eva', 'william']
>>> names
['peter', 'julia', 'paul', 'emma', 'eva', 'william']
>>> names.append('bob')
>>> names
['peter', 'julia', 'paul', 'emma', 'eva', 'william', 'bob']
>>> names.remove('bob')
>>> names
['peter', 'julia', 'paul', 'emma', 'eva', 'william']
>>> names.extend(['peter', 'bob'])
>>> names
['peter', 'julia', 'paul', 'emma', 'eva', 'william', 'peter', 'bob']
>>> names.count('peter')
2
>>> names.pop()
'bob'
>>> names
['peter', 'julia', 'paul', 'emma', 'eva', 'william', 'peter']
>>> names.remove('peter')
>>> names
['julia', 'paul', 'emma', 'eva', 'william', 'peter']
>>> names.sort()
>>> names
['emma', 'eva', 'julia', 'paul', 'peter', 'william']
>>> names.reverse()
>>> names
['william', 'peter', 'paul', 'julia', 'eva', 'emma']
```

Alvaro Sebastián Yagüe www.sixthresearcher.com @SixthResearcher

List methods

And be careful, each list method requires the proper input parameters.

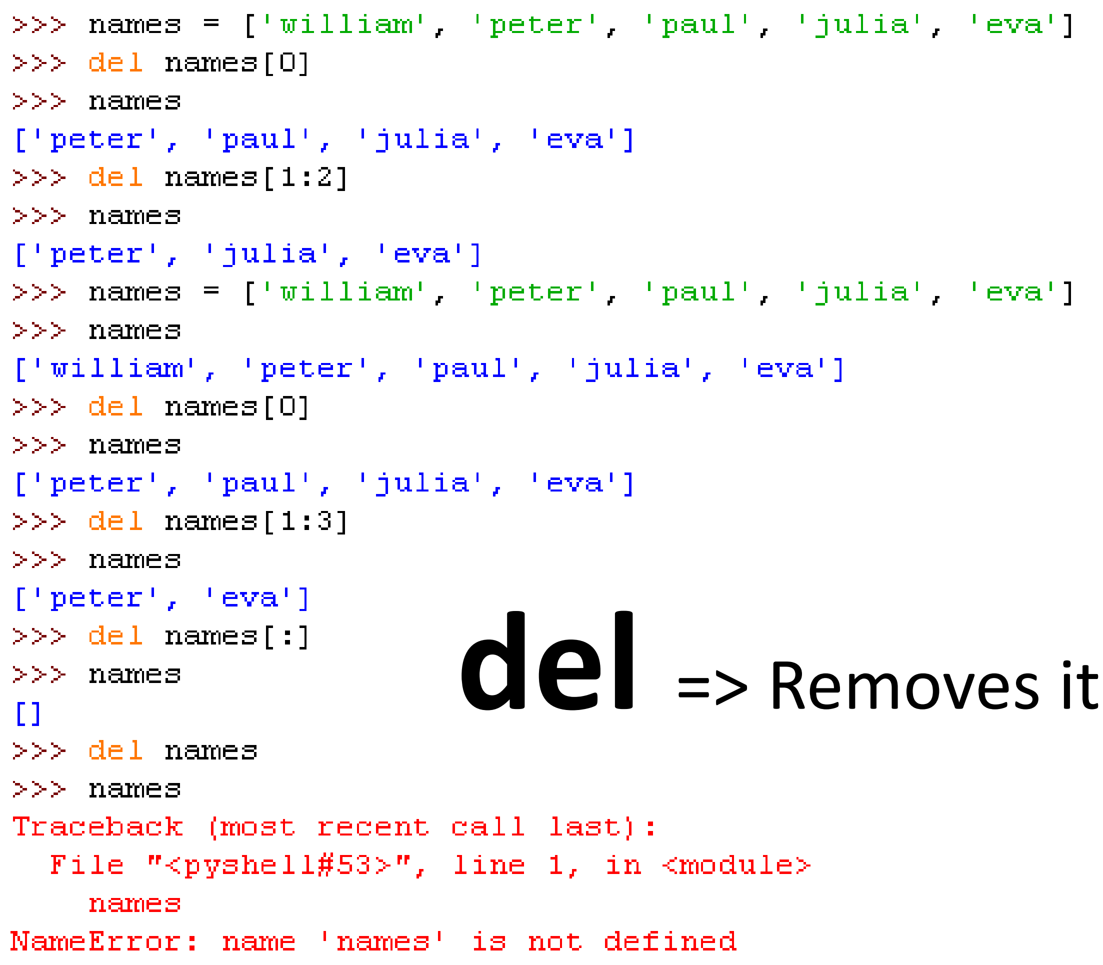


```
>>> names = ['william', 'peter', 'paul', 'julia', 'eva']
>>> names
['william', 'peter', 'paul', 'julia', 'eva']
>>> names.append('bob')
>>> names
['william', 'peter', 'paul', 'julia', 'eva', 'bob']
>>> names.extend(['anna','john'])
>>> names
['william', 'peter', 'paul', 'julia', 'eva', 'bob', 'anna', 'john']
>>> names.pop
<built-in method pop of list object at 0x017BF800>
>>> names.pop()
'john'
>>> names.pop()
'anna'
>>> names
['william', 'peter', 'paul', 'julia', 'eva', 'bob']
>>> names.append(['anna','john'])
>>> names
['william', 'peter', 'paul', 'julia', 'eva', 'bob', ['anna', 'john']]
>>> names.pop()
['anna', 'john']
>>> names.extend('anna','john')
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    names.extend('anna','john')
TypeError: extend() takes exactly one argument (2 given)
>>> names.append('anna','john')
Traceback (most recent call last):
  File "<pyshell#89>", line 1, in <module>
    names.append('anna','john')
TypeError: append() takes exactly one argument (2 given)
```

List methods

```
>>> names = ['william', 'peter', 'paul', 'julia', 'eva']
>>> names
['william', 'peter', 'paul', 'julia', 'eva']
>>> names = ['william', 'peter', 'paul', 'julia', 'eva']
>>> last_name = names[-1] # Only retrieves (not removes) the last element
