Dictionaries

Dictionaries

Non-scalar (compound) and mutable type

dictionary (type `dict`) **: mapping type**
- mapping between a set of **keys** and a set of **values**
- set of pairs **key: value**
- key: unique, immutable type (`int`, `str`, `tuple`, ...)
- value: any type

values: items (pairs **key:value**) in curly braces (`{}`), separated by commas

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire', 'Chloe']}
>>> emptydict = {}
```

Dictionaries

Mapping between product names (string) and prices (float):

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
```

Mapping between letters (string) and list of names beginning with this letter:

```
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire', 'Chloe']}
```

Operations

- concatenation and replication: **not** supported
- **indexing by key**, operator: `[]` slicing: **not** supported
- membership operators by key: `in, not in`
- comparison operators: ==, != (only)
- function `len`

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire', 'Chloe']}
```

```
>>> d1['apples']   # access to an element
2.45
>>> d2['B']        # d2['B'] is a list
['Bertha', 'Blanche']
>>> d2['B'][1]     # second element of list d2['B']
'Blanche'
```

Operations

- indexing by key, operator: `[]`
- **membership operators by key**: `in`, `not in`
- comparison operators: ==, != (only)
- function `len`

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire','Chloe']}
```

```
>>> 'A' in d2
True
>>> 'm' in d2
False
>>> 'pears' in d1
True
>>> 'oranges' in d1
False
```

Operations

- indexing by key, operator: `[]`
- membership operators by key: `in`, `not in`
- **comparison operators**: ==, != (only)
- **function `len`**

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire','Chloe']}
```

```
>>> >>> d1 == d2
False
>>> len(d1)
3
>>> len(d2)
3
```

Updating operations

insert, delete, modify (update)

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire','Chloe']}
```

insert and **update** share the same syntax: `d[key] = value`

```
>>> d1 ['oranges'] = 1.98 # insert a new element 'oranges':1.98
>>> d1['apples'] = 2.9    # update an existing element 'apples':2.9
>>> d1 == {'apples': 2.9, 'cherries': 3.5, 'pears': 2.45, 'oranges': 1.98}
True
```

Updating operations

insert, delete, modify (update)

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d2 = {'A': ['Amelia', 'Ann'], 'B': ['Bertha', 'Blanche'],
...       'C': ['Claire','Chloe']}
```

```
>>> d2['D'] = ['Dolly']     # insert a new element
>>> d2['A'].append('Amy') # update item with key 'A': adding a name
>>> d2 == {'D': ['Dolly'], 'A': ['Amelia', 'Ann', 'Amy'],
... 'C': ['Claire', 'Chloe'], 'B': ['Bertha', 'Blanche']}
True

>>> del d2['C']
>>> d2 == {'D': ['Dolly'], 'A': ['Amelia', 'Ann', 'Amy'],
... 'B': ['Bertha', 'Blanche']}
True
```

item removal (**delete**) with `del` statement

`dict` methods

Methods: `keys, values, items`
Return an iterable that can be converted to list

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> list(d1.keys())
['cherries', 'pears', 'apples'] # list of keys
>>> list(d1.values())
[3.5, 2.45, 2.45]                # list of values

# list of tuples (key, value)
>>> list(d1.items())
[('cherries', 3.5), ('pears', 2.45), ('apples', 2.45)]
```

`dict` methods

Method: `get`
`get(k[,d]): D[k] if k in D, else d. d defaults to None`

is equivalent to:

```
def get_equivalent(D, k, d):
    # D: dict, k: key, d: default value when k not in D
    if k in D:
        return D[k]
    else:
        return d
```

```
>>> >>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d1.get('apples', 0.0)
2.45
>>> d1.get('oranges', 0.0)
0.0
```

Dictionaries mutability

dictionaries are mutable

to clone a dictionary: `dict` method `copy`
also with function `copy` of library `copy`

```
>>> d1 = {'apples': 2.45, 'pears':3.25, 'cherries':3.5 }
>>> d2 = d1                 # d1 and d2 are alias
>>> d3 = d1.copy()          # cloning: d3 is a clone

>>> # a change in d1 is also produced in d2 but not in d3
>>> d1['apples'] = 1.98
>>> d1 == {'cherries': 3.5, 'pears': 3.25, 'apples': 1.98}
True
>>> d2 == {'cherries': 3.5, 'pears': 3.25, 'apples': 1.98}
True
>>> d3 == {'apples': 2.45, 'pears': 3.25, 'cherries': 3.5}
True
```

Dictionaries mutability

dictionaries are mutable

to clone a dictionary: method `copy`

```
>>> # a change in d2 is also produced in d1 but not in d3
>>> del d2['pears']
>>> d1 == {'cherries': 3.5, 'apples': 1.98}
True
>>> d2 == {'cherries': 3.5, 'apples': 1.98}
True
>>> d3 == {'apples': 2.45, 'pears': 3.25, 'cherries': 3.5}
True

>>> d3['pears'] = 2.15  # changing d3 only affects d3
>>> d1 == {'cherries': 3.5, 'apples': 1.98}
True
>>> d2 == {'cherries': 3.5, 'apples': 1.98}
True
>>> d3 == {'apples': 2.45, 'pears': 2.15, 'cherries': 3.5}
True
```

Dictionaries: traversal

Process classification:

- dictionary creation or modification
- dictionary look up
- dictionary synthesis processes
  counting, summation, maximum search, ...

