Lab 02: Python and the Command-line

CMPT 145

Laboratory 02 Overview

Part 1 : Pre-Reading: Your documents, files, and data

Part 2 : Pre-Reading: Executing Python Scripts from the Command-Line

Part 3 : Pre-Reading: Using Command Line Arguments in Python

Part 4 : Activities: Executing Python Scripts from the Command-Line

Part 5 : Activities: Using Command Line Arguments in Python

Hand In : A transcript of your work (see the final slide)

Part I

Pre-Reading: Your documents, files, and data

Your work in CMPT 145

- Your labs should be done using Linux or Mac.
    - Windows computers (probably) don't have UNIX tools installed.
- Your assignments can usually be done on PyCharm on any system (Linux, Mac, Windows).
- Sometimes an assignment question must be done on a UNIX system (Linux, Mac). We'll tell you when!

Network File Systems

- Modern computers/notebooks/tablets store documents locally by default.
    - Locally means one computer only.
- Network filesystems store documents and data on a remote computer.
    - Documents stored on networked filesystems are accessible from any computer connected to the network.
- When you log in to any Spinks lab computer, you are connected to the department's networked filesystems, and also the university's networked filesystems.

Your documents and data

- Python programs are documents (a.k.a. files).
- You could choose to store documents locally on a single computer.
    - Inconvenient! Local documents are not accessible if you move to another computer.
- You should choose to store documents (assignment work, lab work) on the network filesystem.
    - Convenient! You can change computers (even from Mac to Windows, etc), and your documents are accessible.
- You need to know where to put your documents and folders, and where to find them.

Your home folder

- When you log into a computer (Linux, Mac, Windows), you have direct access to a home folder.
- If you are using your own computer, your home folder is local only.
- If you log into a departmental computer, your home folder is on a network filesystem.
    - Your home folder is private to you, by default.
    - Private means other users are prevented from accessing your home folder's contents.
- You can access your documents from any other department computer, depending on what kind of computer you are working on.

Using a primary system

- When you are using a departmental lab computer, your home folder is on a network filesystem.
- If you always save your documents on the network filesystem, you can access your documents from any department computer, and most university computers.
    - But describing all the combinations is confusing!
- The following information is from the perspective of student preference for one system or another.
- Your primary system could be the one you would prefer to use if you were allowed to choose: Windows, Mac, Linux.
    - For many, the primary system might be Mac, because CMPT 145 labs are held in the Mac lab.

Using MacOS as your primary system

- Your home folder is in `/student/machome/abc123`
    - Replace `abc123` with your NSID!
- You can move to another MacOS computer, and this home folder moves with you.
- You can move to a Windows computer. Your Mac documents are found in `M:\`.
- You can move to a Linux computer. Your Mac documents are found in `/student/machome/abc123`
- You can move to any Windows computer on campus. Your Mac documents are found in `\\csfiles.usask.ca\machome\abc123`

Using Linux as your primary system

- Your home folder is in `/student/abc123`
  - Replace `abc123` with your NSID!
- You can move to another Linux computer, and your home folder moves with you.
- You can move to a Windows computer. Your Linux documents are found in `H:\`.
- You can move to a MacOS computer. Your Linux documents are found in `/student/abc123`.
- You can move to any Windows computer on campus. Your Linux documents are found in `\\csfiles.usask.ca\abc123`

Using Windows as your primary system

- Your home folder is `V:\cmpt\cswin`
- You can move to another Windows computer, and your home folder moves with you.
- You can move to a MacOS computer. Your Windows documents are found in `smb://cabinet.usask.ca/work$/abc123/cmpt/cswin`
- You can move to a Linux computer. Your Windows documents are found in `smb://cabinet.usask.ca/work$/abc123/cmpt/cswin`.
- You can move to any Windows computer on campus. Your Windows documents are found in `\\cabinet.usask.ca\work$\abc123\cmpt\cswin`

Make the network work for you

- Choose any system to be your primary system.
- Store all your work (Python scripts, Word docs, data) in your primary system's home folder.
- Don't be afraid to move to other computers in the lab.
- Each system has a different home folder, but your primary system's home folder is accessible from any department computer.
- Sometimes, an application might present you with a default Save As... location that is not in your home folder. Be careful to check!
- Revisit these notes when you move to a different system!

