Lists

Lists are a basic concept in computer science.

It is essentially an ordered set of data. Ordered = each element has its place (1st, 2nd, ...)

In [13]:
```
a = [1,2,3,4]
print (a)
```

```
[1, 2, 3, 4]
```

A list can have values and variables of different types (numbers, decimals, strings, ...)

In [2]:
```
a = [1,2,3,"mitsos", 5, 7.77777778]
print (a)
```

```
[1, 2, 3, 'mitsos', 5, 7.77777778]
```

The elements of a list are accessed just like the letters in strings:

In [3]:
```
a[0] # First element
```

Out[3]:
```
1
```

In [4]:
```
a[0:3] # All elements from first until the 4th without the 4th.
```

Out[4]:
```
[1, 2, 3]
```

In [5]:
```
a[-1] # The last element
```

Out[5]:
```
7.77777778
```

In [6]:
```
a[-2:] # One but the last element
```

Out[6]:
```
[5, 7.77777778]
```

Similarly, we can use steps. Suppose the following list:

In [8]:
```
b = [1,2,3,4,5,6]
```

In [9]:
```
b[2:5:2] # From the 3rd until the 6th element (without taking 6th) with st
```

Out[9]:
```
[3, 5]
```

In [21]:
```
b[::2] # From start until the end with step 2
```

Out[21]:
```
[1, 3, 5]
```

In [10]:
```
b[:3] # From the start unitl the 4th element (without taking the 4th eleme
```

Out[10]:
```
[1, 2, 3]
```

In [11]:
```
# All below are equal
print (b)
print (b[:])
print (b[::])
print (b[::1])
print (b[0:])
print (b[0::])
print (b[0::1])
print (b[:len(b)])
print (b[:len(b):])
print (b[0:len(b)])
print (b[0:len(b):1])
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

In [12]:
```
b[-1:0:-1] # From the end until the first (without taking the first)
```

Out[12]:
```
[6, 5, 4, 3, 2]
```

In [13]:
```
b[-1::-1] # From the end until the first (we also take the first)
```

Out[13]:
```
[6, 5, 4, 3, 2, 1]
```

In [14]:
```
b[::-1] # This is equivalent with the above
```

Out[14]:
```
[6, 5, 4, 3, 2, 1]
```

Like strings, we can apply `len` , `count` , `index` to lists.

In [27]:
```
a = [1,2,3,"mitsos", 5, 7.77777778]
```

In [15]:
```
len(a) # Number of all elements in the list
```

Out[15]:
```
6
```

In [16]:
```
a.count(1) # How many times does exist in the list?
```

Out[16]:
```
1
```

In [17]:
```
a.count(55) # How many times does 55 exist in the list?
```

Out[17]:
```
0
```

In [18]:
```
a.index("mitsos") # What is the index of "mitsos" in the list?
```

Out[18]:
```
3
```

In [19]:
```
a.index(4) # What is the index of 4 in the list?
```

```
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-19-79f48df914e9> in <module>
----> 1 a.index(4) # What is the index of 4 in the list?

ValueError: 4 is not in list
```

A list may contain other lists!

```
In [130…  a = [1,2, [3,4,5], 6, 7]
```

```
In [34]:  len(a)
```

```
Out[34]: 5
```

```
In [35]:  a[2]
```

```
Out[35]: [3, 4, 5]
```

```
In [36]:  a[1]
```

```
Out[36]: 2
```

```
In [131…  a[2][1]
```

```
Out[131… 4
```

We can change the contents of any item in a list:

```
In [132…  a = [1,2,3,4,5]
          a[2] = 8
          print (a)
```

```
[1, 2, 8, 4, 5]
```

```
In [133…  a[3] = ['Mitsos', 'Kwstas']
          print (a)
```

```
[1, 2, 8, ['Mitsos', 'Kwstas'], 5]
```

Caution! this is not allowed in strings:

```
In [134…  a = 'Mitsos'
          a[2] = 'k'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-134-2320b677b6ef> in <module>
      1 a = 'Mitsos'
----> 2 a[2] = 'k'

TypeError: 'str' object does not support item assignment
```

If you want to read more about why this is the case then google: "Why are Python Strings Immutable?"

There is also the empty list: `[]`

```
In [38]:  a = []
          print (len(a))
```

```
0
```

We can write lists in many ways:

In [20]:
```
# The following 2 are equal:
a = [1,2,3]

a = [
    1,
    2,
    3,
]
```

Caution! It is absolutely ok to add a comma right after the last element in a list!

In [21]:
```
# These two are equal:
print ([1,2,3])
print ([1,2,3,])
```

