High-Level Programming Languages 3

The Python Programming Language

Laszlo SZATHMARY
University of Debrecen
Faculty of Informatics

Lab #9

- advanced sorting
- exception handling

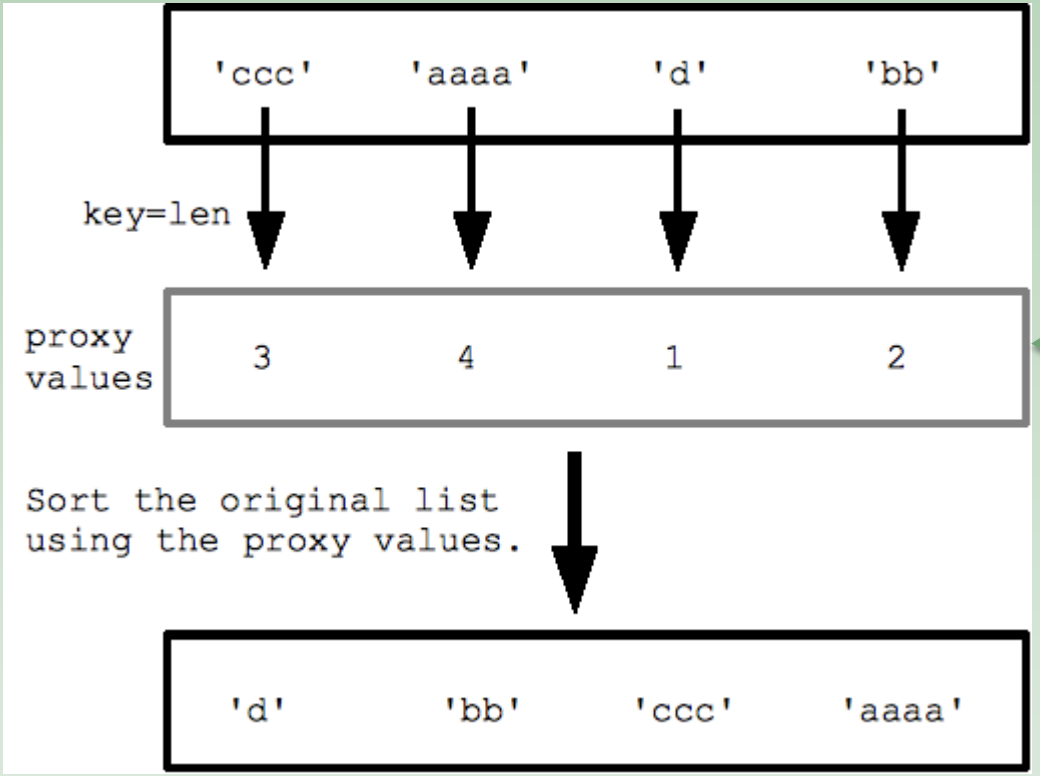
(last update: 2019-09-17 [yyyy-mm-dd])

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.

```
>>> words = ['ccc', 'aaaa', 'd', 'bb']
>>> sorted(words)
['aaaa', 'bb', 'ccc', 'd']
>>>
>>> sorted(words, key=len)
['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'
```

```
>>> words = ['Cc', 'BB', 'aa', 'zz']
>>> sorted(words)
['BB', 'Cc', 'aa', 'zz']
>>>
>>> sorted(words, key=str.lower)
['aa', 'BB', 'Cc', 'zz']
```

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

sort the words by their last character

```
>>> words = ['xc', 'zb', 'yd', 'wa']
>>> sorted(words)
['wa', 'xc', 'yd', 'zb']
>>>
>>> def my_func(s):
...     return s[-1]
...
>>> sorted(words, key=my_func)
['wa', 'zb', 'xc', 'yd']
```

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=EnPy3.20121006e

Exceptions

```
 1  #!/usr/bin/env  python3
 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=EnPy3.20121120a

```
 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

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

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**.

Exercise

Extend the script with exception handlers.

Link: https://arato.inf.unideb.hu/szathmary.laszlo/pmwiki/index.php?n=EnPy3.20121120b

Exercises

homework

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