>>> first_name = names[0] # Only retrieves (not removes) the first element
>>> names
['william', 'peter', 'paul', 'julia', 'eva']
>>> last_name
'eva'
>>> first_name
'william'
>>> last_name = names.pop() # Retrieves and removes the last element
>>> first_name = names.pop(0) # Retrieves and removes the first element
>>> last_name
'eva'
>>> first_name
'william'
>>> names
['peter', 'paul', 'julia']
>>> names.append(last_name) # Inserts an element at the end of the list
['peter', 'paul', 'julia', 'eva']
>>> names.insert(0,first_name) # Inserts an element at the beginning of the list
>>> names
['william', 'peter', 'paul', 'julia', 'eva']
>>> names[0] = 'john' # Replaces the first element of the list
>>> names[-1] = 'maria' # Replaces the last element of the list
>>> names
['john', 'peter', 'paul', 'julia', 'maria']
```

List methods



```
>>> names = ['william', 'peter', 'paul', 'julia', 'eva']
>>> del names[0]
>>> names
['peter', 'paul', 'julia', 'eva']
>>> del names[1:2]
>>> names
['peter', 'julia', 'eva']
>>> names = ['william', 'peter', 'paul', 'julia', 'eva']
>>> names
['william', 'peter', 'paul', 'julia', 'eva']
>>> del names[0]
>>> names
['peter', 'paul', 'julia', 'eva']
>>> del names[1:3]
>>> names
['peter', 'eva']
>>> del names[:]
>>> names
[]
>>> del names
>>> names
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    names
NameError: name 'names' is not defined
```

del => Removes items from a list

Exercise: Shopping list

Exercise: Shopping list

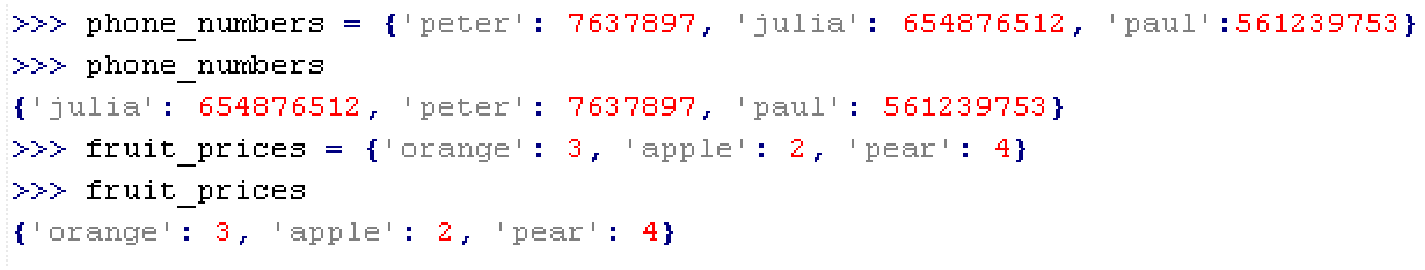
Imagine that we are shopping at supermarket…

- Let’s create a list called ‘basket’ with all the food that we are taking, we should start buying some juice.
- Do not take all at the same time, walk a little and include several articles from different sections.
- We regret of having taken the juice, we prefer beer, so we exchange the juice.
- Then we discover that it is the end of month, we have little money and we took too many things… we should remove something from the ‘basket’.
- When we go to the cashier counter we want to go to the fast one, but before we should count how many items we bought.