```
[1, 2, 3]
[1, 2, 3]
```

So it is fine to add a comma after the last element. But is **not fine** if we do NOT put a comma in between the elements:

In [22]:
```
a = [
    'aaaa',
    'bbbb' # CAUTION! This string is merged with the one below!
    'cccc'
]
print (len(a))
```

```
2
```

In [23]:
```
print (a)
```

```
['aaaa', 'bbbbcccc']
```

A list can have a list, which has a list that has ..

In [42]:
```
a = [[]]
print (len(a))
```

```
1
```

In [43]:
```
a = [[[[[[[[[[[[[[[[[[[]]]]]]]]]]]]]]]]]]]
print (len(a))
```

```
1
```

In [44]:
```
a = [1,2,3,[],4]
```

In [45]:
```
len(a)
```

Out[45]:
```
5
```

In [46]:
```
a=3
b = [a,a,a+1, a/2]
print (b)
```

```
[3, 3, 4, 1.5]
```

We can concatenate two lists:

In [48]:
```
[1,2,3] + ["mitsos", "a"]
```

Out[48]:
```
[1, 2, 3, 'mitsos', 'a']
```

We can multiply a list by a number:

```
In [49]: [1,2,3] *4

Out[49]: [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

We can not multiply or subtract two lists!

```
In [50]: [1,2,3] * [5,6]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-4f81883a1dc4> in <module>
----> 1 [1,2,3] * [5,6]

TypeError: can't multiply sequence by non-int of type 'list'
```

```
In [51]: [1,2,3] - ["mitsos", "a"]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-d5954fea0119> in <module>
----> 1 [1,2,3] - ["mitsos", "a"]

TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

The `list` function converts various data types into lists:

```
In [52]: list("mitsos")

Out[52]: ['m', 'i', 't', 's', 'o', 's']
```

```
In [24]: list([1,2,3]) # This does nothing..

Out[24]: [1, 2, 3]
```

Not everything can be turned into a list:

```
In [54]: list(5)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-0c7f5cd48ec1> in <module>
----> 1 list(5)

TypeError: 'int' object is not iterable
```

A list is always True, unless it is empty:

```
In [57]: if [1,2,3]:
             print ("not empty")

not empty
```

```
In [56]: if []:
             print ("not empty")
         else:
             print ("is empty!")

is empty!
```

Comparing strings

When the comparison operators (`<` , `>` , `<=` , `>=`) are applied to strings, then we compare them alphabetically. String `a` is "lower" than `b` if `a` is placed in a lower index than `b` if we sort `a` and `b` .

```
In [1]: 'ab' < 'fg'
```

```
Out[1]: True
```

```
In [2]: 'ab' < 'b'
```

```
Out[2]: True
```

```
In [3]: 'ab' < 'ac'
```

```
Out[3]: True
```

```
In [4]: 'ab' < 'a'
```

```
Out[4]: False
```

The empty string has the lowest possible value

```
In [5]: '' < '0'
```

```
Out[5]: True
```

```
In [6]: "A" < "a"
```

```
Out[6]: True
```

```
In [7]: "05456745674" < "5"
```

```
Out[7]: True
```

```
In [8]: '8' < '09'
```

```
Out[8]: False
```

The `in` operator

This operator checks if there is "something" "somewhere"

```
In [9]: 'rak' in 'Heraklion'
```

```
Out[9]: True
```

```
In [10]: 'raki' in 'Heraklion'
```

```
Out[10]: False
```

```
In [11]: 'h' in 'Heraklion'
```

```
Out[11]: False
```

```
In [12]: 'H' in 'Heraklion'
```

```
Out[12]: True
```

```
In [58]: 1 in [1,2,3]
```

```
Out[58]: True
```

```
In [59]: [1,2] in [1,2,3]
```

```
Out[59]: False
```

```
In [60]: [1,2] in [1, [1,2], 3]
```

```
Out[60]: True
```

```
In [61]: False in [1, True-True]
```

```
Out[61]: True
```

```
In [63]: None in [3, None, 4]
```

```
Out[63]: True
```

```
In [65]: 'ra' in ['Heraklion']
```

```
Out[65]: False
```

```
In [94]: [] in [1, [], 2]
```

```
Out[94]: True
```

map and filter

We can apply a function to all elements of a list with the `map` function:

```
In [97]: def f(x):
             return x+1

         a = [4,5,6]

         list(map(f,a))
```

```
Out[97]: [5, 6, 7]
```

We can get a subset of a list of elements that have a property with the `filter` function. The filter must return something that can be valued as `True` or `False` .

