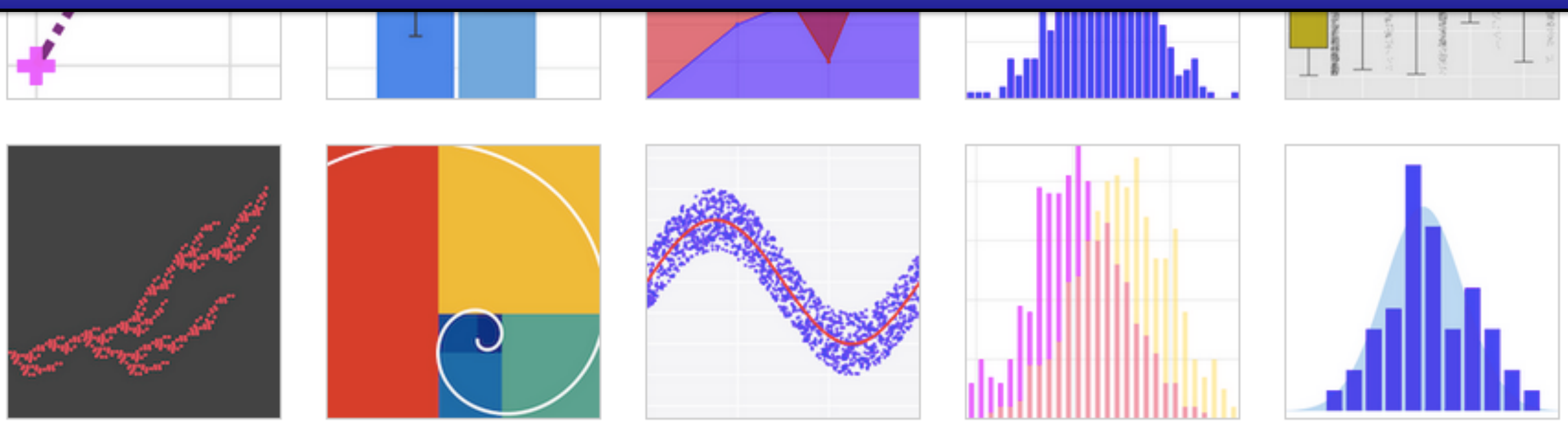
Programming for Scientific Applications

based on slides from Gordon Erlebacher and Xiaquiang Wang



Syllabus

- Class participation (questions, suggestions, comments): might increase your grade if otherwise too low
- Homeworks (1per week): 30%
- Labs: 30%
- Midterm: 20%
- Final: 20%

Class Attendance

- Attendance Mandatory
- Keep your cell phone in your pocket or purse

Class Objectives

- to understand the benefits of interpreted and compiled languages and know when to use each one to best advantage
- to understand Python sufficiently to program applications with confidence
- to understand C++ sufficiently to program applications
- learn to interface C++ and Python to each other, to take advantage of the best features of both languages
- through lab work, develop the skills to apply Python and C++ to a range of practical scientific applications, ranging from graphical user interfaces, web-based display of results, processing of scientific data, and visualization

Honor Code

- The Academic Honor System of The Florida State University is based on the premise that each student has the responsibility 1) to uphold the highest standards of academic integrity in the student's own work, 2) to refuse to tolerate violations of academic integrity in the University community, and 3) to foster a high sense of integrity and social responsibility on the part of the University community. Please note that violations of this Academic Honor System will not be tolerated in this class. Specifically, incidents of plagiarism of any type or referring to any unauthorized material during examinations will be rigorously pursued by this instructor. Before submitting any work for this class, please read the ``Academic Honor System" in its entirety (as found in the *FSU General Bulletin* and in the *FSU Student Handbook* and ask the instructor to clarify any of its expectations that you do no understand.

