(PRACTICAL)
COMPUTATIONAL PHYSICS

Physics 551
Lecture 7

NOTATION

Extra Reading

Optional Exercise

Recommended

- This lecture slides for this course will attempt to use a uniform notation throughout. A normal paragraph looks like this.

☞ *Italicized paragraphs with pen bullets will indicate definitions, with the defined word or phrase shown in* ***SMALL-CAPS****.*

✏ Pencil bullets will indicate the introduction of **new notation**.

☞ Pointing hand bullets indicate important points that might otherwise be overlooked.

ANNOUNCEMENTS

ANNOUNCEMENTS

- To clone this week's **shell command demonstration materials** please invoke

```
$git clone https://github.com/hughdickinson/CompPhysL7Shell.git /
home/computationalphysics/Documents/theShellGym/lecture7
```

- To clone this week's **Python demonstration materials** please invoke

```
$git clone https://github.com/hughdickinson/CompPhysL7Python.git
/home/computationalphysics/Documents/python/lecture7
```

☞You can also find these commands on the Blackboard Learn website.

ANNOUNCEMENTS

- The following will be covered during the **shell demonstration** for this lecture.
- There is some new software to install **on VirtualBox**.
- For future reference, the **shell commands** that will be invoked are

```
$ sudo apt-get install libgsl0ldbl
$ sudo apt-get install libgsl0-dev
$ sudo apt-get install source-highlight
```

☞ **If** you had **difficulties** with the Git demonstration from **Lecture 5**, try invoking

```
$ git config --global push.default simple
```

ANNOUNCEMENTS

- The following will be covered during the **python demonstration** for this lecture.
- There are some Python modules for numerical and scientific computation to install **on VirtualBox**.
- For future reference, the **shell commands** that will be invoked are

```
$ sudo apt-get install python-numpy
$ sudo apt-get install python-scipy
$ sudo apt-get install python-astropy
$ sudo apt-get install python-matplotlib
```

CLARIFICATIONS

GITHUB INSTRUCTIONS REMINDER

☞The **remote URL** of **your *private* repository** is:

https://github.com/ISUComputationalPhysics/
<Firstname><Surname>Homework.git

☞The path to your **local working directory** should be:

/users/computationalphysics/Documents/homework

☞To **synchronize** your local working directory with the remote repository, navigate to your local working directory and invoke

```
$ git pull
```

☞To **update** the remote repository to reflect the changes that have been **committed** to your local working directory, invoke

```
$ git push
```

SHELL USEAGE

SHELL VARIABLES

- ***SHELL VARIABLES*** *are* ***identifiers*** *that can be associated with (almost) arbitrary string-like tokens and thereafter used as* ***aliases*** *in shell commands.*
- The following **shell command** associates a shell variable called `EXAMPLE_PATH` with the token "`./examples`" using the "`=`" operator

```
$ EXAMPLE_PATH=./examples
```

(Spaces are not permitted)

- To reference the value associated with an environment variable use the "`$`" token. The following **shell command** prints "`./examples`" to the terminal

```
$ echo $EXAMPLE_PATH
```

ENVIRONMENT VARIABLES

☞ ***ENVIRONMENT VARIABLES*** *are* ***shell variables*** *with* ***specific identifiers*** *that are used to influence the behavior of the shell or specify particular properties of the system.*

- A **common use** of environment variables is the specification of the filesystem **paths that the shell should search** when looking for particular types of file.

☞ The `PATH` environment variable specifies the list of paths that the shell should search for **standalone binary executables**.

☞ The `LD_LIBRARY_PATH` environment variable specifies the list of paths that the shell should search for **shared library files**.

ENVIRONMENT VARIABLES

- Environment variables are declared using the `export` shell command, followed by a shell variable ***declaration expression***.

```
$ export PATH=./example
```

☞ The token associated with any shell variable can include a ***reference*** to a preexisting shell variable.