```
In [100…  def is_even(x):
              # Return True / False depending in whether x is even or not
              return x%2==0

          a = [1,2,3,4,5,6,7,8,9]
          list(filter(is_even,a))
```

```
Out[100… [2, 4, 6, 8]
```

```
In [26]: def is_first_vowel(x):
             # Return True or False depending to whether x starts with a vowel
             return x[0].lower() in 'aeiouy'

         a = ['Ioannina', 'Thessalonikh', 'Athena']
         list(filter(is_first_vowel,a))

Out[26]: ['Ioannina', 'Athena']
```

Operations on lists

Some of the operations that we can do on lists are:

```
In [27]: sum([2,3,4]) # The sum of all elements

Out[27]: 9
```

Attention `sum` must be applied to lists containing only `int` or `float` values

```
In [68]: sum(['a', 'b'])

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-0ca1e4efb8fe> in <module>
----> 1 sum(['a', 'b'])

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
In [28]: min([3,5,4]) # minimum element

Out[28]: 3
```

```
In [29]: max(['heraklion', 'patras', 'athens']) # Maximum element

Out[29]: 'patras'
```

```
In [71]: max(['heraklion', 'patras', 't'])

Out[71]: 't'
```

`max` and `min` have a very interesting property. When applied to lists that contain other lists, then it tries to compare these elements with the following logic: Compare first element of both lists, if they are the same then move to the second element, if both second elements are the same then move to the third etc.

```
In [31]: min([5, "b"], [6, "a"]) # 5 is lower than 5 so the second element is not c

Out[31]: [5, 'b']
```

```
In [33]: # 5 is equal to 5 so the second elements ("b" and "a") are alse checked:
         min([[5, "b"], [5, 'a']])

Out[33]: [5, 'a']
```

```
In [34]: # Another example:
         min([[5, "b"], [7, "t"], [6, 'r'], [5, 'a']])

Out[34]: [5, 'a']
```

And that's because:

```
In [74]: [5, 'a'] < [5, 'b']
```

```
Out[74]: True
```

Insert and remove items to a list

Remember that we can concatenate two lists:

```
In [103…: [1,2,3] + ['Μίτσος', 7.8]
```

```
Out[103…: [1, 2, 3, 'Μίτσος', 7.8]
```

We can use this property to add any item to a list anywhere:

```
In [35]: # Insert at the end:
         a = [1,2,3]
         a = a + ['Μήτσος']
         print (a)
```

```
[1, 2, 3, 'Μήτσος']
```

Remember that `a = a + b` is equivalent to `a += b` :

```
In [36]: # Insert at the end:
         a = [1,2,3]
         a += ['Μήτσος']
         print (a)
```

```
[1, 2, 3, 'Μήτσος']
```

```
In [37]: # Insert at the beginning:
         a = [1,2,3]
         a = ['Μήτσος'] + a
         print (a)
```

```
['Μήτσος', 1, 2, 3]
```

```
In [38]: # Insert at any index (we use the slicing trick)
         a = [1,2,3]
         a = a[:2] + ['Μήτσος'] + a[2:]
         print (a)
```

```
[1, 2, 'Μήτσος', 3]
```

We can use the append, extend and insert functions instead:

```
In [112…: a = [1,2,3]
          a.append('Mitsos') # equivalent to: a += ['Mitsos']
          print (a)
```

```
[1, 2, 3, 'Mitsos']
```

```
In [114…: a = [1,2,3]
          a.extend(['Mitsos', 7.8]) # equivalent to: a += ['Mitsos', 7.8]
          print (a)
```

```
[1, 2, 3, 'Mitsos', 7.8]
```

```
In [116…: a = [1,2,3]
          a.insert(2, 'Mitsos') # equivalent to: a = a[:2] + ['Mitsos'] + a[2:]
          print (a)
```

```
[1, 2, 'Mitsos', 3]
```

To remove an item we can use slicing again:

```
In [119…
a = [1, 2, 'Mitsos', 3]
a = a[:2] + a[3:]
print(a)
```

```
[1, 2, 3]
```

But we can also use `del` :

```
In [120…
a = [1, 2, 'Mitsos', 3]
del a[2]
print (a)
```

```
[1, 2, 3]
```

Another way is to just use `filter` :

```
In [123…
a = [1, 2, 'Mitsos', 3]

def remove_mitsos(x):
    return x != 'Mitsos'

a=list(filter(remove_mitsos, a))
print(a)
```

```
[1, 2, 3]
```

Sorting

With the command `sorted` we can sort a list:

```
In [75]: a = [3,4,5,3,2,1]
```

```
In [76]: sorted(a)
```

```
Out[76]: [1, 2, 3, 3, 4, 5]
```

Caution! `sorted` does NOT change the list. Instead it returns the result to another variable:

```
In [77]: a
```

```
Out[77]: [3, 4, 5, 3, 2, 1]
```

```
In [78]: b = sorted(a)
```

```
In [127… b
```

```
Out[127… [1, 2, 3, 3, 4, 5]
```

if we want to change our list (sorting in place) we should use the `sort` function:

