Lists in Python

Lists are an abstract data type which stores elements in an ordered manner. In Python, a list is created by enclosing the elements in square brackets [], separated by commas.

```
list = [1, 2, 3]
print(list)
```

Lists can contain different types of data: integers, real numbers, characters, lists, tuples.

```
#list of integers
int_list = [1, 2, 3]
#list of characters
char_list = ['a', 'b', 'c']
#list of mixed types
mixed_list = [1, 'a', [2]]
```

Accessing list elements via indexes

We can use the index [] operator to access elements of a list. Indices start at 0, so a list with 3 elements will have an indices from 0 to 2.

```
list = [1, 2, 3, 4, 5]

print(list[0]) #first element
print(list[1]) #second element
print(list[-1]) #last element
print(list[-2]) #penultimate element
```

List Slicing

We can access an array of elements in a list using the truncation operator : (colon).
When cutting lists, the start index is inclusive, and the end index is exclusive. For example, list[2:4] returns a list with elements at indices 2 and 3, but not 4.

```
list = [1, 2, 3, 4, 5]
print(list[2:4])
>>> [3, 4]
```

The head, the tail of the list

The head of the list contains only the first element of the list - accessed via the index [] operator. The tail of the list contains all the remaining elements from the second to the last list element - accessed by the cut operation.

```
list = [1, 2, 3, 4, 5]
# head
print(list[0])
#tail
print(list[1:])
```

Concatenation of two lists

We can use the + operator to combine two lists.

```
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7]
concat = list1 + list2
print(concat)
```

Traversing a list recursively

Note: to solve the exercises in this lab, do not use repetitive structures (for, while), use recursion!

We use the notions of head and tail presented previously to go through the lists.

```
def display(list):
     if len(list) >= 1:
          head = list[0] # first item in list
          tail = list[1:] # all remaining items
          print(head)
          display(tail)

display([1, 2, 3, 4, 7])
```

Accessing an index that does not exist

```
list = [1, 2, 3]
print(list[7])
>>> IndexError: list index out of range
```

The reduce() function

The reduce() function is defined in the functools module. It receives two arguments, a function and an iterable (in our case, a list) and returns a single value. Also, the reduce() function has an optional argument: an initial value. If this initial value is present, it will be placed before all elements in the calculation.

```
functools.reduce(function, iterable, initial_value)
```

To calculate the sum of all the integers in a list we can use reduce() together with an anonymous function (defined with lambda).

```
import functools
sum = functools.reduce(lambda a, b: a + b, [1, 2, 3])
print(sum)
```

We can also use the operator module, learned in laboratory 2.

```
import functools
import operator
sum = functools.reduce(operator.add, [1, 2, 3])
print(sum)
```

The filter() function

In a similar way to the reduce() function, the filter() function takes two arguments: a function and an iterable. However, instead of returning a single value, it returns another iterable. As its name suggests, the function creates a list of elements for which the function returns true (it filters the original list's elements).

```
# even numbers
result = list(filter(lambda x: x % 2 == 0, [1, 2, 3, 4, 5]))
print(result)
```

The map() function

Like the filter() and reduce() functions, the map() function takes two arguments: a function and an iterable. The function applies the function received as an argument to all elements in the list.

```
result = list(map(lambda x: x+1, [1, 2, 3, 4, 5]))
print(result)
```

List methods (functions on lists)

Python has a set of predefined methods that allow working with lists:

```
append()    #Adds an element to the end of the list
clear()     #Clear all elements from the list
copy()      #Returns a copy of the list received as a parameter
extend()    #Appends the elements of a list to the end of the
             current list
index()     #Returns the index of the first occurrence of the
             element received as a parameter
insert()    #Adds an element at the specified position
pop()       #Delete the element at the specified position
remove()    #Remove the first occurrence of the element given as
             a parameter
reverse()   #Reverses the order of the elements in the list
sort()      #Sort the list
```