☞ When associating a shell variable with a **list** of filesystem paths, the "`:`" token is used to separate individual paths in the list e.g.

```
$ export PATH=$PATH:./example:/binaries
```

DEMONSTRATION

Using **Shell variables** and **Environment variables**

Clone the Shell demonstration material from Github:

```
$git clone https://github.com/hughdickinson/CompPhysL7Shell.git
 /home/computationalphysics/Documents/theShellGym/lecture7
```

C++ LEGACY CODE: HEADER FILES AND LIBRARIES

C++ SHARED LIBRARIES: CREATION

- On **Linux** operating systems, the names of shared libraries typically incorporate a "*`lib`*" prefix and "*`.so`*" suffix.
- Shared libraries can be **created** from C++ source code by specifying the `-shared` and `-fPIC` flags when invoking `clang++`.

```
$ clang++ -std=c++11 -shared -fPIC -o
  libSharedLibrary.so sourceFiles…
```

☞ Source code used to create shared libraries should **not** contain a `main` function.

C++ SHARED LIBRARIES: USEAGE

- Recall that when `clang++` is invoked, libraries are associated with the executable during the **linking** stage of the build.

☞ *Linking is actually performed by a separate utility called the* ***LINKER***

- **By default**, the linker only searches particular **standard locations** to find the libraries you specify should be linked against.
- You can specify **additional search locations** by providing a `-L` flag when invoking `clang++`.

C++ SHARED LIBRARIES: USEAGE

- To specify that the linker should include *`libDir`* among the locations it searches for libraries when linking an executable, the following invocation is required

```
$ clang++ -std=c++11 -Llibdir -o output inputs…
```

- The *`inputs`* may now include one or more shared libraries.
- **If** the library name uses the conventional "*`lib`*" and "*`.so`*" prefix and suffix are used, then an **abbreviation** can be used.
- Specifically, the library *`libLibraryName.so`* can **also** be specified using the token *`-lLibraryName`*.

C++ SHARED LIBRARIES: USEAGE

☞ *When a **standalone binary** that has been **linked with shared libraries** at build time is invoked, a separate utility called the **DYNAMIC LINKER** is invoked to assemble the required binary code from the various shared libraries.*

- The `LD_LIBRARY_PATH` environment variable specifies the list of filesystem paths that the dynamic linker will consider when **searching for shared libraries**.

☞ If you create new shared libraries in the *`myLibraries`* directory, you should **update** the `LD_LIBRARY_PATH` environment variable using

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:myLibraries
```

DEMONSTRATION

Creating and **Linking Against** a **Shared Library**.

***If necessary*, clone the *Lecture 6* C++ demonstration material from Github:**

```
$ git clone https://github.com/hughdickinson/CompPhysL6CPP.git
  /home/computationalphysics/Documents/cPlusPlus/lecture6
```

PYTHON

PYTHON BASICS

- Python is a **modern** programming language that has a **very large scientific user community**,
- This community has provided a rich suite of of **preexisting functionality** that you can (and should, with citation!) use in your programs.
- Python is an **object-oriented** programming language that uses **weak variable typing**.
- Python implements a **simple syntax** that promotes **rapid development** and **prototyping**.

For more information about the Python Language, see the Python Homepage and the Online Python Documentation.

PYTHON BASICS

- Python is primarily an **interpreted language**.

☞ *In **INTERPRETED LANGUAGES**, each instruction in the program's source code is parsed and executed **directly** by a utility called an **INTERPRETER**.*

☞ Several Python interpreters exist with differing levels of utility.

- The **most basic** Python interpreter runs within the terminal window and can be invoked using

```
$ python
```

☞ To **exit** the Python interpreter, hold the ***Ctrl*** key and press "D".

IPYTHON BASICS

☞ ***IPython*** is a **more sophisticated** interpreter that **may** also be invoked from the shell command line using

```
$ ipython
```

- When used **within the terminal**, `ipython` provides several improvements over the basic `python` interpreter, including ***limited colorization***, more ***user friendly error diagnostics***, and an ***enhanced inline help system***.
- However, the primary strength of ***IPython*** is its **web-browser based interface** - the ***IPython Notebook***.

IPYTHON NOTEBOOK

☞The ***IPython Notebook*** interface can be launched from the shell command line using