Exercises based on this dictionary structure:

A grocery keeps the data concerning fruits of its store in a dictionary such that the key is the fruit name (string) and the value is the number of Kg (integer) of that fruit in the store. In the following exercises, we will work with this dictionary example and we must assume that all the given structures are not empty.

Exercise 1

Write function `create_dict` that takes a list of tuples. Each tuple corresponds to an order and has two elements, a fruit name (string) and the number of Kg (integer) of this order. In this list a fruit name can appear in several orders. This function returns a dictionary as described. Save this function in file `create.py`. You can download this file with tests `test-create.txt`. Example:

```
>>> lorders = [('pears', 5), ('oranges', 10), ('pears', 4),
...            ('apples', 6), ('oranges', 8), ('pears', 8)]
>>> dfruit = create_dict(lorders)
>>> dfruit == {'pears':17, 'oranges':18, 'apples':6}
True
```

Exercise 1

Write function `create_dict` that takes a list of tuples. Each tuple corresponds to an order and has two elements, a fruit name (string) and the number of Kg (integer) of this order. In this list a fruit name can appear in several orders. This function returns a dictionary as described.

Dictionary creation from a list of tuples.

```
def create_dict(lorders):
  dfruits = {}
  for fruit, kg in lorders:
    if fruit in dfruits:
      dfruits[fruit] = dfruits[fruit] + kg
    else:
      dfruits[fruit] = kg
  return dfruits
```

initialize an empty dictionary

tuple unpack

if fruit is already in the dictionary: update the value

if fruit is not in the dictionary: add a new pair `fruit:kg`

Exercise 2

Write function `report_fruit` that takes a list of fruits (strings) and a dictionary as described. This function returns a list of tuples with the same length than the given one. Each tuple of the returned list has two elements, a fruit name (string) and the number of Kg (integer) of this fruit in the store (dictionary). Fruit names in the returned list are the same and in the same order than in the given fruit list. Save this function in file `report.py`. You can download this file with tests `test-report.txt`. Example:

```
>>> dfruits = {'pears':17, 'oranges':18, 'apples':6}
>>> lfruits = ['pears', 'mangoes', 'cherries', 'apples']
>>> lreturned = report_fruit(lfruits, dfruits)
>>> dfruits
{'pears':17, 'oranges':18, 'apples':6}
>>> lfruits
['pears', 'mangoes', 'cherries', 'apples']
>>> lreturned
[('pears', 17), ('mangoes', 0), ('cherries', 0), ('apples', 6)]
```

Cheking that the 2 mutable parameters have not been modified

Returned list with the same fruits as `lfruits`

Exercise 2

Write function `report_fruit` that takes a list of fruits (strings) and a dictionary as described. This function returns a list of tuples with the same length than the given one. Each tuple of the returned list has two elements, a fruit name (string) and the number of Kg (integer) of this fruit in the store (dictionary). Fruit names in the returned list are the same and in the same order than in the given fruit list. The returned list must be sorted by the fruit name.

Dictionary look up

```
def report_fruit(lfruits, dfruits):
  lnew = []
  for fruit in lfruits:
    if fruit in dfruits:
      lnew.append((fruit, dfruits[fruit]))
    else:
      lnew.append((fruit, 0))
  return lnew
```

dictionary look up

append a tuple

if fruit is not in the dictionary, kg = 0

Exercise 3

Write function `max_fruit` that takes a dictionary as described and returns the fruit name with more Kg in the store. Suppose that there are no repetitions of the number of kg. Save this function in file `max.py`. You can download this file with tests `test-max.txt`. Example:

```
>>> dfruits = {'pears':17, 'oranges':18, 'apples':6}
>>> max_fruit(dfruits)
'oranges'
```

Exercise 3

Write function `max_fruit` that takes a dictionary as described and returns the fruit name with more Kg in the store. Suppose that there are no repetitions of the number of kg.

Dictionary traversal, synthesis process: maximum search

```
def max_fruit(dfruits):
    maxkg = 0
    for fruit in dfruits:
        if dfruits[fruit] > maxkg:
            maxkg = dfruits[fruit]
            maxfruit = fruit
    return maxfruit
```

dictionary traversal through keys:
`fruit` is the key

dictionaries do not have integer indexes:
traversal `for i in range`… makes nonsense

Testing with dictionaries

Dictionaries are not a sequence type: there is **NO predefined order** in dictionary items.
Two dictionaries are the same if they have the same elements: the order is not relevant.
Tests for the correctness of a result dictionary involve dictionary comparison.

```
>>> d1 = {'apples': 2.45, 'pears':2.45, 'cherries':3.5 }
>>> d1 ['oranges'] = 1.98
>>> d1['apples'] = 2.9
>>> if d1 != {'apples': 2.9, 'cherries': 3.5, 'pears': 2.45, 'oranges': 1.98}:
...     print(d1)
```

Usual test for dictionaries: compare the result dictionary with the expected one
If d1 is equal to the expected dictionary it continues with the next test
But if `d1` is not equal to the expected dictionary, it shows `d1` so that we can check possible errors in it