Outline

- Class Work (T-H)
  - fundamentals of Python and C++
  - example-based
  - use of web to supplement class material
- Lab work (F)
  - illustration of Python and C++ in practical situations
  - 2.5 hours per week
  - write a report about lab (summary, pictures, tables)
  - each lab: 1-2 weeks. Reports due weekly

Labs (Python / C++)
(may change and not in order of presentation)

- Introduction to Python (Python)
- Python Modules(Python)
- A small number game(Python)
- Monitoring and displaying stocks [or other online quantities] (Python)
- Plotting scientific data (Python)
- Use of the Python Imaging Library (Python)
- Application of FFTs: sound (Python)
- Solving ODEs (C++)
- Manipulating video (C++)
- Wrapping an existing scientific code in C++ and GUI construction (Python/C++)

Class Contents: Python

- Overview (today)
- Modules, Operating system
- lists, dictionaries, sequences
- functions
- text processing
- files, I/O
- numpy for arrays
- classes
- numpy for statistics
- graphical user interfaces
- C/C++ and wrapping

Class contents: C++

1. documenting source code (Python and C++)
2. Hello world (#include, main, printf)
3. Functions (value, reference, pointers, overloading)
4. input/output (streams)
5. command line input (< and >, stdin, stdout)
6. class (attributes, functions, public, private, protected)
7. class (subclass, polymorphism)
8. Libraries (static and shared) + string, ar, nm
9. Standard Template Library
10. arrays (1d, 2d, 3d)
11. my array class (example)
12. Polymorphism

What is programming?

- What is a computer program?
  - "A set of coded **instructions** that enables a machine, especially a computer, to perform a desired **sequence** of operations." – American Heritage Dictionary

- Programming instructions are written using a "programming language"
  - Examples: C/C++, Java, Assembly, Fortran, Cobol, BASIC
  - LOTS of programming languages, different uses for different languages

C++

Back to index

```
// Hello World in C++ (pre-ISO)

#include <iostream.h>

main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Haskell

Back to index

```
-- Hello World in Haskell

main = putStrLn "Hello World"
```

Lisp

Back to index

```
;;; Hello World in Common Lisp
(defun helloworld ()
  (print "Hello World!")
)
```

Assembler-Linux

Back to index

```
    ;; Hello World for the nasm Assembler (Linux)

    SECTION .data

    msg  db   "Hello, world!",0xa ;
    len  equ     $ - msg

    SECTION .text
    global main

main:
        mov     eax,4       ; write system call
        mov     ebx,1           ; file (stdou)
        mov     ecx,msg         ; string
        mov     edx,len         ; strlen
    int     0x80        ; call kernel

    mov  eax,1          ; exit system call
        mov     ebx,0
        int     0x80        ; call kernel
```

Terminology

- **Computer program** – a set of instructions that tell a computer exactly what to do
  - The instructions might tell the computer to add up a set of numbers, or compare two numbers and make a decision based on the result, or whatever.
- **Programming language** – a language used by humans to program computers
  - e.g., Fortran, Cobol, Basic, Pascal, C, C++, Java, Perl
- **Compiler** – translates a computer program written in a human-readable computer language (like C++) into a form that a computer can execute
  - You have probably seen .exe files or .app ‘files’ on your computer.
  - These executable files are the output of compilers.
  - They contain executables -- machine-readable programs translated from human-readable programs.

Programming

- *Problem solving*
  - *Logical/methodical way of solving a problem*
- *Algorithm/abstraction*
  - *An algorithm is a series of step-by-step instructions that produces a solution to a problem*
- *Step wise refinement*
  - *Incrementally adding functionality to a program*

Five steps to writing a program

- Define the problem
- Plan the solution
  - pseudocode
- Code the program
  - Using a programming language
- Test and debug
  - Using a compiler
- Document

Python vs C++

- Python is interpreted
- C++ is compiled

Python vs C++ and other languages

- Python is interpreted (and Julia)
- C++ is compiled (and D)

Hello World

- In a file named **HelloWorld.py**
  - print ‘Hello World’

Scripting vs traditional

- Traditional programming
  - C, C++, Fortran, Java (kind of), C#
  - mostly computation, networking, other low-level activities
  - traditional code if often wrapped with scripting code for ease of use and integration into scripting framework
- Scripting framework
  - Perl, Python, Ruby, Scheme, Julia, Tcl, ...
  - Integrates text processing, I/O, report writing, and computation
  - Often sections of programs written in scripting languages are translated to traditional code for efficiency

Why Scripting?

- Easier to use
- Faster development time
- Avoid compilation and linking
- Integration of visualization, networking, datea analysis, etc.
- Python makes it simple to *glue together* different applications (plenty of tools, modules)
- Scientific computing is more than number crunching. In addition:
  - data manipulation, data analysis, visualization, format conversion, parametric studies, cataloguing, database access, etc.
  - these tasks are much much easier in Python than C++, Fortran, etc.
- Graphical user interfaces (use of Tk to wrap existing programs)

Why Scripting (cont.)

- Creation of demos to illustrate one's work. Use a single GUI to
  - read the data, run a simulation code, visualize and analyze the results
- Modern interfaces to old codes
  - Fortran codes from 1960's still run
  - horrible to read
  - front-end GUIs make the codes useful again
  - Instead of code-rewrite into C++, wrap with Python
- Python can provide unix-interface on Windows
  - also write code that runs on all operating systems

Language Classification

- Scripting languages: dynamically typed
  - variables types are not declared
  - syntax closer to natural language
- Traditional programming languages: type safe
  - declare variable types
  - syntax closer to the hardware

Gluing existing applications

- Visualization, application, data analysis can be glued into a single *larger* application
- Needs a graphical user interface (GUI)
- Method 1
  - applications communicate through files
- Method 2
  - python calls functions from the different applications, and moves data around through the use of pointers (a pointer is the memory address of the data)

Reading numbers from a file with Python