- Finally we arrive at home and we want to ‘sort’ all the food by names to easily localize it when we will be hungry.

Dictionaries

Another data type: dictionaries

A dictionary is a kind of list, but instead of refer to a value by an index, we will refer by a key. A key is some data, usually a string of text, that helps us to easy locate some content from the dictionary. A dictionary is referred in other languages as “associative memory”, “associative array” or “hash” (Perl).



```
>>> phone_numbers = {'peter': 7637897, 'julia': 654876512, 'paul':561239753}
>>> phone_numbers
{'julia': 654876512, 'peter': 7637897, 'paul': 561239753}
>>> fruit_prices = {'orange': 3, 'apple': 2, 'pear': 4}
>>> fruit_prices
{'orange': 3, 'apple': 2, 'pear': 4}
```

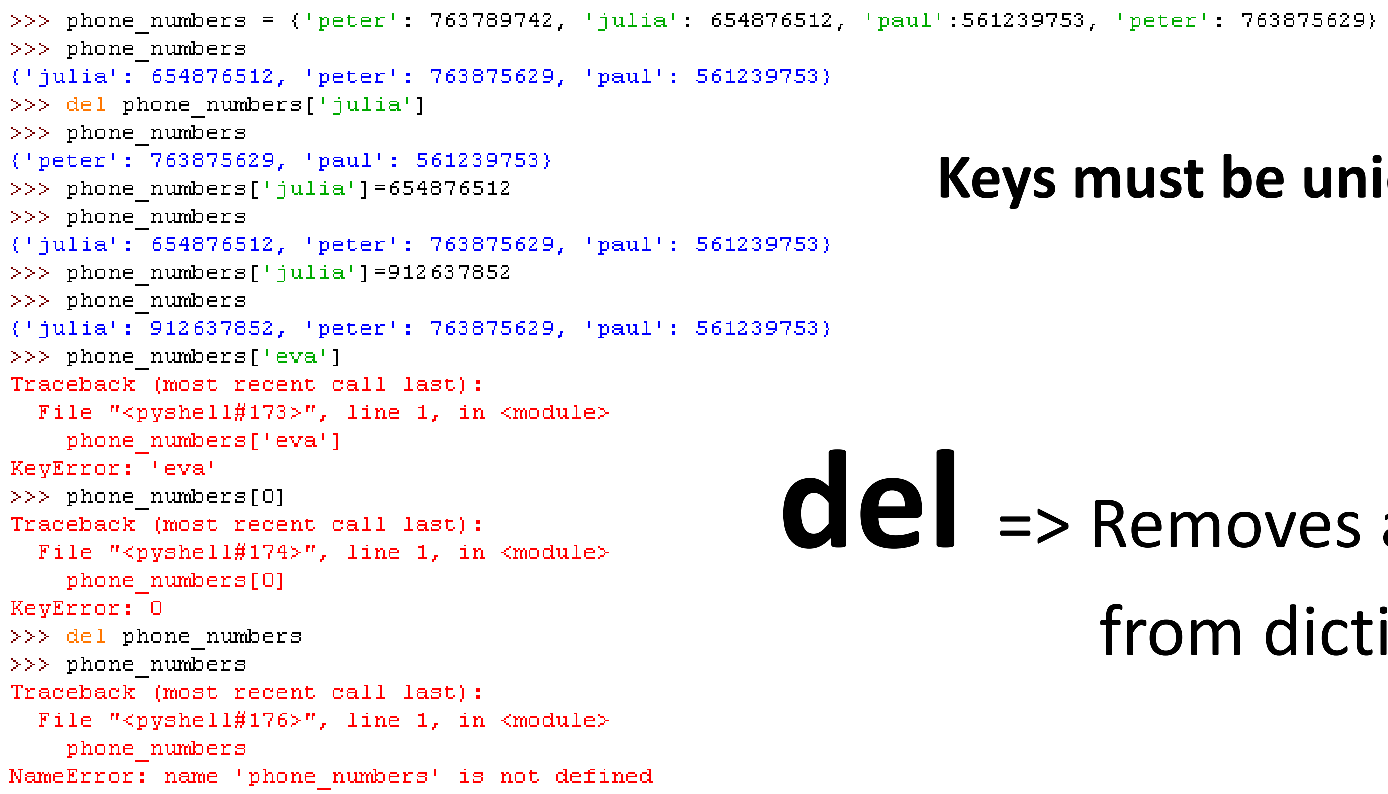
A dictionary is defined by a comma-separated list of key:value pairs within braces:

{ key1:value1, key2:value2, ke3:value3 }

Another data type: dictionaries

Dictionary values are not ordered, we can refer to them only by their keys.

Dictionaries CANNOT be indexed and sliced like we did with lists and strings.



```
>>> phone_numbers = {'peter': 763789742, 'julia': 654876512, 'paul':561239753, 'peter': 763875629}
>>> phone_numbers
{'julia': 654876512, 'peter': 763875629, 'paul': 561239753}
>>> del phone_numbers['julia']
>>> phone_numbers
{'peter': 763875629, 'paul': 561239753}
>>> phone_numbers['julia']=654876512
>>> phone_numbers
{'julia': 654876512, 'peter': 763875629, 'paul': 561239753}
>>> phone_numbers['julia']=912637852
>>> phone_numbers
{'julia': 912637852, 'peter': 763875629, 'paul': 561239753}
>>> phone_numbers['eva']
Traceback (most recent call last):
  File "<pyshell#173>", line 1, in <module>
    phone_numbers['eva']
KeyError: 'eva'
>>> phone_numbers[0]
Traceback (most recent call last):
  File "<pyshell#174>", line 1, in <module>
    phone_numbers[0]
KeyError: 0
>>> del phone_numbers
>>> phone_numbers
Traceback (most recent call last):
  File "<pyshell#176>", line 1, in <module>
    phone_numbers
NameError: name 'phone_numbers' is not defined
```

Keys must be unique.

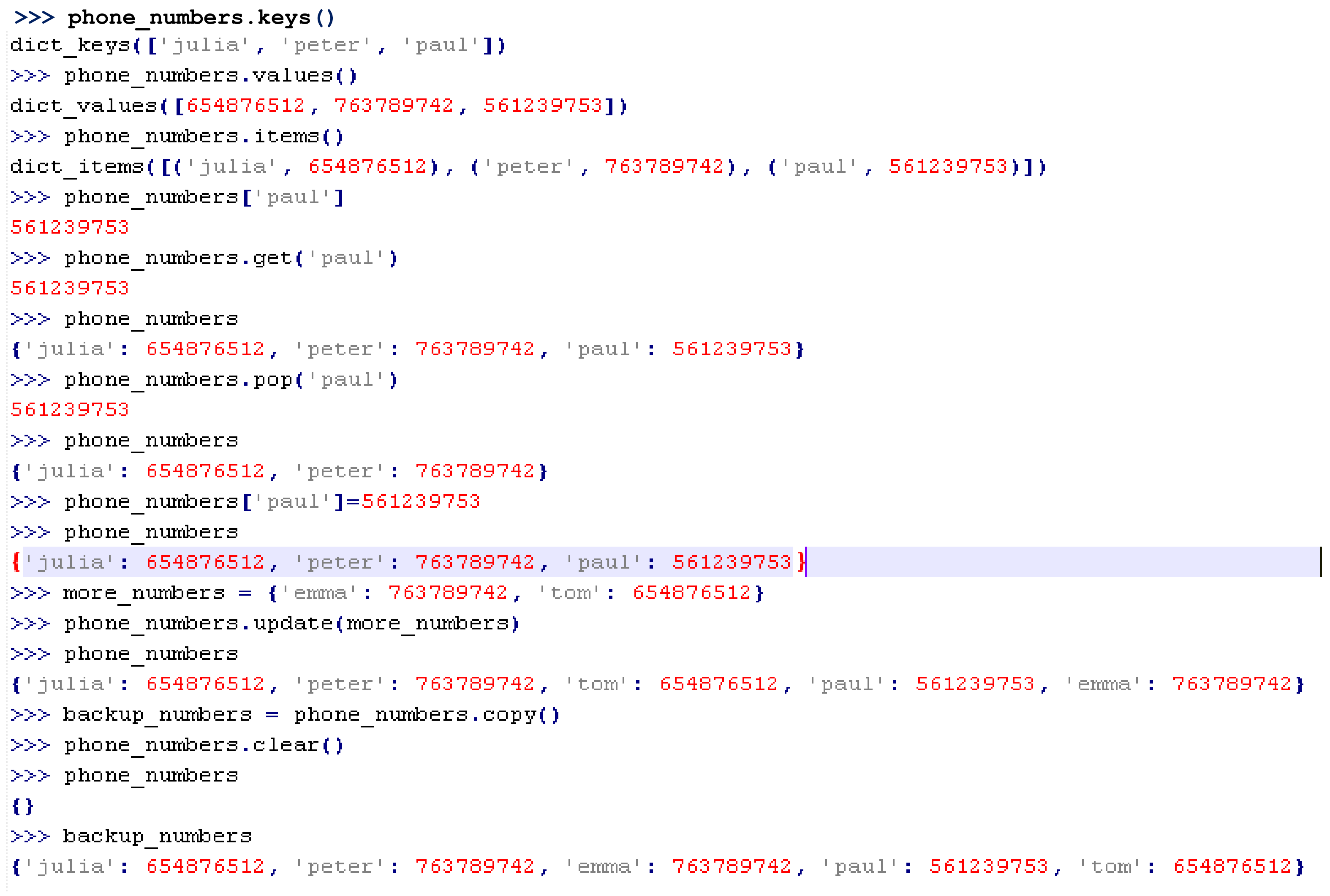
del => Removes an item from dictionary

Dictionary methods

dict.keys()	Returns a list of the keys.