```
$ ipython notebook
```

- **By default** on your VirtualBox Ubuntu Linux installation, this invocation will launch the ***Firefox*** web browser and start an ***IPython Notebook*** session.
- The ***IPython Notebook*** interface facilitates, interactive editing, annotation and invocation of Python source code, and enables straightforward **code-sharing** using a **portable file format**.

IPYTHON NOTEBOOK

- If you work in using an ***IPython Notebook***, your work will be periodically **autosaved**.
- The ***IPython Notebook*** provides **syntax highlighting** functionality, which makes your code **easier to read and develop**.
- The ***IPython Notebook*** allows you to **separate** your code into separate ***cells*** and provides facilities for media-rich, formatted **annotation** of each code-containing cell.
- This encourages **modular** code development and helps make your code easily **comprehensible** when **shared** with collaborators.

DEMONSTRATION

Introducing **Python** and the **IPython Notebook**

Clone the Shell demonstration material from Github:

```
$git clone https://github.com/hughdickinson/CompPhysL7Python.git
 /home/computationalphysics/Documents/python/lecture7
```

LECTURE 7 SUMMARY

- After reviewing the material from this lecture (including the demonstration material) **and completing the reading exercises** you should know:

1. How to **work effectively** with your **private homework repository** on GitHub.
2. How to ***declare***, ***initialize***, ***reset*** and ***combine*** **shell variables** in the Ubuntu Linux terminal.
3. How to **export** shell variables in the Ubuntu Linux terminal.

LECTURE 7 SUMMARY

4. That shared libraries contain **compiled binary code** that can be **loaded** and **invoked at runtime** by a binary executable.
5. That **shared library names** on **Linux** systems typically begin with "*`lib`*" and have a "*`.so`*" suffix.
6. That source code that is intended to be compiled into a shared library should **not** define a `main()` function.
7. How to **create shared libraries** from C++ source code by invoking `clang++` and supplying the `-fPIC` and `-shared` flags.

LECTURE 7 SUMMARY

8. How to **link** a binary executable with **shared libraries** by supplying them as input files to an invocation of `clang++`.
9. That shared library **names** supplied to clang++ can be **abbreviated** if they begin with "*`lib`*" that have a "*`.so`*" suffix e.g. *`libExample.so`* abbreviates to *`-lExample`*.
10. How to **invoke** and **use** the **basic Python interpreter**, the terminal-based **IPython interpreter** and the web-browser-based **IPython Notebook interface**.

LECTURE 7 SUMMARY

I 1. That Python is a ***modern***, ***interpreted***, ***object-oriented*** programming language that uses ***weak variable typing***.

I 2. The **basic properties** of Python's four built-in numeric types - **`int`**, **`long`**, **`float`** and **`complex`**.

I 3. The **basic properties** and functionality of four of Python's built-in sequence types - **`list`**, **`tuple`**, **`str`** and **`unicode`**.

I 4. The basic properties of Python's built-in **`set`** type.

LECTURE 7 SUMMARY

15. The basic properties of Python's built-in **`map`** type.
16. How to specify **conditional branching statements** in Python using **`if`**, **`elif`** and **`else`** clauses as well as the **ternary branching construct**.
17. How **control the flow** of a Python program using **`for`**-loops and **`while`**-loops.
18. How to **define** and **call functions** within in your Python programs.

OPTIONAL READING

A **markup language** called ***Markdown*** is used to **annotate your code** in **IPython Notebook** documents is developed by Daring Fireball. You can find more information about the **features** and **syntax** of ***Markdown*** from the following websites

The Daring Fireball Homepage
daringfireball.net/projects/markdown
The Wikipedia Page about Markdown
en.wikipedia.org/wiki/Markdown
A syntax overview on StackOverflow
stackoverflow.com/editing-help

LECTURE 7 HOMEWORK

Be sure to thoroughly review the C++ demonstration material from Lecture **6**!

Read Sections:

5.4. Numeric Types
5.6. Sequence Types (str, unicode, list, tuple)
5.7. Set Types (set)
5.8. Mapping Types

From the **Python Standard Library Reference**
https://docs.python.org/2.7/library/index.html

- Complete the **Lecture 7 Homework Quiz** that you will find on the course Blackboard Learn website.