Tuple/Dictionary/Set

Tuple
- immutable however can be ***mutable***

```
>>> c_tp = (1, [2, 6], 3, 'name')        # or just, a_tp = 1,2,3
>>> c_tp[1]
    ?
>>> c_tp[1].append(10)
>>> c_tp
(1, [2, 6, 10], 3, 'name')
          # can be modified if an obj. inside the tuple is mutable

>>> b_tp + c_tp                          # + operator, no -
('1', '2', '3', '4', '5', 1, [2, 6, 10], 3, 'name')

>>> b_tp*2                               # * operator, no /
                ?
```

Dictionary
(*hash map* or *associative array*)

```
#  Dict is a flexibly sized collection of key-values pairs, i.e.
#  maps a key on a specific value,
#  very important Python data structure.
#  Duplication of keys is not allowed, but values.
>>> ch_sbl= {
                        "H": "hydrogen",
                        "He": "helium",
                        "Li": "lithium",
                        "C": "carbon",
                        "O": "oxygen",
                        "N": "nitrogen"
                  }

                  # How to get the value for a given key, e.g. 'H'
>>> ch_sbl['H']   # case sensitive, ['h'] won't work
'hydrogen'
```

Valid dict *key* & *value* types

- The **value**s of a dict : any Python obj.
- **Key**s are *hashable* (can be hashed), i.e. immutable obj. e.g. scalar types, tuples. An object is *hashable*: it has a hash value which **never changes** during its lifetime

Key - any immutable values: numbers, strings, tuples (not list, dictionary)

```
>>> atm_nr = {
                        1: "hydrogen",
                        6: "carbon",
                        7: "nitrogen",
                        8: "oxygen"
                  }

>>> nobel_prize_winner = {
          (1979, "physics"): ["Glashow", "Salam", "Weinberg"],
          (1962, "chemistry"): ["Hodgkin"],
          (1984, "biology"): ["McClintock"]
                                        }
>>> nobel_prize_winner[1979, "physics"]    # [(1979,'physics')] is also fine.
['Glashow', 'Salam', 'Weinberg']
```

# Dictionary

```
>>> ch_sbl={'C': 'carbon', 'H': 'hydrogen', 'O': 'oxygen', 'N': 'nitrogen', 'Li': 'lithium', 'He': 'helium'}
>>> ch_sbl["C"]
'carbon'

>>> "O" in ch_sbl, "K" in ch_sbl, "oxygen" in ch_sbl
(True, False, False)    #  It is a tuple
```

# *Dictionary* checks only the keys existing in it, but not the values.

Dictionary - Default values, .get()

```
ch_sbl={'C': 'carbon', 'H': 'hydrogen', 'O': 'oxygen', 'N': 'nitrogen', 'Li': 'lithium', 'He': 'helium'}
```

```
>>> key='P'              # or  key='O'
>>> if key in ch_sbl:
          value = ch_sbl[key]    # when key='O'
    else:
          value = 'unknown'
          # A given default value when key='P'
>>> value
        ?
```

```
>>> key='O'
 >>> if key in ch_sbl:
         value = ch_sbl[key]
      else:
         value = 'unknown'
>>> value
```

```
# Instead of if/else, we can use .get method
>>> value = ch_sbl.get(key, 'unknown?')
>>> value
'unknown?'   #   when key='P'
'oxygen'     #   when key='O'
```

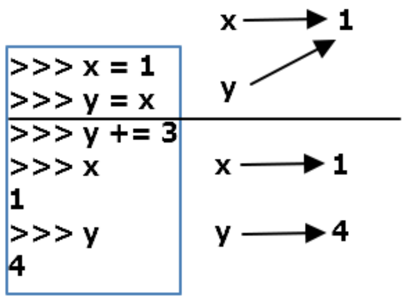
Useful dictionary methods

```
>>> ch_sbl.keys()      # return only keys
 ['C', 'H', 'O', 'N', 'Li', 'He']

>>> ch_sbl.values()   # return only values
 ['carbon', 'hydrogen', 'oxygen', 'nitrogen', 'lithium', 'helium']

>>> ch_sbl.update( {"P": "phosphorous", "S": "sulfur"} )
>>> ch_sbl.items()
 [('C', 'carbon'), ('H', 'hydrogen'), ('O', 'oxygen'), ('N', 'nitrogen'),
  ('P', 'phosphorous'), ('S', 'sulfur'), ('Li', 'lithium'), ('He', 'helium')]

>>> del ch_sbl['C']
>>> ch_sbl
{'H': 'hydrogen', 'O': 'oxygen', 'N': 'nitrogen', 'Li': 'lithium',
  'He':'helium'}
```