Part II

Pre-Reading: Executing Python Scripts from the Command-Line

PyCharm vs Command-Line

- PyCharm is an IDE (integrated development environment) which manages editing and running Python programs all in one application. IDEs are great!
- PyCharm does not run Python programs directly.
- Python programs are (typically) executed by a separate application called a Python interpreter, which runs your Python program.
- The Python interpreter used by PyCharm can also be used directly from the command-line. Command-lines are great too!
- Since we already know how to use Python in PyCharm, we will add to our toolbox, and learn to use Python on the command-line.

Executing a Python script on the command-line

- On a UNIX system (Mac, Linux) Python is an application you can run on the command-line.
- To run Python in interactive mode:
  - Type `python3.6` on the command-line.
- To execute a Python script saved in a document:
  1. Use `cd` to "move" to a folder containing the Python script.
  2. Type `python3.6 docname.py`

  (replace `docname.py` with the name of any document containing a Python script)

Notes on Executing Python programs

- If your Python script has any errors, they will be displayed in the command-line window.
- If your Python script `docname.py` is not in your working directory when you run `python3.6 docname.py`, you'll get an error.
- There are at least 2 versions of Python on lab machines. `python3.6` is the one that PyCharm should be set up to use.
- If you forget the `3` you may get a different, older version of Python, and your program may not run, or may not run properly.

Using PyCharm's Terminal

- At the bottom of the PyCharm window is a button labelled Terminal.
- Clicking on this button starts a command-line from within PyCharm.
- If PyCharm is running on Linux or Mac, the Terminal window is UNIX.
    - All of the UNIX commands are available here.
    - This command-line should be exactly the same as if you opened a Terminal outside of PyCharm.
    - The terminal sets your context (the current working directory) to the folder for your current PyCharm project.

Part III

Pre-Reading: Using Command Line Arguments in Python

A simple Python program

```python
# count.py

example = 100

def sum_to(x):
    # TODO: write the function interface!!
    total = 0
    for i in range(x+1):
        total += i
    return total

print(sum_to(example))
```

You can find this program in the Laboratory folder.

Running `count.py` in the console

```
UNIX[1]% pwd
/Users/horsch/CMPT145/Lab02
UNIX[2]% ls
count.py
UNIX[3]% python3.6 count.py
5050
```

The behaviour of `count.py` is static, because to change its behaviour, we have to use the editor.

Command-line arguments

- The command-line can run Python programs!
- Python’s console input and output is directed to the command-line.
- We’ll see how to send information to a Python program from the command-line.
- We call this kind of information “command-line arguments”; it’s similar to the way we send arguments to a function in Python.

The value of sending information to a program

Consider if we could tell `count.py` to use a different value for the variable `example`. The program would be much more useful.

```
UNIX[5]% python3.6 count.py 10
55
UNIX[6]% python3.6 count.py 20
210
UNIX[7]% python3.6 count.py 100
5050
```

Being able to send a program information through the command-line is what we mean by “command-line arguments”.

Getting information from the command-line

```python
# count.py
# version 2

import sys as sys

example = int(sys.argv[1])

def sum_to(x):
    # TODO: write the function interface!!
    total = 0
    for i in range(x+1):
        total += i
    return total

print(sum_to(example))
```

We use the module `sys`, and a list in that module called `argv`. Nothing else changed.

The list `sys.argv`

- When the command-line runs your Python program, it sends most of the command to the Python interpreter.
- Python initializes the `sys.argv` list and then runs your program.
- Your scripts can look at the `sys.argv` list, or ignore it.
- The first item in the `sys.argv` list (at index 0) is the name of your program. This is a UNIX tradition.
- The data in the `sys.argv` list are strings. You may need to convert the data, as in our example.
- Note: A script that uses command-line arguments should be run from the command-line, not PyCharm.