dict.values()	Returns a list of the values.
dict.items()	Returns a list of pairs (key, value).
dict.get(k)	Returns the item associated to a key. Equivalent to dict[k].
dict.pop(k)	Removes the item associated to the key, and return its value.
dict.update(D)	Adds dictionary ‘d’ key-values pairs to dict.
dict.clear()	Removes all items from the dictionary.
dict.copy()	Returns a shallow copy of the dictionary.

k is a key, **v** is a value and **D** is a dictionary.

Dictionary methods



```
>>> phone_numbers.keys()
dict_keys(['julia', 'peter', 'paul'])
>>> phone_numbers.values()
dict_values([654876512, 763789742, 561239753])
>>> phone_numbers.items()
dict_items([('julia', 654876512), ('peter', 763789742), ('paul', 561239753)])
>>> phone_numbers['paul']
561239753
>>> phone_numbers.get('paul')
561239753
>>> phone_numbers
{'julia': 654876512, 'peter': 763789742, 'paul': 561239753}
>>> phone_numbers.pop('paul')
561239753
>>> phone_numbers
{'julia': 654876512, 'peter': 763789742}
>>> phone_numbers['paul']=561239753
>>> phone_numbers
{'julia': 654876512, 'peter': 763789742, 'paul': 561239753}
>>> more_numbers = {'emma': 763789742, 'tom': 654876512}
>>> phone_numbers.update(more_numbers)
>>> phone_numbers
{'julia': 654876512, 'peter': 763789742, 'tom': 654876512, 'paul': 561239753, 'emma': 763789742}
>>> backup_numbers = phone_numbers.copy()
>>> phone_numbers.clear()
>>> phone_numbers
{}
>>> backup_numbers
{'julia': 654876512, 'peter': 763789742, 'emma': 763789742, 'paul': 561239753, 'tom': 654876512}
```

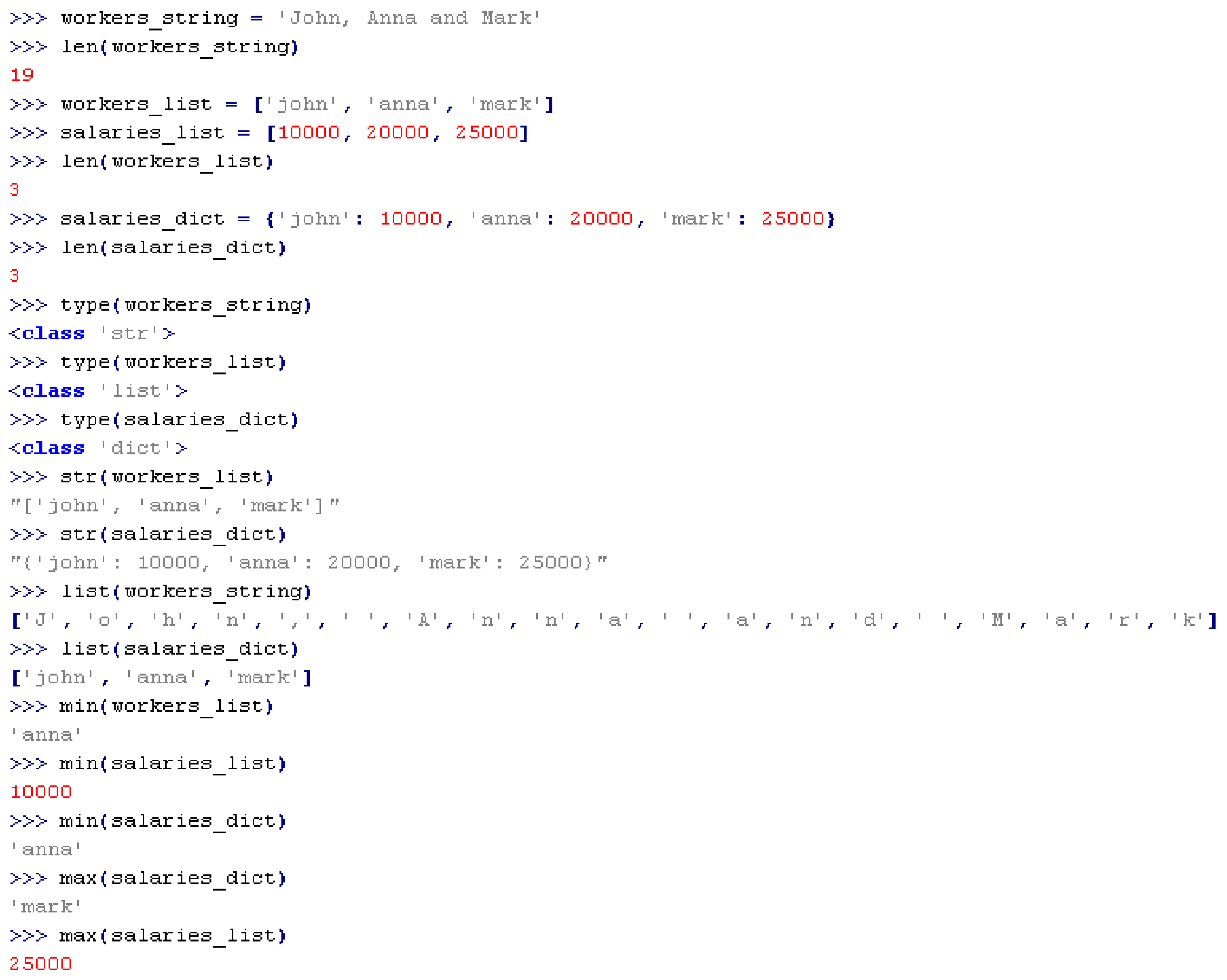
Common functions for strings, lists and dictionaries

Common functions for strings, lists and dictionaries

len(A)	Returns the length or number of items.
