(PRACTICAL)
COMPUTATIONAL PHYSICS

Physics 551
Lecture 9

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 **Python demonstration materials** please invoke

```
$git clone https://github.com/hughdickinson/CompPhysL9Python.git
 /home/computationalphysics/Documents/python/lecture9
```

- You may have **already** cloned this material. **In that case**, navigate to to the cloned working directory and synchronize with GitHub.

```
$cd /home/computationalphysics/Documents/python/lecture9
$git pull
```

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

PYTHON

WORKING WITH MATPLOTLIB

THE PYPLOT MODULE

- The `matplotlib` package includes a module called `pyplot` which provides a ***MATLAB***-like interface to the functionality of ***MatplotLib***.
- The `pyplot` module maintains an **internal state** that models **abstract concepts** such as the **current figure** and the **current axes**.
- These concepts define an **implicit context** in which invocations of the `pyplot`-provided **drawing** and **plotting** functions are interpreted.

THE PYPLOT MODULE

- Accordingly, the **target** plotting "canvas" for many function invocations does not need to be **explicitly** specified.
- For example, invocation of the `pyplot.plot(…)` function **implicitly** draws on the **current axes** in the **current figure**.
- The `pyplot` interface defines functions that explicitly or implicitly modify the invocation context for drawing and plotting functions.
- For example, invocation of the `pyplot.subplot(…)` function **may** implicitly change the **current axes**.

USEFUL PYPLOT FUNCTIONS

- The demonstration material for this lecture includes example usages of several **`pyplot`**-provided functions.
  - The **`subplot(...)`** function facilitates unification of **multiple sets of plot axes** within the same figure.
  - The **`legend(...)`** function adds a **legend** to disambiguate multiple curves that are drawn on the **same axes**.
  - The **`xlim(...)`**, **`ylim(...)`** and **`axis(...)`** functions facilitate modification of the **ranges** spanned by the **current axes**.

DEMONSTRATION

Working with **Matplotlib**

***If necessary*, clone the *Lecture 9* Python demonstration material from GitHub:**

```
$ git clone https://github.com/hughdickinson/CompPhysL9Python.git
  /home/computationalphysics/Documents/python/lecture9
```

OPERATING ON FILES USING PYTHON

OPENING FILES

☞Files are abstractly represented in ***Python*** as instances of the `file` class.

- ***Python*** provides the built-in **open(...)** function that attempts to open a specified file and returns a corresponding `file` instance if successful.
- Files can be opened using various **access modes** including ***read-only***, ***write-only*** and ***read-write***.
- The **target file** and the **desired access mode** are specified using the **arguments** of the **open(...)** function.

WRITING TO FILES

- The `file` class provides several methods that can be used **insert new data**.
- To use these methods the file **must** have been opened in a **writeable state**.
- The `file.write(...)` method accepts a `str`-type argument specifying the data to be inserted.
- The `file.writelines(...)` method accepts `list` containing `str`-type elements and will insert all the data they contain.

READING FROM FILES

- The `file` class provides several methods that can be used **extract file data**.To use these methods the file **must** have been opened in a **readable state**.
- The `file.read(...)` method accepts an `int`-type argument specifying the number of **bytes** to be extracted from the file. The extracted data are returned as a `str` instance.
- The `file.readlines()` returns a `list` containing `str`-type elements corresponding to the **individual lines** of a textual file.

DEMONSTRATION

Reading from and **Writing** to **Files** using Python

***If necessary*, clone the *Lecture 9* Python demonstration material from GitHub:**

```
$git clone https://github.com/hughdickinson/CompPhysL9Python.git
 /home/computationalphysics/Documents/python/lecture9
```

OBJECT ORIENTATION IN PYTHON

THE PYTHON OBJECT MODEL

- The ***Python*** language enables the definition of **classes**.

☞ ***Python*** *classes define methods and member data, collectively referred to as* ***ATTRIBUTES***.

☞ Attributes cannot be declared as private in ***Python***. **All** attributes of ***Python*** classes are **publicly accessible**.

- ***Python*** implements class **inheritance** and classes may be defined that **derive** from others.
- Derived ***Python*** classes may **override** the methods of their parents.

DEMONSTRATION

Implementing **Classes** in Python

***If necessary*, clone the *Lecture 9* Python demonstration material from GitHub:**

```
$ git clone https://github.com/hughdickinson/CompPhysL9Python.git
  /home/computationalphysics/Documents/python/lecture9
```

LECTURE 10 SUMMARY

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

1. That the `matplotlib` Python package includes the `pyplot` module, which provides a MATLAB-like plotting interface.
2. That the `pyplot` module maintains internal state that defines the ***current figure*** and ***current axes***.
3. How to modify the ranges spanned by the current axes using the `pyplot xlim()`, `ylim()` and `axis()` functions.

LECTURE 10 SUMMARY

4. How the `pyplot.subplot()` function is used to **set the current axes** for drawing and generate figures comprising **multiple independent sets of axes**.
5. How to draw a **legend** on the current axes using the `pyplot.legend()` function.
6. Various methods for **reading** from and **writing** to **files** using ***Python***.
7. How to **define** simple **classes** using ***Python***.
8. How to **define** class **methods** using ***Python***.

LECTURE 10 SUMMARY

9. That the **first argument** of all class methods **must** be interpreted as a reference to the invoking class instance and is conventionally assigned the identifier `self`.
10. How to provide a class constructor in Python by defining an `__init__` method.
11. How to specify class **inheritance** relationships using ***Python***.
12. How to **override** base class methods in derived classes using ***Python***.

LECTURE 10 SUMMARY

13. That the `__init__` method of a derived class **must** call the `__init__` methods of its base classes, and **must explicitly** pass the `self` argument when it does so.
14. How to **instantiate** classes using ***Python***.
15. How to comment your Python source code to enable automatic integration with the built-in `help()` function.
16. How to automatically generate **HTML documentation** using the `pydoc` utility.

OPTIONAL READING

The Python Online Tutorial
docs.python.org/2/tutorial/

The Matplotlib "Beginner's Guide"
matplotlib.org/users/beginner.html

The doxygen "Getting Started" guide:
www.stack.nl/~dimitri/doxygen/manual/starting.html

The NumPy Online Tutorial
wiki.scipy.org/Tentative_NumPy_Tutorial

LECTURE 10 HOMEWORK

Review the **Python** demonstration material from Lecture **9**!

- Read **Section 9: Classes** from the **Python Online Tutorial**
  docs.python.org/2/tutorial/
- Read **all** subsections of **Section 25.1: pydoc** from the **The Python Standard Library** online reference
  docs.python.org/2/library/index.html
- **If necessary**, reread the ***Pyplot Tutorial*** section from the **Matplotlib "Beginner's Guide"**
  matplotlib.org/users/beginner.html

- Complete the **Lecture 10 Homework Quiz** that you will find on the course Blackboard Learn website.