

COM6115: Text Processing

Configuring Program Behaviour

Mark Hepple

Department of Computer Science
University of Sheffield

Configuring Program Behaviour

- Often want to *configure* the behaviour of a program, e.g. to:
 - ◊ specify files from which to take *input*
 - ◊ name of files to which to write *output/results*
 - ◊ *set* various *parameters*:
e.g. weight/threshold values, number of results to print, etc
- For *scientific computing*, often want to run program under a wide *range* of different *settings*:
 - ◊ i.e. so alternative results can be *compared*, *plotted*, etc.
- Might configure via a *GUI*, *but*
 - ◊ time-consuming to develop
 - ◊ time-consuming to use, if each configuration must be entered separately
- Alternative: configure via the *command line*
 - ◊ use '*flag*' symbols (e.g. '*-s*') to *name* specific command line options

Command Line Options

- Using command line options — e.g. might have call:

```
python myCode.py -w -t 0.5 -d data1.txt -r results1.txt
```

- ◇ options for input data (**-d**), results file (**-r**), a threshold value (**-t**), etc
- ◇ a *boolean* option **-w** to direct some aspect of behaviour
e.g. whether terms are *weighted* or not
- **Help option**: good practice to include a boolean *help* option **-h**:
 - ◇ if present, code *just prints help message* and *then quits*
 - ◇ help message says how to call program, lists options, etc
- Allows for use of *batch files*:
 - i.e. text file of commands, invoking program under range of settings
 - ◇ easy way to generate a range of experimental results
- Python modules for options handling:
 - ◇ for *serious* use (e.g. distributed software), use **argparse** module
 - but significant overhead for learning to use
 - ◇ **getopt** module provides a *lightweight* alternative

The getopt Module

- The **getopt** module helps with parsing command line options
 - ◇ allows both *short* options (**-s**) and *long* ones (**--long-option**)
 - ◇ here consider only short options
- Specify allowed options via a string, e.g. **'hi:o:I'**
 - ◇ each letter in string accepted as an option
 - ◇ letters followed by **":"** require an arg string, e.g. **-i** here
 - ◇ otherwise flag is boolean, e.g. **-h** here
- Command line strings given by list **sys.argv**
 - ◇ **sys.argv[0]** is name of your *code file* – don't pass this
 - ◇ hence, option parsing usually applied to **sys.argv[1:]**
- Example, call in your code might be:

```
opts, args = getopt.getopt(sys.argv[1:], 'hi:o:I')
```

- ◇ here, **opts** is the options found — given as a *list of pairs*
 - may be convenient to *convert* to *dictionary*, e.g. **opts = dict(opts)**
- ◇ **args** is any remaining 'bare' arguments – as a list
 - flag options should *precede* bare args on command line

The getopt Module — *example*

- Demo code file `getopt_demo.py` illustrates use of `getopt` module
 - ◇ run as shown below, in a terminal or CMD window

```
> python getopt_demo.py -h
-----
USE: python getopt_demo.py (options) input-file1 ... input-fileN
OPTIONS:
    -h : print this help message
    -s FILE : use stoplist file FILE (required)
    -b : use binary weighting (default is off)
-----

> python getopt_demo.py -s stoplist.txt -b file1.txt file2.txt
SUMMARY
Command line strings: ['getopt_demo.py', '-s', 'stoplist.txt',
                      '-b', 'file1.txt', 'file2.txt']
Arguments: ['file1.txt', 'file2.txt']
Options:
    Stopwords file: stoplist.txt
    Binary weighting: 1
>
```