type(A)	Returns the data type stored in A.
str(A)	Produces a printable string representation of a list or dictionary.
list(A)	Produces a list from a string or dict keys.
min(A)	Returns the minimum value from A (if is a dict, from keys).
max(A)	Returns the maximum value from A (if is a dict, from keys).

A and **B** can be either a string, list or dictionary.

Common functions for strings, lists and dictionaries

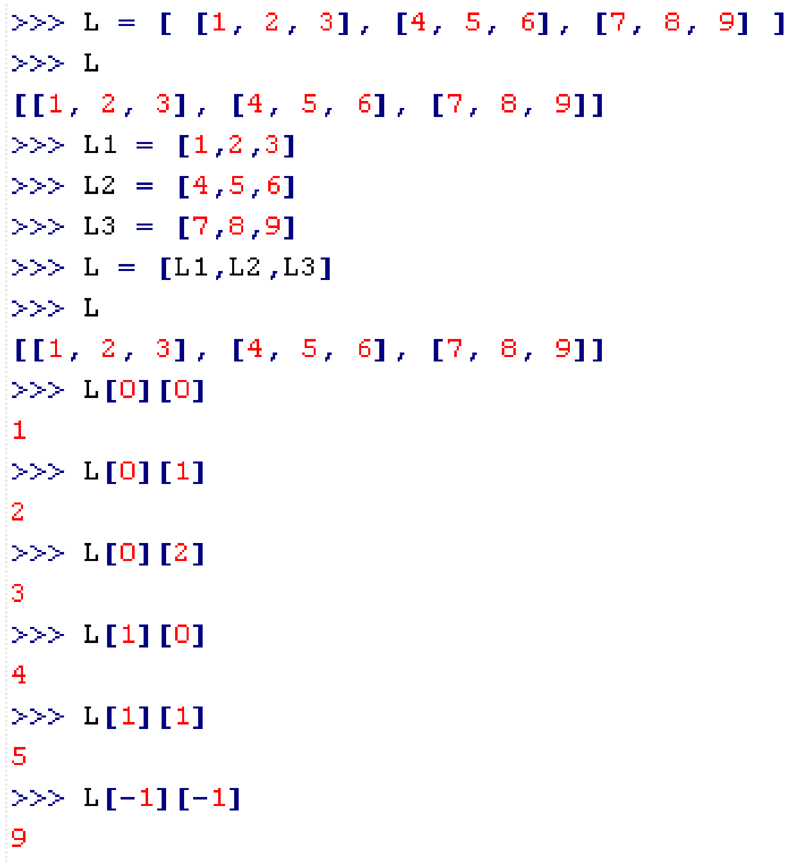


```
>>> workers_string = 'John, Anna and Mark'
>>> len(workers_string)
19
>>> workers_list = ['john', 'anna', 'mark']
>>> salaries_list = [10000, 20000, 25000]
>>> len(workers_list)
3
>>> salaries_dict = {'john': 10000, 'anna': 20000, 'mark': 25000}
>>> len(salaries_dict)
3
>>> type(workers_string)
<class 'str'>
>>> type(workers_list)
<class 'list'>
>>> type(salaries_dict)
<class 'dict'>
>>> str(workers_list)
"['john', 'anna', 'mark']"
>>> str(salaries_dict)
"{'john': 10000, 'anna': 20000, 'mark': 25000}"
>>> list(workers_string)
['J', 'o', 'h', 'n', ',', ' ', 'A', 'n', 'n', 'a', ' ', 'a', 'n', 'd', ' ', 'M', 'a', 'r', 'k']
>>> list(salaries_dict)
['john', 'anna', 'mark']
>>> min(workers_list)
'anna'
>>> min(salaries_list)
10000
>>> min(salaries_dict)
'anna'
>>> max(salaries_dict)
'mark'
>>> max(salaries_list)
25000
```

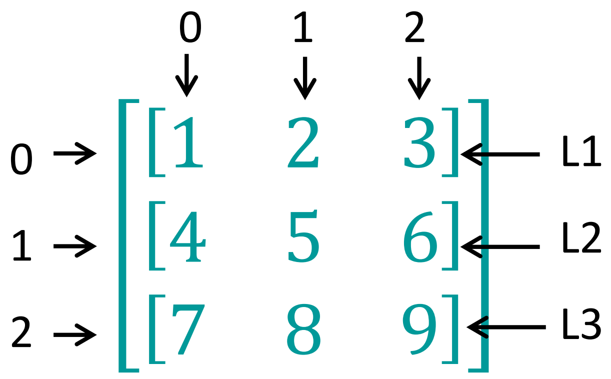
Nested lists and dictionaries

More complexity: nested lists

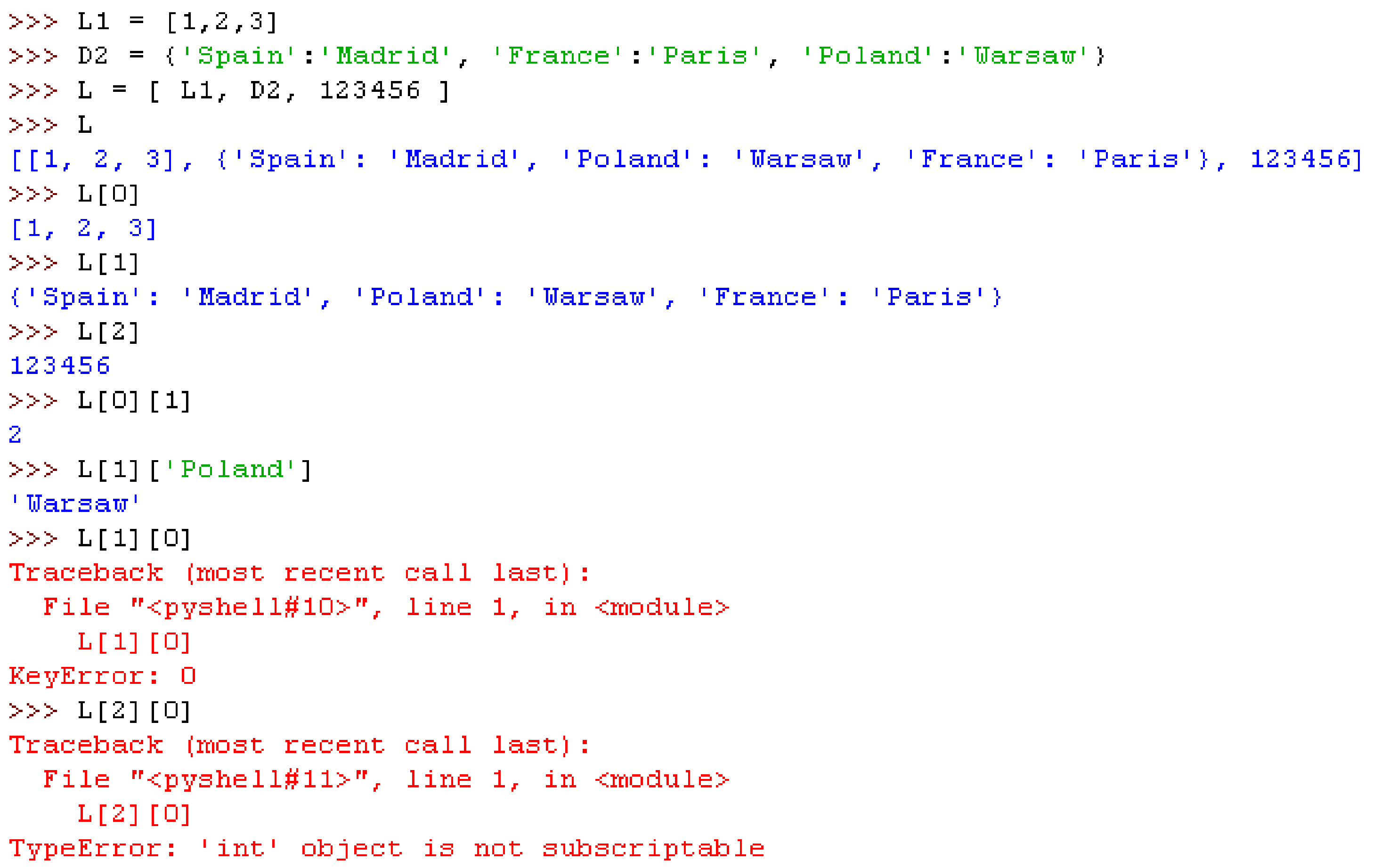