```
In [39]:
a = [3, 4, 5, 3, 2, 1]
a.sort()
print (a) # a changed!
```

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

We can only sort lists that have elements that can be compared:

In [80]:
```
sorted(["b", "a", "c"])
```
Out[80]:
```
['a', 'b', 'c']
```
In [81]:
```
sorted(["b", "a", 100, "c"])
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-b245b7eeb5df> in <module>
----> 1 sorted(["b", "a", 100, "c"])

TypeError: '<' not supported between instances of 'int' and 'str'
```
We can sort from largest to smallest:

In [82]:
```
sorted([3,4,5,2,3,4,5,2,1], reverse=True)
```
Out[82]:
```
[5, 5, 4, 4, 3, 3, 2, 2, 1]
```
As with `min` and `max` , if sort a list of lists (or tuple), then it first checks the first element of the list. If it is equal, then check the second etc:

In [83]:
```
a = [
    ["mitsos", 50],
    ['gianni', 40],
    ['gianni', 30]
]
sorted(a)
```
Out[83]:
```
[['gianni', 30], ['gianni', 40], ['mitsos', 50]]
```
In the example above `['gianni', 30]` is less than `'gianni', 40]` :

In [84]:
```
['gianni', 30] < ['gianni', 40]
```
Out[84]:
```
True
```
Sometimes we may want to sort a list that contains sublists but we want the sorting to be done not based on the first element but on our own function. E.g. Given the list:

In [85]:
```
a = [["gianni", 30, 20000], ["mitsos", 50, 4000], ["anna", 60, 100000]]
```
Suppose we want to sort the list items based on their third item (20000, 4000, 100000). Be careful that if we apply the `sorted` command then it will not return what we want:

In [86]:
```
sorted(a)
```
Out[86]:
```
[['anna', 60, 100000], ['gianni', 30, 20000], ['mitsos', 50, 4000]]
```
We want the list whose third element is 4000 to come out first. Then the list whose third element is 20000 to come out second and finally the list whose third element is 100000 to come out last.

In this case we can create a function which takes as an argument an element of a list and returns the value through which the sorting will be done:

In [87]:
```
def sort_according_to_this(x):
    return x[2]
```

I can test this function by calling it with various elements of my list. It should return the value for which I want the sorting to be based on:

```
In [88]: sort_according_to_this(a[0])
```

```
Out[88]: 20000
```

```
In [89]: sort_according_to_this(a[1])
```

```
Out[89]: 4000
```

```
In [90]: sort_according_to_this(a[2])
```

```
Out[90]: 100000
```

Now I can pass `sort_according_to_this` to `sorted` as an argument to the sorting function and sort list `a` according to the third element of each of its elements:

```
In [91]: sorted(a, key=sort_according_to_this)
```

```
Out[91]: [['mitsos', 50, 4000], ['gianni', 30, 20000], ['anna', 60, 100000]]
```

Another example. Let the list be:

```
In [124…: a = ["heraklion", "patras", "thessaloniki", "athens"]
```

The following command sorts the list according to the length of the strings:

```
In [93]: sorted(a, key=len)
```

```
Out[93]: ['patras', 'athens', 'heraklion', 'thessaloniki']
```

The `max` and `len` functions also support key = ...:

```
In [125…: min(a,key=len) # H polh me to mikrotero onoma
```

```
Out[125…: 'patras'
```

```
In [126…: max(a,key=len) # H polh me to megalutero onoma
```

```
Out[126…: 'thessaloniki'
```

A thought experiment

Perhaps a function that sorts according to a sorting function might be difficult to understand. This is a thought experiment that could make this a bit more clear. Suppose that someone hands you the task to "sort all countries of Europe". What would be the immediate qeustion after this from you? "Sort according to what?". A sorting of a list of numbers or a list of strings is straightforward. What we usually want is an ordering from lower to greater or an alphabetical ordering. But what if a list refers to a real world entity (countries of Europe) or contains a complex item like list of lists... ?

So let's return to the countries of Europe:

In [40]:
```
# A random list of european countries
countries = ['France', 'Germany', 'Italy', 'Greece', 'Spain', 'Belgium', ''
```

How can we order them? The default alphabetical sorting is of no interest:

In [41]:
```
sorted(countries)
```

Out[41]:
```
['Belgium', 'France', 'Germany', 'Greece', 'Italy', 'Spain', 'The Netherlan
ds']
```

Now suppose that we have a function the returns the GDP of each country:

In [ ]:
```
def get_GDP(country):
    return ... # Magic commands tha return the GDP of a country
```

Now we can sort the list of countries according to their GDP:

In [ ]:
```
sorted(countries, key=get_GDP)
```

In [ ]:
```

```