COMP 204
Dictionaries

Mathieu Blanchette, based on material from Carlos Oliver Gonzalez and Christopher Cameron
Note about two-dimensional lists

In your assignment #2, you will need to represent two-dimensional tables, with a fixed number of rows and columns. Two-dimensional lists can be used to do this in Python.

A two-dimensional list is a list of lists, where each of the lists is of the same length. Example: A tic-tac-toe grid:

```python
tictactoe = [ ['X', '', 'O'], [' ', 'X', ''], ['O', '', ''] ]

print(tictactoe)  # [['X', '', 'O'], [' ', 'X', ''], ['O', '', '']]

# to access an element in a 2D list, # specify the index of the row and column
tictactoe[1][2] = 'X'

print(tictactoe)  # [['X', '', 'O'], [' ', 'X', 'X'], ['O', '', '']]
```
Note about two-dimensional lists

Example: A position frequency matrix (see assignment #2).

```python
# A position frequency matrix of 4 rows and 6 columns
PFM = [
    [0, 4, 2, 5, 1, 3],
    [5, 11, 4, 10, 6, 5],
    [0, 0, 7, 0, 4, 7],
    [10, 0, 2, 0, 4, 0]
]

print(PFM[0][2]) # 6
```
Creating two-dimensional lists

To create a new 2D list filled with zeros:

```python
# Creating a two-dimensional list of 4 rows and 5 columns, filled with zeros
nrows = 4
cols = 5
PFM = [[0 for i in range(cols)] for j in range(nrows)]
print(PFM)
```
Copying 2D lists

Because lists are compound objects, we need to be careful when copying them.

```
nrows = 4
ncols = 5
PFM = [[0 for i in range(ncols)] for j in range(nrows)]
```
Copying 2D lists

Because lists are compound objects, we need to be careful when copying them.

\[ PFM[1][3] = 9 \]
Copying 2D lists

Because lists are compound objects, we need to be careful when copying them.

```python
newPFM = PFM
```

![Diagram showing the copying of 2D lists](image)
Copying 2D lists

Because lists are compound objects, we need to be careful when copying them.

```
PFM[1][2] = 7
print(newPFM[1][2])  # 7
```
Copying 2D lists

Cloning PFM results in newPFM being a different list object than PFM. However, the elements of newPFM are the same 1D lists as the elements of PFM.

newPFM = PFM[:]

# cloning PFM

PFM

newPFM

[ , , , , ]

[0,0,0,0,0]
[0,0,0,0,0]
[0,0,0,0,0]
[0,0,7,9,0]
[ , , , , ]
Copying 2D lists

So changing a value in PFM still changes the value in newPFM!

```
PFM[1][2] = 8
print(newPFM[1][2]) # 8
```
Copying 2D lists

The correct way to clone a 2D list is:

```
newPFM = [ row[:] for row in PFM]
```
Copying 2D lists

Now the two 2D lists share no elements, and change values in one does not change the values in the other.

```
PFM[1][2] = 9
print(newPFM[1][2])  # 8
```
A dictionary is said to be a *mapping* type because it maps *key* objects to *value* objects.

Dictionaries are immensely useful and are the magic behind a lot of Python functionality.

Syntax:

```python
my_dict = { [key1]: [value1], [key2]: [value2], ... }
```

The analogy to a real dictionary works. The word you look up is the **key** and the definition is the **value**
# this dictionary maps strings to integers

```python
periodicTable = {
    "H": 1,
    "C": 12,
    "N": 14,
    "O": 16
}
```

```
periodicTable["He"] = 4  # adds key "He" with value 4
periodicTable["Na"] = 23  # adds key "Na" with value 23
```

**periodicTable** dictionary:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;H&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;C&quot;</td>
<td>12</td>
</tr>
<tr>
<td>&quot;N&quot;</td>
<td>14</td>
</tr>
<tr>
<td>&quot;O&quot;</td>
<td>16</td>
</tr>
</tbody>
</table>

**elementCodes** dictionary:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Hydrogen&quot;</td>
<td>&quot;H&quot;</td>
</tr>
<tr>
<td>&quot;Carbon&quot;</td>
<td>&quot;C&quot;</td>
</tr>
<tr>
<td>&quot;Nitrogen&quot;</td>
<td>&quot;N&quot;</td>
</tr>
<tr>
<td>&quot;Oxygen&quot;</td>
<td>&quot;O&quot;</td>
</tr>
</tbody>
</table>
Accessing elements in a dictionary

```python
# this dictionary maps strings to integers
periodicTable = {"H":1, "C":12, "N":14, "O":16}

periodicTable["He"] = 4  # adds key "He" with value 4
periodicTable["Na"] = 23  # adds key "Na" with value 23

# periodicTable now contains 6 keys, value pairs

periodicTable["C"] = 12.01  # overwrites value for key "C"

del periodicTable["N"]  # deletes key "N" and its value 14
```
About keys and values

Keys:

- Have to be immutable objects: int, float, str, tuple.
- Have to be unique in the dictionary: A dictionary cannot contain two elements with the same key.

