F21SC Industrial Programming: Python: Revision

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 1 2015/16

[0]No proprietary software has been used in producing these slides

Case Study: Higher-order Functions in Python

- Python supports higher-order functions
- This means, functions can be assigned to variables
- The variable can then be called like a function
- See the `simple_histo.py` sample sources from the libraries slides.
- Functions can also be passed to other functions to modify the behaviour
- One concrete application of this is modifying how sorting works

The sample code is in ho_sort.py.

A simple example of using higher-order functions

Example

```
def my_cmp(x,y):
    """Custom comparison operator to return inverse of the
    return (-1)*(x-y);
...
  xs = mkList(n)
  ...
  ys = list(xs) # this clones the input
  ys.sort()     # this sorts the list
  zs = list(xs)
  zs.sort(key=functools.cmp_to_key(my_cmp)) # comp. functio
  ...
  zs = list(xs)
  zs.sort(key=functools.cmp_to_key(gcd_cmp)) # comp. functi
```

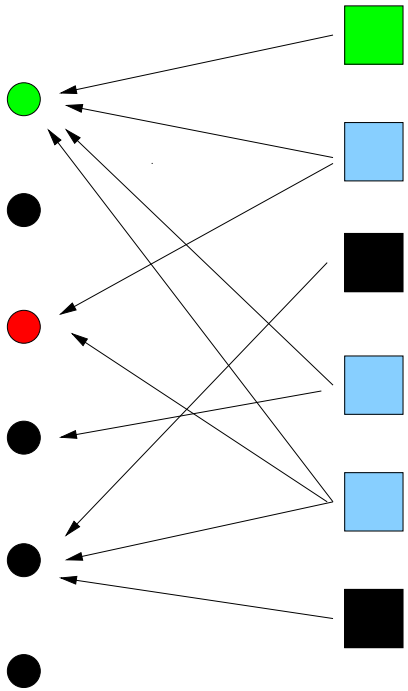
[0] In Python 2 use: `zs.sort(cmp=my_cmp)`

Towards the CW

An example closer to the CW is this:

- We have a list of persons, and each person has a set of **favourite numbers**
- We want to compute what is the most favourite number, i.e. one that appears most often in the favourite sets
- Note, that this structure is very similar to Task 5 in the CW

Documents Readers



Input

Also read ...

Result (most also readers)

Towards the CW

In our favourite number example

- we have numbers instead of documents (as circles)
- we have names instead of readers (as boxes)
- the structure of the computation we need to do is similar to the coursework

Simple solution to “favourite number”

Example

```
def mostFavNum(dict):
    """Return the most favourite number."""
    # we use sets to collect the numbers
    xs = set([])
    # iterate over the dictionary entries
    for k in dict.keys():
        xs = xs | set(dict[k])
    # decorate each number with the matches, and use this a
    xs_dec = [ (countMatches(x,dict), x) for x in xs ]
    # sort the list by first component in the tuple (no of
    xs_dec.sort()
    # return xs_dec[-10:-1]  # return largest 10 values, if
    n, x = xs_dec[-1] # largest elem
    return x          # return it's value
```

Higher-order version

- The basic idea is to turn `countMatches` into **a function argument**
- Then in the main part of the code you just need to call:

Example

```
print(mostFavNumGen(favs,countMatches))
```

- You can check the full code at ho_sort.py.
- Use this as an example how to tackle Task 5 in the CW.