PHYSICS 2G03
Scientific Computing

- Instructor: James Wadsley (ABB 352)
  wadsley@mcmaster.ca
- TAs :
  - James Lambert
  - Anton Borissov
  - Hector Robinson
  - Josef Rucska
  - Rishita Gudapati
- Contact:
  - Teams PHYSICS 2G03: Channel: **Class Chat**
  - Personal/Health issues: Instructor: wadsley@mcmaster.ca
  - Private Work Issues (e.g. HW, accounts): phys2g03.mcmaster@gmail.com
  - System admin (Anton): borissoa@mcmaster.ca

Course Info: Avenue to learn
http://avenue.mcmaster.ca
Complete lectures (video + PDF), all handouts, all HW will be on avenue
All quizzes and assessments (e.g. hand-ins) on avenue

Scientific Computing: Reference Material

The lecture notes are fairly complete but you may want an introductory reference for C/C++.

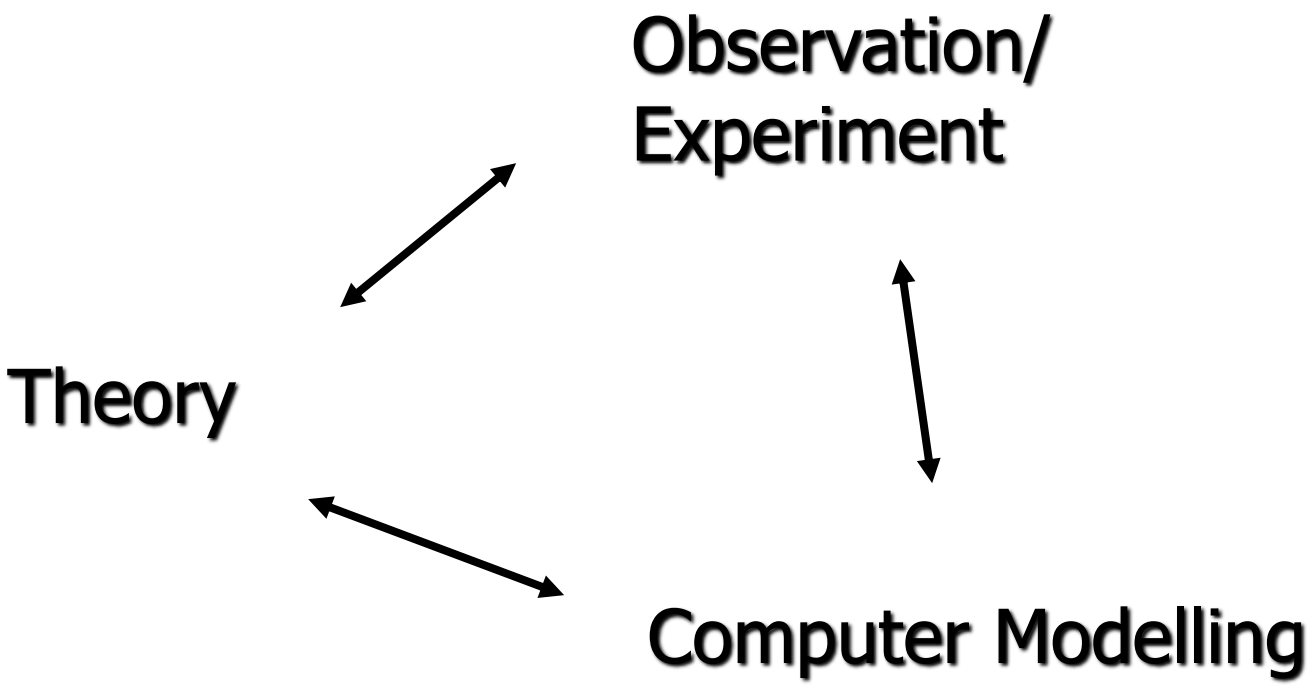
- A good introductory book, especially for those with limited programming background, (but expensive for new copies) is: *"C++ for Engineers and Scientists" Gary J. Bronson (This book mostly targets traditional, procedural programming as 2G03 does)*
- If you are quite interested in programming and want a comprehensive C++ text (also not too expensive), we recommend: *"The C++ Primer" Lippman, Lajoie, Moo (This covers modern C++ features and object oriented programming. It is a large book to carry around though)*
- A good book on just C (which covers basic C++ only) is: *"Programming in C" Stephen G. Kochan*
- You'll probably need a reference of some kind. Googling is an option. There are also many on-line references, such as https://en.cppreference.com but you may find it a bit technical.

Scientific Computing: Why PHYS 2G03?

