# IEMS 308 Lab 2.2

Running scripts from command line

Andrea Treviño Gavito

2020

## argparse library

The argparse module is a convenient way to write user-friendly command-line interfaces. argparse deals with

- Parsing arguments.
- Generating help and usage messages.
- Error handling when invalid arguments are provided.

There are broadly two types of arguments: positional and optional

## Example I

File in repository: argparse_simple.py

```
# Simple example which returns the square of the given integer
import argparse

# Define the argument parser
parser = argparse.ArgumentParser(description="Returns the square of
    the given integer")
# Define arguments
parser.add_argument("value",help="The integer whose square is
    needed",type=int)
args = parser.parse_args()

# Print to stdout
print(args.value**2)
```

## Example II

What do the following commands return?

```
python argparse_simple.py --help

python argparse_simple.py 10

python argparse_simple.py 10.1
```

## Example I

File in repository: argparse_twoargs.py

```
import argparse

# Define the argument parser
parser = argparse.ArgumentParser(description="Returns the given
    power of the given integer")
# Define arguments
parser.add_argument("value",help="The integer",type=int)
parser.add_argument("power",help="The power",type=int)
args = parser.parse_args()

# Print to stdout
print(args.value**args.power)
```

## Exercise

Complete the script np_summary.py which reads the given file and print either the mean or standard deviation across columns, as specified by the user. To do this, define the argument parser with two positional arguments:

- filepath: a string containing the path to the file
- function: a string from ["mean","sd"] (use the argument `choices` when defining this argument)

Test your script on the iris dataset located in "../data/iris.csv"

## Optional arguments

- name in `add_argument` begins with - - to indicate that the argument is optional
- Unlike positional arguments, optional arguments can be specified in any order.
- By default, if the optional argument is not used, `None` is stored in the corresponding variable. You can change it by specifying `default=<default value>` when adding the argument.
- Short options: You can also add a shorter version for the argument using `-<letter>`. For example:
  `parser.add argument("-s","--slow",...)`

## Optional arguments - Example 1

File in repository: argparse_verbose.py

```
import argparse

# Define the argument parser
parser = argparse.ArgumentParser(description="Returns the given
    power of the given integer")
# Define arguments
parser.add_argument("value",help="The integer",type=int)
parser.add_argument("power",help="The power",type=int)
# arguments beginining with -- indicate optional arguments
parser.add_argument("-v","--verbose",default=0,type=int,
  help="increase verbosity")
args = parser.parse_args()

# Print to stdout
out = args.value**args.power
if args.verbose>=1:
  print("{}^{} equals {}".format(args.value,args.power,out))
else:
  print(out)
```

## Optional arguments - Example 2 (flags)

File in repository: argparse_verbose2.py

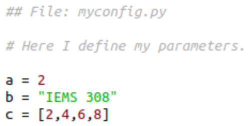
```
import argparse

# Define the argument parser
parser = argparse.ArgumentParser(description="Returns the given
    power of the given integer")
# Define arguments
parser.add_argument("value",help="The integer",type=int)
parser.add_argument("power",help="The power",type=int)
# arguments beginining with -- indicate optional arguments
parser.add_argument("-v","--verbose",action="store_true",
  help="verbose output")
args = parser.parse_args()

# Print to stdout
out = args.value**args.power
if args.verbose:
  print("{}^{} equals {}".format(args.value,args.power,out))
else:
  print(out)
```

## Parameter files

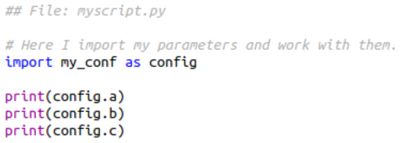
A convenient way to structure code is to define your parameters in a separate script and load them in the same way you load packages.

By doing this, if you want to test your script with several different values, you only need to modify the parameters file.

Files in repository: my_config.py and my_script.py



```
## File: myconfig.py

# Here I define my parameters.

a = 2
b = "IEMS 308"
c = [2,4,6,8]
```



```
## File: myscript.py

# Here I import my parameters and work with them.
import my_conf as config

print(config.a)
print(config.b)
print(config.c)
```

## Exercise: Putting it all together!

Complete the script classify.py which trains a classifer on a given dataset and returns the training score. It should take the following arguments:

- filepath: a string containing the path to the file
- header: binary variable ([0,1]) indicating whether the csv file has a header
- classifier: a string from ["logreg","svm","rf"] - type of classifier
- -n, -normalize: flag to normalize the features

Test it on the spambase and banana datasets. Sample commands (from script directory):

```
python classify.py --normalize ../data/spambase.data 0 logreg
python classify.py ../data/banana.csv 1 svm
```