Now we will go a step further and we will create a list of lists (nested list or matrix).



```
>>> L = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
>>> L
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> L1 = [1,2,3]
>>> L2 = [4,5,6]
>>> L3 = [7,8,9]
>>> L = [L1,L2,L3]
>>> L
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> L[0][0]
1
>>> L[0][1]
2
>>> L[0][2]
3
>>> L[1][0]
4
>>> L[1][1]
5
>>> L[-1][-1]
9
```



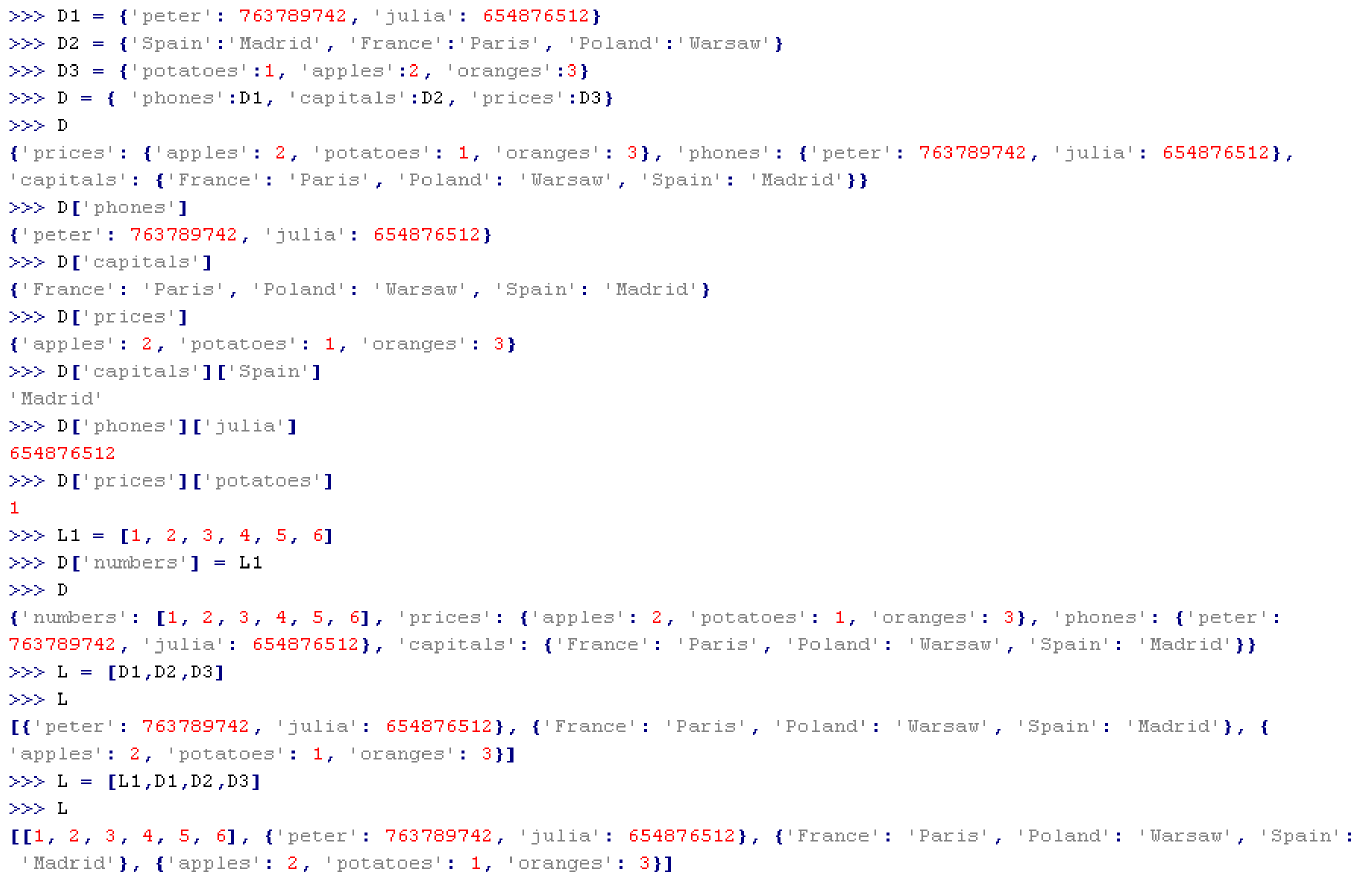
More complexity: nested lists



```
>>> L1 = [1,2,3]
>>> D2 = {'Spain':'Madrid', 'France':'Paris', 'Poland':'Warsaw'}
>>> L = [ L1, D2, 123456 ]
>>> L
[[1, 2, 3], {'Spain': 'Madrid', 'Poland': 'Warsaw', 'France': 'Paris'}, 123456]
>>> L[0]
[1, 2, 3]
>>> L[1]
{'Spain': 'Madrid', 'Poland': 'Warsaw', 'France': 'Paris'}
>>> L[2]
123456
>>> L[0][1]
2
>>> L[1]['Poland']
'Warsaw'
>>> L[1][0]
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    L[1][0]
KeyError: 0
>>> L[2][0]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    L[2][0]
TypeError: 'int' object is not subscriptable
```

More complexity: nested dictionaries

It is also possible to create a dictionary of dictionaries or dictionary of lists or list of dictionaries and mix…



```
>>> D1 = {'peter': 763789742, 'julia': 654876512}
>>> D2 = {'Spain':'Madrid', 'France':'Paris', 'Poland':'Warsaw'}
>>> D3 = {'potatoes':1, 'apples':2, 'oranges':3}
>>> D = { 'phones':D1, 'capitals':D2, 'prices':D3}
