Practical 6
Scripts and Automation

Learning Objectives

1. Understand and use Bash and Python as a scripting languages
2. Apply Python scripts to implement a parameter sweep using an existing program
3. Understand and modify supplied code to automate experiments

Overview

In this practical you will be writing Bash and Python scripts to automate the process of running experiments. We will start with some simple bash examples, then you will download and modify code, and write scripts to run the programs multiple times with different parameters.

Tasks

1. The Power of Unix

Each week we have be creating a README file to describe the programs we have written for each practical. So far that's five READMEs in different directories. It might be interesting to list the programs and their descriptions.

Begin by creating the directory for Prac6 and changing to that directory. We can start with listing the contents of all our Prac* directories. That's a level above where we are, so we need to use ../ . Try the following commands…

```
ls ../Prac*
ls –R ../Prac*
ls –l ../Prac*/*.py
ls –l ../Prac*/README*
```

What is the difference between the various commands?

If we want to pull information out of a file, we can use grep. Grep matches to the strings within each line of the file, and prints the ones that match.

```
grep .py ../Prac*/README*
grep .py ../Prac*/README* | wc
grep .py ../Prac*/README* | wc -l
grep .py ../Prac*/README* > myprogs.txt
```

Try these:

1. Find out which programs use matplotlib.pyplot using a grep command.
2. Calculate how many print statements are in each program (grep & wc)

2. Using command line arguments

In most of our programs, we have hard-coded the values into the variables. That means we have to edit the code to change and experiment with the variables. We may have a few variables we want to work without creating new versions of our program each time. We implement this through command line arguments or input files (as in heatsource.py). In this task we will use command line arguments.

Download **repeatdosageP.py** from Blackboard. It is a modified version of our previous program – providing more information about parameters and results. If you look at the min, max and mean values at the end of the program – there are two versions. As what we are really interested in with this experiment is the drug_in_system being between the MEC and MTC – a simple minimum and mean don't really help us. The code gives a simple approach to checking when the value first goes above MEC, and calculates the min, max and mean after that. Run the code to confirm what it does.

We will make a new version of the program, **dosagebase.py** to use command line arguments. Make the following changes at the start of the program:

```
import sys

# Process command line arguments

if (len(sys.argv) < 3):
   print('argv too short, usage: python3 <interval> <dosage>')
   print('Using default values for interval (8) and dosage 
(100)')
   Vinterval = 8         # 8 hours between doses
   Vdosage = 100         # dosage 100mg
else:
   Vinterval = int(sys.argv[1]) # hours between doses
   Vdosage = int(sys.argv[2])   # dosage
```

Then change the setup of the interval and dosage variables to use Vinterval and Vdosage. Test the program to see that it works. See how the output changes for different dosage and interval values.

Another change we might want is to stop the program plotting to the screen. We can save the plot to a file instead.

```
plt.savefig('dosage_' + 'I' + str(Vinterval)+'_D' + 
            str(Vdosage) + '.png')
```

Now our program is ready to be the base program for a parameter sweep. We'll create the parameter sweep program in the next task.

3. Run a parameter sweep

Our base program takes two command line variables: interval and dosage. We want to be able to sweep across a range of values for these two variables, to find workable ways to prescribe the medication. Of course, we could consider changing other variables – but we'll stick to two for now.

A bit like our range statements in Python, we'd like to be able to have a start, stop and step value, for each variable. Thus our script will take six command line arguments: *low_int high_int step_int low_dose hi_dose step_dose*

The driver script, **dosage_sweep.sh**, below creates a directory for the experiment, processes the command line arguments and then has two for loops to go through the parameter sweep. Inside the loop it calls the dosagebase.py program and redirects the output to a file. Type in the driver script and see if you can get it working.

```
#!/bin/bash

exp_dir=dosage`date "+%Y-%m-%d_%H:%M:%S"`

mkdir $exp_dir
cp dosagebase.py $exp_dir
cp dosage_sweep.sh $exp_dir
cd $exp_dir

low_int=$1
hi_int=$2
step_int=$3
low_dose=$4
hi_dose=$5
step_dose=$6

echo "Parameters are: "
echo "Interval : " $low_int $hi_int $step_int
echo "Dosage : " $low_dose $hi_dose $step_dose

for i in `seq $low_int $step_int $hi_int`;
do
    for d in `seq $low_dose $step_dose $hi_dose`;
    do
        echo "Experiment: " $i $d
        outfile="dosage_I"$i"_D"$d".txt"
        python3 dosagebase.py $i $d > $outfile
    done
done
```

If you look in the directory after the script has run, you will see the saved png plots and the txt files. You can look at the results using: `tail –n 3 *.txt`

Run the script with the following values:

```
          sh dosage_sweep.sh 4 12 4 50 300 50
```

Can you find the extreme results using tail? Look at the graphs to confirm the data.

4. Automatic weather plots

In the lecture we looked at a workflow for plotting data. The steps were:

1. Get data
2. Extract lines
3. Extract columns
4. Plot

Type in the commands from the lecture on the command line to ensure they work. You may need to add a –p to gnuplot to make the plot stay on the screen:

```
gnuplot –p plotcmd.txt
```

Use the history command and redirection to put the workflow commands into a file weather_workflow.sh

```
history 20 > weather_workflow.sh
```

Add **`#!/bin/bash`** as the first line of the file – this specifies the shell to use to run the program. Delete unnecessary lines (dd), remove numbers from the start of liens and make any other changes you need, then you should be able to run the code with:

```
sh weather_workflow.sh
```

Once you have this working, add code to the script to

1. Put command line arguments in to take in the year and month
2. Make a directory based on the year and month: plot_year_month
3. Copy the script and plotcmd.txt file into the directory
4. Change to the directory
5. Download the data – substituting the year and month into the URL
6. Extract the lines
7. Extract the columns
8. Plot the data

If you want to plot to a file, you need to set the terminal and output file name. See the example below:

```
gnuplot -e "set terminal png size 400,300; set output 'xyz.png'; plot [-
4:4] exp(-x**2 / 2)"
```

Submission

All of your work for this week's practical should be submitted via Blackboard using the link in the Week 6 unit materials. Make sure you have the directory named correctly and include a README file describing the contents of the directory.

Reflection

1. Knowledge:
2. Comprehension:
3. Application:
4. **Analysis**: Look at the parameter sweep outputs for an interval of 4 hours, with a dosage of 100. It provides a stable concentration – what would be the down-side to this schedule of medication
5. Synthesis:
6. Evaluation:

Challenge

A few extra items to try:

1. To read about a science project with data processing pipelines, view the Fireballs in the Sky guest lecture notes and have a look at their website http://fireballsinthesky.com.au/
2. Another version of the drug dosage program is available on blackboard. Download and run it. Consider the differences in implementation. What benefits are there in the alternative approach?