Mutable vs *immutable* variables

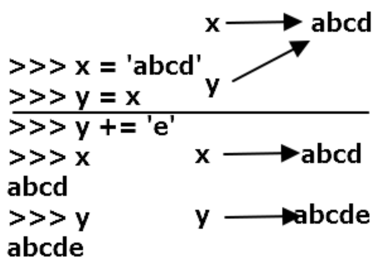
Call-By-Value, Call-By-Reference

- Number: immutable



```
>>> x = 1
>>> y = x
>>> y += 3
>>> x
1
>>> y
4
```

- String: immutable



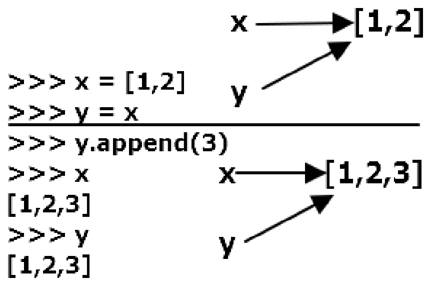
```
>>> x = 'abcd'
>>> y = x
>>> y += 'e'
>>> x
abcd
>>> y
abcde
```

Call-by-Value : conventional way. The **value** of an actual parameter (1) has been **copied** to formal parameters (y).

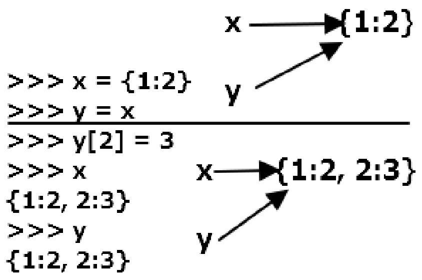
- List: mutable

- Dictionary: mutable

Mutable: "synchronised and simultaneous mutual changes happen"



```
>>> x = [1,2]
>>> y = x
>>> y.append(3)
>>> x
[1,2,3]
>>> y
[1,2,3]
```



```
>>> x = {1:2}
>>> y = x
>>> y[2] = 3
>>> x
{1:2, 2:3}
>>> y
{1:2, 2:3}
```

Call-by-Reference : instead of the parameter, the ***address*** is passed. The formal parameter is ***pointing*** to the actual parameters.

List and **dictionary** types are ***mutable***, otherwise all ***immutable*** in Python.

Set()

# A ***set()*** is an **unordered** collection of **unique elements**

```
>>> set1 = set([2,1,3])        # or just simply set1={2,1,3}
>>> set1
set([1, 2, 3])                 # unordered
>>> ~~set1[1]~~                    # neither order nor index
TypeError: 'set' object does not support indexing
>>> set2 = set([4,1,2,'three',2])
                               # collection of unique elements
>>> set2
set([1, 2, 4, 'three'])
>>> set3=set2                  # assignment
>>> set3
set([1, 2, 4, 'three'])
```

Data Structure - *String, Tuple, List*

- **String :** character (text) string, ***immutable***
  - ✓ a_str = 'abcde'
  - ✓ a_str = '12345'
  - ✓ ~~a_str[1]=8~~ # illegal
- **Tuple :** a fixed length composite data type, ***immutable***
  - ✓ a_tp = (1,2,3,4,5)
  - ✓ a_tp = 1,2,3,4,5
  - ✓ a_tp= 1,2,'a','b','3pf','@email.com
  - ✓ a_tp = tuple(a_str) # type casting a string to a tuple
  - ✓ ~~a_tp[1]=4~~ # illegal
- **List :** composite data type, ***mutable***
  - ✓ a_lst = [1,2,'a','b','3pf','@email.com']
  - ✓ a_lst[1]=4 # legal, Okay

Summary
- Python data structures

- string: " "
- tuple: ( ) or tuple()
- list: [ ]
- dictionary: { key1:'value1' }
- set: { } or set([ ])