>>> D
{'prices': {'apples': 2, 'potatoes': 1, 'oranges': 3}, 'phones': {'peter': 763789742, 'julia': 654876512},
'capitals': {'France': 'Paris', 'Poland': 'Warsaw', 'Spain': 'Madrid'}}
>>> D['phones']
{'peter': 763789742, 'julia': 654876512}
>>> D['capitals']
{'France': 'Paris', 'Poland': 'Warsaw', 'Spain': 'Madrid'}
>>> D['prices']
{'apples': 2, 'potatoes': 1, 'oranges': 3}
>>> D['capitals']['Spain']
'Madrid'
>>> D['phones']['julia']
654876512
>>> D['prices']['potatoes']
1
>>> L1 = [1, 2, 3, 4, 5, 6]
>>> D['numbers'] = L1
>>> D
{'numbers': [1, 2, 3, 4, 5, 6], 'prices': {'apples': 2, 'potatoes': 1, 'oranges': 3}, 'phones': {'peter':
763789742, 'julia': 654876512}, 'capitals': {'France': 'Paris', 'Poland': 'Warsaw', 'Spain': 'Madrid'}}
>>> L = [D1,D2,D3]
>>> L
[{'peter': 763789742, 'julia': 654876512}, {'France': 'Paris', 'Poland': 'Warsaw', 'Spain': 'Madrid'}, {
'apples': 2, 'potatoes': 1, 'oranges': 3}]
>>> L = [L1,D1,D2,D3]
>>> L
[[1, 2, 3, 4, 5, 6], {'peter': 763789742, 'julia': 654876512}, {'France': 'Paris', 'Poland': 'Warsaw', 'Spain':
 'Madrid'}, {'apples': 2, 'potatoes': 1, 'oranges': 3}]
```

Copying data in Python

Copying data in Python

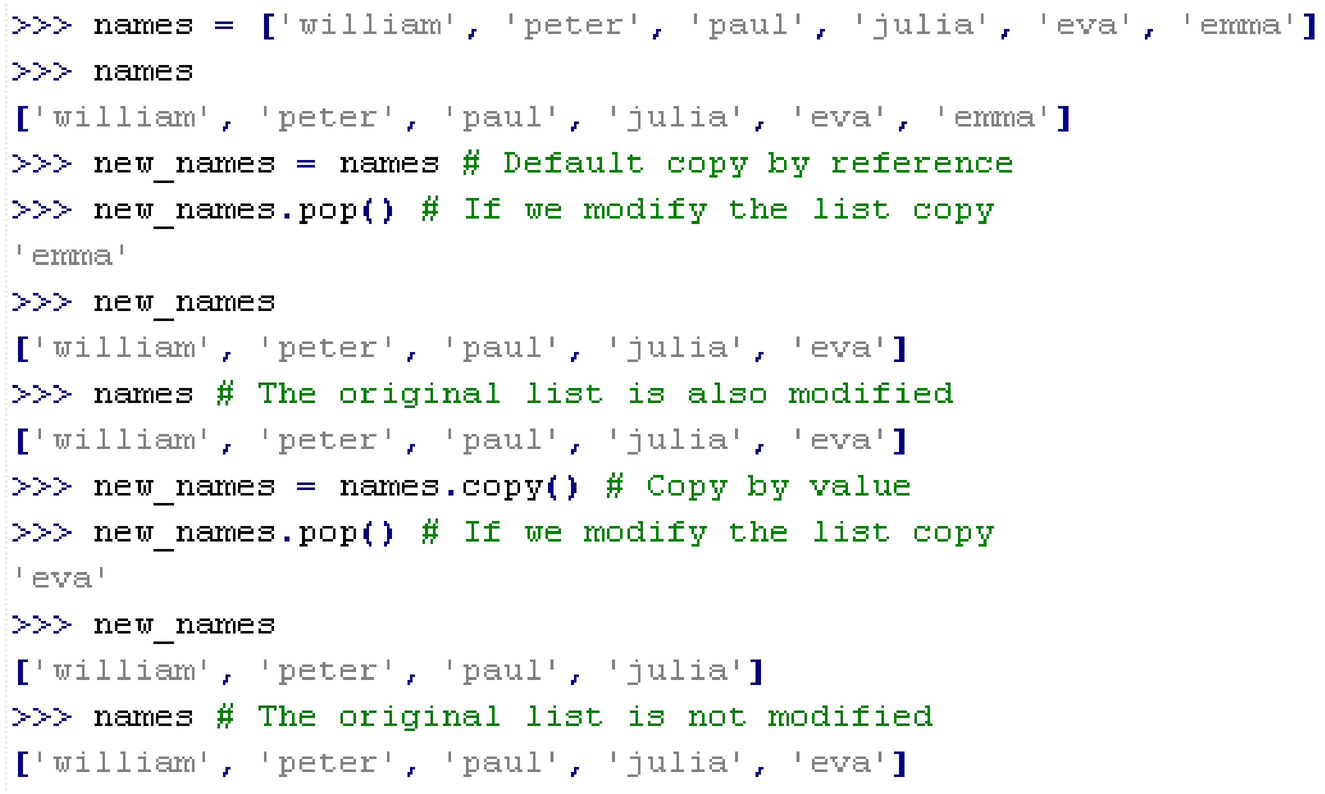
There are two ways to copy data in Python: by value or by reference.