```
F=open('filename', 'r')
n = F.read().split()
```

Extract real and imaginary part of the string '(3.45,-52)'

```
In [2]: m = "(3.45, -52)".strip("()")
        re, im = [float(x) for x in m.split(",")]
```

```
In [4]: print m
        print re, im
```

```
3.45, -52
3.45 -52.0
```

Much more difficult in C++

Possible with a script, much harder with a compiled language

```
str(object)     # convert object to string
eval("strg")    # convert string to object
```

Efficiency

- Scripting languages
  - first compiled to byte code (independent of OS)
  - byte code is interpreted, line by line
  - better error messages
  - in general, codes run slower (not important for short codes)
  - sometimes speed is optimal (e.g., regular expressions)
  - inefficient sections of code can be rewritten in C/C++/Fortran/Java
- Traditional (compiled) languages
  - source code is translated to machine code (closer to the hardware)
  - machine code is hardware-dependent
  - in general, codes runs MUCH faster

Variable Declaration

- C, C++: type-safe languages
  - protect the user against himself
  - less bugs, safer programming
  - code reuse is harder (types are set in stone)
  - somewhat relaxed with classes and templates (for C++, but not for C)
- Python, Lua, etc.
  - when a variable is needed, assign a value
  - type is determined by the value
  - type conversion is sometimes automatic (Perl), sometimes not (Python)
  - same piece of code can be used in many different contexts

Dumping a datastructure: Python

```
def debug(leading_text, variable):
  if os.environ.get('MYDEBUG', '0') == '1':
    print leading_text, variable
```

Dumping a datastructure: C++

```
template <class T>
void debug(std::ostream& o,
        const std::string& leading_text,
        const T& variable)
{
        char* c = getenv("MYDEBUG");
        bool defined = false;
        if (c != NULL) { // if YDEBUG is defined ...

                if (std::string(c) == "1") {   // IF MYDEBUG
                        defined = true;
                 }
        }
        if (defined) {
                o << leading_text << " " << variable <<
std::endl;
        }
}
```

This is not possible with Java, Fortran, C: a different function must be written for each type.

For large projects, the flexibility of Python might lead to dangers, and static typing (as opposed to dynamic) might be important, at the cost of longer code development.

Flexible Function Interfaces

- def function(a,b,c,*kw, **dict)
  - a,b,c are variables (like all languages)
  - kw is an *arbitrary* list of variables
  - dict is a dictionary (hash table)
- Calling functions
  - Multiple return values allowed
  - function(c=3.4, b=12,[5,6,'g'],{title:'going home'})
  - types can be anything: checked at runtime
  - can use exception to handle errors (not possible in C and older versions of Fortran (earlier than Fortran 90)

Interactive Computing

- Can use python interactively (not possible with Fortran, Java, C, C++)
- Type python at the command line
- Alternatives: iPython, ipythonx (try for example: ipython notebook)
- Python can be used as a sophisticated calculator
  - try some examples in class

