Class 4: Adding command line arguments and config files to a script

-one level up from modularization using defined functions is making a script that can be used or reused as a module

command line arguments and config files

-Allow optional commands to your scripts!

* Argparse package
* Different kinds of args
* Always have a default

-Allow common options to be stored in a config file, and read at execution time!

* Configparser package
* The dict layout of config

Lecture

Command Line Arguments:

*Argparse*

[*https://docs.python.org/3.5/library/argparse.html*](https://docs.python.org/3.5/library/argparse.html)

*The most salient bits:* [*https://docs.python.org/3.5/library/argparse.html#the-add-argument-method*](https://docs.python.org/3.5/library/argparse.html#the-add-argument-method)

**3 kinds of command line arguments that we will go over today**

-flags

-string, e.g. directory

-list,

regular expressions (regex)

|  |  |
| --- | --- |
| **?** | The question mark indicates *zero or one* occurrences of the preceding element. For example, colou?r matches both "color" and "colour". |
| **\*** | The asterisk indicates *zero or more* occurrences of the preceding element. For example, ab\*c matches "ac", "abc", "abbc", "abbbc", and so on. |
| **+** | The plus sign indicates *one or more* occurrences of the preceding element. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac". |
| **{n}**[[19]](https://en.wikipedia.org/wiki/Regular_expression#cite_note-grep-19) | The preceding item is matched exactly *n* times. |
| **{min,}**[[19]](https://en.wikipedia.org/wiki/Regular_expression#cite_note-grep-19) | The preceding item is matched *min* or more times. |
| **{min,max}**[[19]](https://en.wikipedia.org/wiki/Regular_expression#cite_note-grep-19) | The preceding item is matched at least *min* times, but not more than *max* times. |

Sometimes you want to specify sometimes-options to a program (in the same way that you might give a function an *optional parameter*)

Important parts to check out:

* **Add\_argument**
  + The command line argument to assign to
* **Action**
  + What the storage action is
* **Default**
  + What the command line argument defaults to
* **Dest**
  + What variable the argument is assigned to
* **Help**
  + What to display when --help is used

Configuration Files

Sometimes you have something that has a set value (say, a directory location, or a default username) that will be different based on who is running it, or where it is being run. These are good candidates for a configuration file.

*YAML*

[*http://pyyaml.org/wiki/PyYAMLDocumentation*](http://pyyaml.org/wiki/PyYAMLDocumentation)

*Pip install PyYAML*

*YAML looks like:*

*treeroot:  
   branch1:  
       name: Node 1  
       branch1-1:  
           name: Node 1-1  
   branch2:  
       name: Node 2  
       branch2-1:  
           name: Node 2-1*

Load into python like:

*import yaml  
with open(config.yaml') as f:  
   # use safe\_load instead load  
   dataMap = yaml.safe\_load(f)*

The convenience of YAML is that in Python, it looks like an arbitrarily deep, nested dictionary!