



# Introduction to the Python programming language

Laszlo SZATHMARY

University of Debrecen  
Faculty of Informatics

## Lab #9

- advanced sorting
- exception handling

(last update: 2016-11-02 [yyyy-mm-dd])

2016-2017, 1st semester

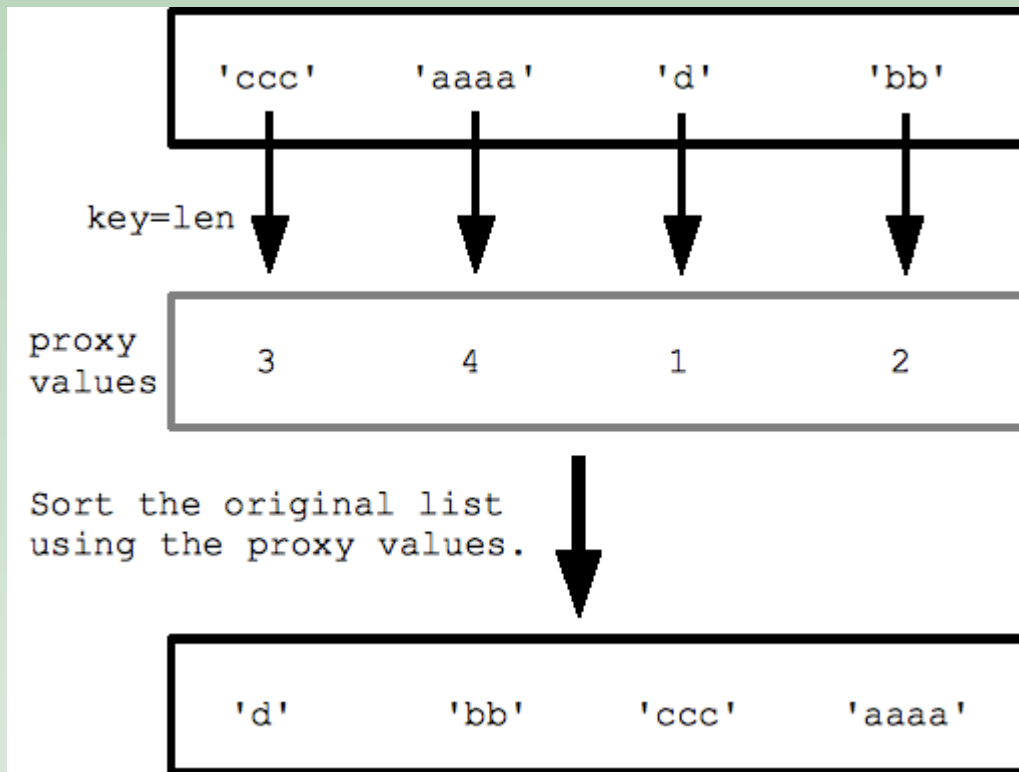


# Advanced sorting

If you want to do some more complex sorting, you can provide a „key” parameter to the `sorted()` function. The value of the „key” parameter is another function. Before sorting, this function transforms the elements. The function takes a value and it returns a new value. Sorting is based on the values that are returned by the function.

```
1 >>> words = ['ccc', 'aaaa', 'd', 'bb']
2 >>> sorted(words)
3 ['aaaa', 'bb', 'ccc', 'd']
4 >>>
5 >>> sorted(words, key=len)
6 ['d', 'bb', 'ccc', 'aaaa']
```

Here, the value of „key” is the built-in „len” function. We want to sort the words by their lengths. The function `len` is called with each element of the list, and the sorting is done by the values returned by `len`.



from the values returned by „len” a proxy list („shadow list”) is constructed, and this proxy list is sorted:  
[1, 2, 3, 4]

then replaced by the original values:

1 -> 'd'  
2 -> 'bb'  
3 -> 'ccc'  
4 -> 'aaaa'

```
1 >>> words = ['Cc', 'BB', 'aa', 'zz']
2 >>> sorted(words)
3 ['BB', 'Cc', 'aa', 'zz']
4 >>>
5 >>> sorted(words, key=str.lower)
6 ['aa', 'BB', 'Cc', 'zz']
```

sort the words in *ignore-case* mode, i.e.  
no difference between lowercase and  
uppercase

```
1 >>> words = ['xc', 'zb', 'yd', 'wa']
2 >>> sorted(words)
3 ['wa', 'xc', 'yd', 'zb']
4 >>>
5 >>> def my_func(s):
6 ...     return s[-1]
7 ...
8 >>> sorted(words, key=my_func)
9 ['wa', 'zb', 'xc', 'yd']
```

sort the words by their  
last character

you can provide  
your own function

# Exercise


Solve some exercises with advanced sorting.

Link: <https://arato.inf.unideb.hu/szathmary.laszlo/pmwiki/index.php?n=EnPy.20121006e>

# Exceptions

```
1  #!/usr/bin/env python
2
3  import sys
4
5
6  def cat(fname):
7      f = open(fname, 'r')
8      text = f.read()
9      print '---', fname
10     print text
11     f.close()
12
13     #####
14
15     if __name__ == "__main__":
16         args = sys.argv[1:]
17         for arg in args:
18             cat(arg)
```

this output is similar to  
the output of the Unix  
command `cat`



## Exercise:

Produce a warning message if  
a file doesn't exist, then  
continue processing the next  
argument.

see also <https://arato.inf.unideb.hu/szathmary.laszlo/pmwiki/index.php?n=EnPy.20121120a>

```
6 def cat(fname):
7     try:
8         f = open(fname, 'r')
9         text = f.read()
10        print '---', fname
11        print text
12        f.close()
13    except IOError:
14        print '--- I/O error:', fname
```

Exception is caught and handled during runtime.

If the file doesn't exist: warning message and the program goes on.

```
6 def cat(fname):
7     try:
8         f = open(fname, 'r')
9         text = f.read()
10        print '---', fname
11        print text
12        f.close()
13    except IOError as e:
14        print '--- I/O error:', e
```

„e” is the exception object

print the exception object

You can catch different kinds of exceptions in an except branch. In this case the types of the exceptions must be grouped in a **tuple**.

```
except (KeyboardInterrupt, EOFError):
```

# Exercise

Extend the script with exception handlers.

Link: <https://arato.inf.unideb.hu/szathmary.laszlo/pmwiki/index.php?n=EnPy.20121120b>





# Exercises

1. [[20130920a](#)] swap case
2. [[20120904b](#)] reverse a part of a list
3. [[20130326a](#)] last N lines, version **B** [bonus point: +1]