```
In [24]: import scipy

In [25]: def myfunc(x):
             return scipy.sin(x)

In [26]: integrate.quad(myfunc,0,10)

Out[26]: (1.8390715290764525, 9.9211633091092 9e-12)
```

Code Creation at Runtime

- from math import sin
  val = 'sin(1.2*x) + x**a' # a string
  x = 2.2; a = 3
  print "val= ", eval(val)
- Store functions into input files, read the input file, and execute the function

Heterogeneous Data Structures

- In C, C++, etc., arrays are of a single type:
  - float* a = new float [20];
  - std::string* str = new string [20];
  - can be more general using subclassing and polymorphism
- In Python, a list can be of any type:
  - functions, class instance, string, float, list
  - a = [‘temperature’, 3.45, [5,‘gone’], -35]

GUI (graphical user interfaces) Programming

- GUI interfaces in C/C++ are quite hard to write and error-prone
- Libraries for GUI exist in C/C++, but code writing is still harder than with libraries for Python, Ruby, etc.
- Example on the next two slides

GUI: Python

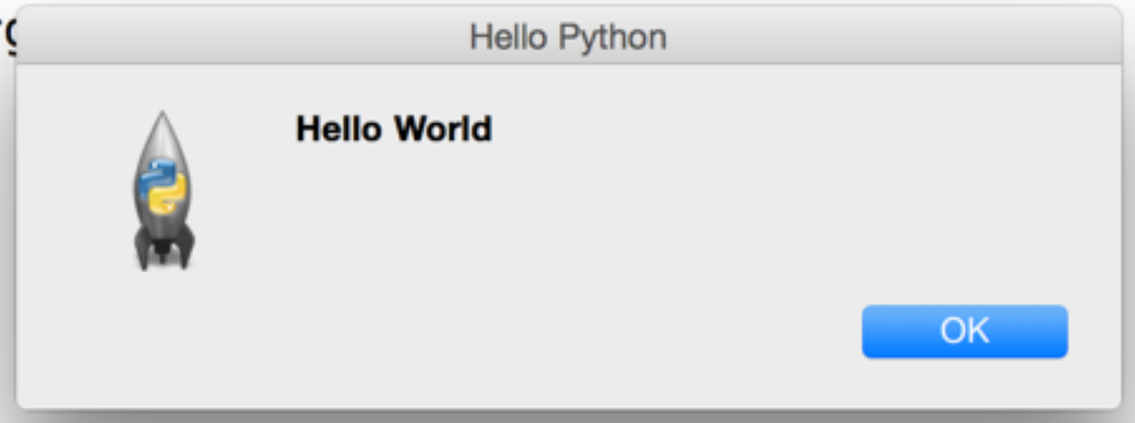
```
import Tkinter
import tkMessageBox
top = Tkinter.Tk()
def helloCallBack():
        tkMessageBox.showinfo( "Hello Python", "Hello World")

B = Tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
top.mainloop()
```

Equivalent Java code: 7 lines
Microsoft Foundation Classes: 25 lines

There are frameworks to create GUIs for C++, but they are often not portable across operating systems

Many scripting languages can be used to create GUIs that are viewable from browsers (PHP, Javascript, Ruby, Python, etc.)