Command Line Arguments via Terminal

On the command line, arguments are passed to a Python script by listing them after the script filename:

- Arguments are separated by spaces on the command-line.
- To indicate a string argument that contains spaces (like a sentence), use quotation marks (e.g. `'Good job!'` or `"Hello, world"`).

```
python3.6 scriptname.py arg1 arg2 arg3 ...
```

Summary: Acquiring Arguments within Python

Extract command line arguments using the `sys` module:

- Arguments are stored in `sys.argv` as a list of strings.
- `sys.argv[0]` contains the name of the script.
- Any command line arguments are in the list starting at index 1.
- If no arguments were given, `sys.argv` has length exactly 1.

```python
import sys

prog_name = sys.argv[0]    # program name
args_list = sys.argv[1:]   # list of arguments
```

Part IV

Activities: Executing Python Scripts from the Command-Line

Executing Python Programs from PyCharm

ACTIVITY:

(a) Start up PyCharm.

(b) Create New Project for this lab, using the $\boxed{\cdots}$ button to the right of the Location.

(c) Create a new folder called `Lab2`, and create the new project there.

(d) Is the working directory inside your home directory on a network filesystem? If not, learn how to create a project in your home directory!

(e) Be sure that you're using the correct project interpreter, using the Create Project window.

Using the Terminal inside of PyCharm

ACTIVITY:

- Click the Terminal button in PyCharm to open a terminal frame.
- Use the UNIX command `pwd` to see your project's working directory.
- Is the working directory inside your home directory on a network filesystem? If not, learn how to create a project in your home directory!
- Practice some of your other UNIX commands from last week!

Write a Simple Python Program in PyCharm

- ACTIVITY: Add a new script "`hello.py`" to your "`Lab2`" project:

```
# your name here
# your student number here
# your section number here
# your lab/assignment info
# Print hello in English and in French

print("Hello!")
print("Bonjour!")
```

  - Replace "your" comments with your information
  - Provide one sentence or less description of what your program does
- It's a good habit to add these comments to all Python files you hand in

Executing Python Programs

ACTIVITY:

(a) Using PyCharm's terminal, type `python3.6 hello.py` to run the script,
(b) Using the Run button in PyCharm, run your script.
(c) Look up the control sequence for running scripts in PyCharm, and use it!

Part V

Activities: Using Command Line Arguments in Python

Activity 1

- Download the `count.py` program (Slide 19), and change it so that it behaves as in our example (Slide 23).
- Run the new version of the `count.py` program in your PyCharm Terminal. At least 3 times with 3 different integers!
- Copy/paste the output of your 3 different examples from the PyCharm Terminal to a file called `lab02-transcript.txt`.

Activity 2

- Run the new version of the `count.py` program, but without any command-line arguments.
- Observe the error that is reported!
- Add an if-statement to `count.py` so that it only prints the sum if exactly 1 command-line argument is given.
  Hint: Check the length of `sys.argv`!
- If your script detects a missing command-line argument, have it display a helpful message reminding the user to give an integer argument.
- Copy/paste the output of improved version from the PyCharm Terminal to a file called `lab02-transcript.txt`.

Activity 3

- Download the script `self-avoiding-random-walk.py` from the Laboratory.
- Add this script to your Lab2 project.
- Run `self-avoiding-random-walk.py` a few times in the PyCharm Terminal. Note that the output varies a little.
- Modify the script so that it uses command-line arguments to initialize the variables:
  - `n`: grid width and height
  - `trials`: number of times to repeat for an average

Activity 3 continued

- Run the revised version of `self-avoiding-random-walk.py` with different values for `n` and `trials`.
- Use the command-line to explore different values for `n` and `trials`. Find input values that consistently lead to an output of around 49-51 percent dead ends.
  Hint: Keep running the script using different values for `n` first, leaving `trials` small. When you're close to 50%, increase trials to get a more stable result.
- Copy/paste the output of your exploration of `n` and `trials` from the PyCharm Terminal to your `lab02-transcript.txt` file.

Part VI

Hand In

What To Hand In

Hand in your `lab02-transcript.txt` file showing:

- The results of running the new version of the `count.py` program least 3 times with 3 different integers.
- The results of running the improved version of the `count.py` program that checks for the correct number of command-line arguments.
- The results of your exploration of `n` and `trials` for `self-avoiding-random-walk.py` leading to a consistent output around 49-51 percent dead ends.