Values:

- Values can be any type of object: int, float, str, tuple, list, dictionary, etc.
- Many keys can map to the same value

A dictionary can contain keys of many different types, and values of many different types:

```python
# a dictionary with keys and values of different types
mixedDict = {
    "H": "Hydrogen",
    17: "prime",
    30: [1, 2, 3, 5],
    (4, 5): 20
}

product = mixedDict[(4, 5)]  # 20
primeFactors = mixedDict[30]  # [1, 2, 3, 5]
fac = mixedDict[20]          # KeyError: 20 not in mixedDict
```
Dictionaries of dictionaries

The values stored in a dictionary can themselves be dictionaries!

```python
# a dictionary where each value is itself a dictionary
periodicTable = {
    "H" : { "name" : "Hydrogen", "mass" : 1},
    "C" : { "name" : "Carbon", "mass" : 12},
    "N" : { "name" : "Nitrogen", "mass" : 14},
    "O" : { "name" : "Oxygen", "mass" : 16}
}

carbonDic = periodicTable["C"]  # {"name":"Carbon","mass":12}
m = carbonDic["mass"]  # 12

# or more directly
m = periodicTable["C" ]["mass"]  # 12
```
Adding and deleting key/value pairs to a dictionary

Adding new key/value pairs:
- Syntax: myDict[ key ] = value
- If key does not already exist in the dictionary, the new key/value pair is added
- If the key already exists, its previous value is overwritten

Deleting key/values: del myDict[ key ]

```python
# this dictionary maps strings to integers
periodicTable = {"H": 1, "C": 12, "N": 14, "O": 16}

periodicTable["He"] = 4  # adds key "He" with value 4
periodicTable["Na"] = 23  # adds key "Na" with value 23

# periodicTable now contains 6 keys, value pairs

periodicTable["C"] = 12.01  # overwrites value for key "C"

del periodicTable["N"]  # deletes key "N" and its value 14
```
Iterating through dictionaries

The function `keys()` returns the keys present in the dictionary.

```python
per = {"H": 1, "C": 12, "N": 14, "O": 16}
```  
1. `keyList = list(per.keys())`  
   # ["H", "C", "N", "O"]
   # Note: the keys() function returns an object of
   # type dict_keys. This object is converted to a
   # list using the list() function

2. ```python
   for k in keyList:
       print("Key", k, "has value", per[k])
```  

The function `items()` returns key/value tuples in the dictionary

```python
per = {"H": 1, "C": 12, "N": 14, "O": 16}
```  
3. `itemList = list(per.items())`
   # Note: the items() function returns an object of
   # type dict_items. This object is converted to a
   # list using the list() function

4. ```python
   # itemList is now a list of tuples:
   # [('H', 1), ('C', 12), ('N', 14), ('O', 16)]

   for k, v in itemList:
       print("Key", k, "has value", v)
```
More functions on dictionaries

To test if a key is present in a dictionary, use the in operator: key in myDict, which evaluates to True if key is in myDict.

```python
periodic = {'H': 1, 'C': 12, 'N': 14, 'O': 16}
newElement = 'Na'
if newElement in periodic:
    print("Na is already in the dictionary")
else:
    print("Na is not in the dictionary")
```

To add the content of one dictionary, use the update() function.

```python
per = {'H': 1, 'C': 12, 'N': 14, 'O': 16}
newTable = {'Na': 23, 'K': 39}

# Add the content of newTable to per
per.update(newTable)  # per now has 6 elements
# newTable still has 2
```

For more functions on dictionaries:

https://docs.python.org/3/library/stdtypes.html#mapping-types-dict
Example

Goal: Count the number of occurrences of all characters in a string.

```python
sequence = "Hello my name is Mathieu"
counts = {}  # an empty dictionary
for nuc in sequence:
    if (nuc not in counts):
        counts[nuc] = 1
    else:
        counts[nuc] += 1
print(counts)
```
Example

Goal: Compute the mass of a molecule based on its chemical composition. Assume that you have access to a dictionary of atomic masses.

```python
periodicTable = {"H":1, "C":12, "N":14, "O":16}
aceticAcid = "CHHHCOOH"
mass = 0
for element in aceticAcid:
    mass += periodicTable[element]
print("Mass of acetic acid is", mass)
```
Example

Goal: Create a dictionary using as keys the english name of molecules and as values their molar mass. Assume that you have access to a dictionary of atomic masses and a dictionary of chemical compositions:

```python
periodicTable = {"H":1, "C":12, "N":14, "O":16}
molecules = {"Carbon dioxide":"COO", "Nitric oxyde":"NO", "Acetic acid":"CHHHCOOH"}
moleculeMass = {}  # the new dictionary we are about
                    # to populate with name/mass pairs
for name, composition in molecules.items():
    mass = 0
    for atom in composition:
        mass += periodicTable[atom]
moleculeMass[name] = mass
print(moleculeMass)
```