The default Python mode is **copy by reference**: when we copy a variable, we are copying the positions in the memory of its contents. This means that when we modify the copy of the variable we will modify the original variable too, because both store the same data in the same memory positions.

Copy by value is the most intuitive mode for begginers, in this case, when we copy a variable we copy all its contents and we duplicate them in new memory positions. In this way when we modify the copy we will not modify the original.

Copying data in Python

list.copy() method allows us to copy by value:

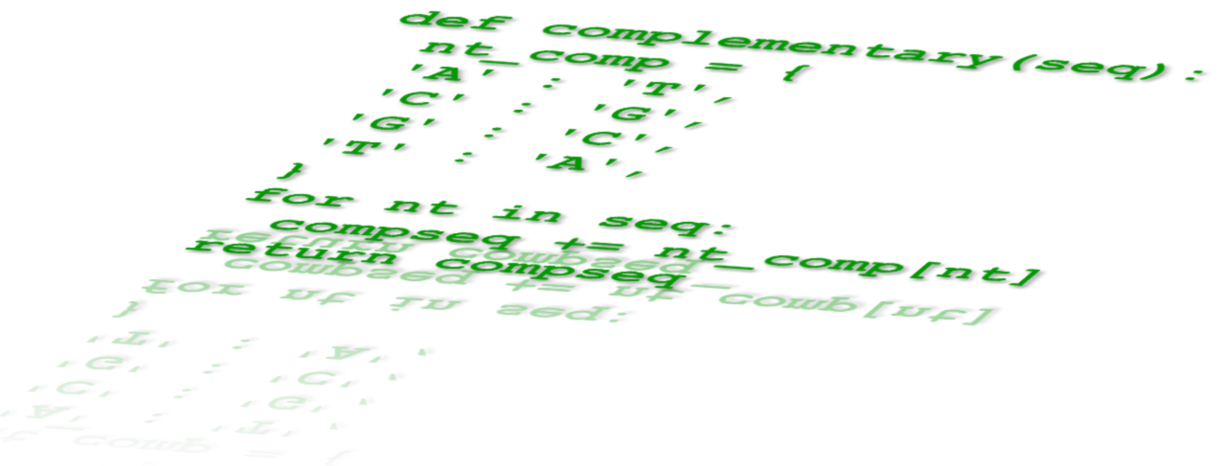


```
>>> names = ['william', 'peter', 'paul', 'julia', 'eva', 'emma']
>>> names
['william', 'peter', 'paul', 'julia', 'eva', 'emma']
>>> new_names = names # Default copy by reference
>>> new_names.pop() # If we modify the list copy
'emma'
>>> new_names
['william', 'peter', 'paul', 'julia', 'eva']
>>> names # The original list is also modified
['william', 'peter', 'paul', 'julia', 'eva']
>>> new_names = names.copy() # Copy by value
>>> new_names.pop() # If we modify the list copy
'eva'
>>> new_names
['william', 'peter', 'paul', 'julia']
>>> names # The original list is not modified
['william', 'peter', 'paul', 'julia', 'eva']
```

Python for scientists

Next lesson...

Data comparison and conditional statements



```
def complementary(seq):
    nt_comp = {
    'A' : 'T',
    'C' : 'G',
    'G' : 'C',
    'T' : 'A',
    }
    for nt in seq:
        compseq += nt_comp[nt]
    return compseq
```