```
nagal:~>python test.py
2015-01-07 23:27:18.678 Python[23512:1887018] ApplePersistenceIgnoreState: Exist
ing state will not be touched. New state will be written to /var/folders/j5/5fml
jc597bs2dshlnskxrz480000gn/T/org
```



Mixed Languages

- Each language is efficient for some tasks
- Dynamically typed languages are often implemented in C
  - as a result, extending these languages with functions written in C is relatively straightforward and well-documented (SWIG, F2PY)
- C, C++, Fortran were originally written to handle large applications in a single code. Thus calling functions written in other languages can be difficult
- The computational scientists seeks efficiency and the best tool for any job, for example,
  - python to call visualization tools
  - Fortran for fluid simulation
  - C++ to handle complex grids
- It is therefore useful to be able to write a code that can access each of the above components

Efficiency

Pure Python

```
# x is a list
for i in
range(len(x)):
    x[i] = sin(x[i])
```

20x slower than Fortran

Numpy module

```
# x is a numpy array
x = sin(x)
```

13x slower than Fortran

When to use a Statically Typed Language (C, C++, Fortran)

- Answer yes to one of the following:
  - do you need complex algorithms and data structures where low level control of implement is important?
  - does app manipulate large datasets so that low level control of memory handling is important?
  - Are the application's functions well-defined and changing slowly?
  - Will static typing be an advantage, e.g., for large development teams?

When to use a Dynamically Typed Language (Python, Ruby, etc.)

- Answer yes to one of the following:
  - main task is to connect existing components?
  - application includes GUI
  - app performs extensive text manipulation
  - the design of the app is expected to change significantly
  - CPU-time intensive parts of the application are concentrated in a tiny fraction of the code (so can be recoded in C/C++/ Fortran)
  - The app can be shortened if it is based heavily on lists and dictionaries with automatic memory administration
  - The app needs to communication with web servers
  - The app must be portable on Unix, Windows and Mac, with a GUI

Bottom line

- Most applications are a combination of type-safe and dynamically typed languages

Script or Program?

- Programs usually solve large, complex problems: fluid simulation, building simulation, biological problems, etc.
- Scripts can also solve large, complex problems, but usually, they are used to manipulate the results of these program, glue things together, and perform a variety of administrative tasks

Hello World
Python

- file hello.py
- print “hello world”

- # execute program
- python hello.py

Hello World Python (2)

- file hello_2.py
- #!/usr/bin/env python
  print “hello world”
- chmod +x hello_2.py
- # execute program
- ./hello_2

Hello World (C++)

- Source code in hello.cpp
- #include <stdio.h>
- int main()
- {
- printf("hello world\n");
- return 0;
- }

- Compile and execute
- g++ -o hello hello.cpp
- ./hello

Python Documentation

- from within interactive python shell:
  - help(command)
- From the unix/mac shell:
  - pydoc command
- Books and tutorials
  - http://docs.python.org/
  - http://www.alan-g.me.uk/tutor/index.htm
  - http://www.awaretek.com/tutorials.html

Class Contents: Python

- Overview (today)
- Modules, Operating system
- lists, dictionaries, sequences
- functions
- text processing
- files, I/O
- numpy for arrays
- classes
- numpy for statistics
- graphical user interfaces
- fortran/C++ and wrapping

Class contents: C++

1. documenting source code (Python and C++)
2. Hello world (#include, main, printf)
3. Functions (value, reference, pointers, overloading)
4. input/output (streams)
5. command line input (< and >, stdin, stdout)
6. class (attributes, functions, public, private, protected)
7. class (subclass, polymorphism)
8. Libraries (static and shared) + string, ar, nm
9. Standard Template Library
10. arrays (1d, 2d, 3d)
11. my array class (example)
12. Polymorphism

Labs (Python / C++)
(may change and not in order of presentation)

- Introduction to Python (Python)
- Python Modules(Python)
- A small number game(Python)
- Monitoring and displaying stocks (Python)
- Plotting scientific data (Python)
- Use of the Python Imaging Library (Python)
- Application of FFTs: sound (Python)
- Solving ODEs (C++)
- Manipulating video (C++)
- Wrapping an existing scientific code in C++ and GUI construction (Python/C++)

Tools

- iPython
  - simpler and more powerful command line framework
- Editor: vi, emacs, pico, gedit, etc.
  - must be simple text (MS-Word not appropriate)