Conatiners in this course...

So far, we saw the following containers in this course:

- lists
  
  ```python
  a = [1,2,'some text',[-20.3,\sin(0.8)]]
  ```

- tuples
  
  ```python
  b = 3,4,5,'some text'
  ```

- dictionaries
  
  ```python
  c = {'age':23,'length':1.84,'town':'Tomellila'}
  ```

- sets
  
  ```python
  d = {'banana','apple','kiwi'}
  ```

We now look at these in more detail.
Creating sublists

Slicing – \( L[i:j] \)

*Slicing* a list between index \( i \) and \( j \) is forming a new list by taking elements with indices starting at \( i \) and ending *just before* \( j \).

Alternatively ...

\( L[i:j] \) means: create a list by removing the first \( i \) elements from \( L \) and keeping the next \( j - i \) elements (for \( j > i \geq 0 \)) and removing the rest.

Example

\[
L = ['C', 'l', 'o', 'u', 'd']
\]
\[
L[1:4] \quad \# \text{ remove one element and take three from there:}
\quad \# \quad ['l', 'o', 'u']
\]
Partial slicing

One may omit the first or last bound of the slicing:

```
L = ['C', 'l', 'o', 'u', 'd']
L[1:]  # ['l', 'o', 'u', 'd']
L[:3]  # ['C', 'l', 'o']
L[-2:] # ['u', 'd']  # negative indexing !
L[:-2] # ['C', 'l', 'o']
L[:]   # the entire list
```

Mathematical Analogy

This is similar to half lines in $\mathbb{R}$: $[-\infty, a)$ means: take all numbers strictly lower than $a$, cf. syntax of $L[:j]$.

Let's sum it up

- $L[i:]$ take all elements except the $i$ first ones
- $L[:i]$ take the $i$ first elements
- $L[-i:]$ take the last $i$ elements
- $L[:i]$ to take all elements except the $i$ last ones
Examples

- \( L[2:] \)
- \( L[:2] \)
- \( L[:\ -2] \)
- \( L[\ -2:] \)
- \( L[2:\ -1] \)
- \( L[2:5] \)
- \( L[\ -4:\ -1] \)
Dictionaries

A dictionary is an unordered structure of (key, value) pairs. It is similar to a list but the the objects are accessed by *keys* instead of *index*.

One indicates dictionaries by square brackets:

```
homework = {'Anna': True, 'Kerstin': False}

homework['Anna']  # True
# changing a value:
homework['Kerstin'] = True
# deleting an item
del homework['Anna']
```

_Note:_ Any *immutable* (unchangable) object can be used as a key, e.g. strings, numbers, Booleans, etc. are fine, but not lists.
Looping through Dictionaries

A dictionary is an object with the following useful methods:

keys, items, values

By default a dictionary is considered as a list of keys:

```python
for key in homework:  # or homework.keys()
    print('{k} {res}'.format(k=key, res=homework[key]))
```

One may also use items to loop through keys and values:

```python
for key, value in homework.items():
    print('{k} {v}'.format(k=key, v=value))
```

You can also print just the values by using the values method:

```python
for value in homework.values():
    print('{v}'.format(v=value))
```
Dictionaries in this course

Dictionaries are mainly used for

- providing functions with arguments in a compact way (see Unit 4).
- to collect options of a method:

  ```python
  {'tol':1.e-3,'step size': 0.1, 'maxit':1000}
  ```
Sets

Sets are containers which mimic sets in their mathematical sense:

**Definition**
A *set* is a collection of well defined and distinct objects, considered as an object in its own right. (Wikipedia)

```python
fruitbasket = set(['apple', 'pear', 'banana'])
```

The most important operation is *in*, meaning $\in$ (is element of):

```python
'plum' in fruitbasket  # returns False  
'pear' in fruitbasket  # returns True
```
Operations on Sets

Operations on sets are $A \cap B$, $A \cup B$ and $A/B$ (intersection, union and relative complement):

```python
fruithbasket = set(['apple', 'pear', 'banana'])
rotten = set(['pear'])
exotic = set(['mango', 'kiwi'])
emptyset = set([])  # the empty set
# Are all fruits edible (non-rotten)?
emptyset == fruithbasket.intersection(rotten)
# Add some diversity
extendedbasket = fruithbasket.union(exotic)
# The edible fruits
goodfruits = fruithbasket - rotten  # rel complement
```

The following syntax can also be used:

```python
exotic = {'mango', 'kiwi'}
```

Compare with dictionaries: {'a': 'mango', 'b': 'kiwi'}
No duplicate elements

“...distinct objects ...” (see mathematical definition)

This is reflected in Python by

```python
set({'apple', 'apple', 'pear'})
== set({'apple', 'pear'})  # True
```
**Tuples**

**Definition**
A *tuple* is an *immutable* list. Immutable means that it cannot be modified.

**Example**

```python
definition = (1, 2, 3)  # our first tuple!
definition = 1, 2, 3  # parentheses not required
definition = len(definition)  # 3, same method as for lists

definition[0] = 'a'  # error! tuples are immutable

singleton = 1,  # note the comma
singleton = (1,)  # preferred for readability
len(singleton)  # 1
```
Packing and unpacking

One may assign several variables at once by *unpacking* a list or tuple:

```python
a, b = 0, 1  # a gets 0 and b gets 1
a, b = [0, 1]  # exactly the same effect
(a, b) = 0, 1  # same
[a,b] = [0,1]  # same thing for lists
```

The swapping trick!

Use packing and unpacking to *swap* the contents of two variables.

```python
a, b = b, a
```
A function may return several values:

```python
def argmin(L):  # return the minimum and index
    ...
    return minimum, minimum_index  # a tuple

min_info = argmin([1, 2, 0])
min_info[0]  # 0
min_info[1]  # 2
# often unpacking is used directly
m, i = argmin([1, 2, 0])  # m is 0, i is 2
```
A final word on tuples

► Tuples are *nothing else* than immutable lists
► In most cases lists may be used instead of tuples
► The parentheses free notation is nice but *dangerous*, you should use parentheses when you are not sure:

```python
a, b = b, a  # the swap trick, equivalent to:
(a, b) = (b, a)
# but
1, 2 == 3, 4  # returns (1, False, 4)
(1, 2) == (3, 4)  # returns False
```
## Conversions

Containers can be converted to other container types:

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Tuple</td>
<td><code>tuple([1, 2, 3])</code></td>
</tr>
<tr>
<td>Tuple</td>
<td>List</td>
<td><code>list((1, 2, 3))</code></td>
</tr>
<tr>
<td>List, Tuple</td>
<td>Set</td>
<td><code>set([1,2,3]), set((1,2,3))</code></td>
</tr>
<tr>
<td>Set</td>
<td>List</td>
<td><code>list({1,2,3})</code></td>
</tr>
<tr>
<td>Dictionary</td>
<td>List</td>
<td><code>{‘a’: 4}.values()</code></td>
</tr>
<tr>
<td>List</td>
<td>Dictionary</td>
<td>–</td>
</tr>
</tbody>
</table>
## Summary and Overview

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Order</th>
<th>Duplicate Values</th>
<th>Mutability</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>index</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Tuple</td>
<td>index</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Dictionary</td>
<td>key</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Set</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>