Motivation

- Unix and Programming are essential skills for research: Experiment, Theory or (e.g. Astronomical) Observations
- As scientists we want to know how computers work, not just use them. This helps us know how to be efficient and to understand unexpected results
- Intro Computer Science courses in programming do not focus in on the key skills important for scientific (as opposed to commercial) computing

Scientific Computing: Third Way for Research



Careers can be built on Scientific Computing

e.g. Research, Data science, Aerospace, Design, Animation

Scientific Computation: Related Courses

- Programming/Computer Science
- Physics (Theory Computing / Astro)
- Mathematics/Numerical Analysis

  (Python/Matlab)
- Computational Physics

  Physics 4G03

Scientific Computing: Assessment

- Assigned work/homework assignments

Total: 50%

- Individual Programming Project

Total: 30%

- Regular quizzes

Total: 20%

Guidance on quizzes will be available

Project

- Each student will design and write a program that demonstrates the numerical solution of a research problem
- You are strongly encouraged to talk with other lecturers, grad students or research supervisors to identify a project of interest to you
- Discuss your ideas with the instructor + TAs!
  (we want to keep the projects do-able)

Project Timeline:

- Written proposal: Due Oct

  There is no midterm, instead you will hand in a written proposal outlining your project idea

  including relevant equations and basic discussion of related algorithms
- The TA and instructor will work with you to generate a suitable programming project from your idea
- Projects Due: Dec
- NB: You will be required to demonstrate your project code in class time

Course Structure

- There will be a strong focus on fundamental programming skills
- The classes Mon 12:30, Tue 1:30, Thur 12:30 on Microsoft Teams (via Mac Office 365/Teams)
- For 2020, most classes are tutorials or help sessions with some activities planned – you will need your laptop for every class
- Mon 10:30 classes will always be a simple help session (e.g. via Team chat)
- The TAs and instructor will be available outside class time for brief assistance.
- Office Hours: TBA – we can run additional office hours. Some input on good times would be useful.

Online Classes for 2020

- 2G03 has previously been taught in a lab with TAs present because **trying it yourself is critical** to the course
- We will be working via Microsoft Teams with multiple channels. The idea is work through exercises in synchronous class time on your own laptop.
- Start in “class chat”. Use individual TA channels to discuss things in small groups with each TA
- Use screen sharing to show the TA the issue and get help.
- TAs can also log into phys-ugrad and help with issues directly as well.

Course Outline

- Using the Labs, Unix, Algorithms, C/C++ Code, Hardware, Compiling, Debugging

  (handouts) (First 3-4 weeks)
- Variables, Programs, Functions, Program Flow, Arrays, I/O, Files

  (Next 4-5 weeks)
- Advanced Topics: Program Development, Object Oriented Programming, C vs C++, Fortran + other languages, Shell Scripts, Graphics, Parallel Programming

  (handouts) (Final 4 weeks)

Course Outline

- Available on avenue to learn
- The Homework handouts will include information about topics covered
- We will focus mostly on core language elements rather than advanced features

Scientific Computing

"Nothing worth knowing can be taught"

- Oscar Wilde

Good computer programming is a skill learned through practice. This course will focus on practical experience.

Good vs. Bad Programming

Consider this "Program" fragment (it could be in one of many languages but that doesn't matter)

```
Func1( 2*c*f, 2*e, c*m, eps*s, x*s2,
2*x*b, 0.5*x*b2 );
```

What does it do?

Good vs. Bad Programming

```
Func1( 2*c*f, 2*e, c*m, eps*s, x*s2,
2*x*b, 0.5*x*b2 );
```

Clear version of the “Program”:

```
//Make pancakes

Mix ( 2*cup*flour, 2*egg, cup*milk,
pinch*salt, tablespoon*sugar,
2*tablespoon*butter,
0.5*tablespoon*baking_powder );
```

Some Elements of Good Programming

- Write clearly: Choose informative names
- Use existing library functions: Don't reinvent the wheel (or sine function)
- Avoid repetition or long rambling code
- Write self-contained code
- Use comments: authors, dates, mathematical algorithms and even references can and should appear in comments
- Format your program …

Some Elements of Good Programming

The bottom line:

- **The intent, flow and results of your programs should be obvious to any programmer**

Code that worked once is of little value if a future user (which could be you) can't easily understand what it is doing

Programming Languages

- There is a lot of hype about what is the "best" programming language
- No language is ideal for all applications
- Good programming is mostly independent of the language used

Programming Language

Considerations:

- Compatibility with existing code, co-workers
- Portability (availability)
- Readability
- Efficiency for your task

Programming Languages: A Short History

- Assembly = Machine Language (1940's,50's)
- Fortran: "Formula Translator" High level language (1954)
  Revisions: 60, 77, 90, 95, 2000, 2003, 2008
- Basic: "Beginners Language" (1964)
  Offspring: Visual Basic – drag and drop programming
- Pascal: Teaching tool (1968) "structured programming"
- C: Developed to write Unix OS (1972), Dennis Ritchie
  Offspring: C++ (1983) Bjarne Stroustrup, Java
- Perl: "Duct tape of the internet" (1987) Scripting language
- Java: "The Portable Language", SUN (1994)
- Python – flexible language, simple to write vs. fast to run, also quite hard to know what it really does

Programming vs. Scripting

Scientific computing focuses on efficiently calculating answers

- Understanding how programs are executed by computers is essential
- Compiled programming languages (C/C++, Fortran) translate directly into efficient computer code
- Scripts and some high level languages don't: makes it hard to understand what computers really do (e.g. Java, Perl, Python, Matlab)
- A good scientist can use both approaches, e.g. scripting to analyse data and test ideas and programming to run large scale computations

Programming Language Characteristics

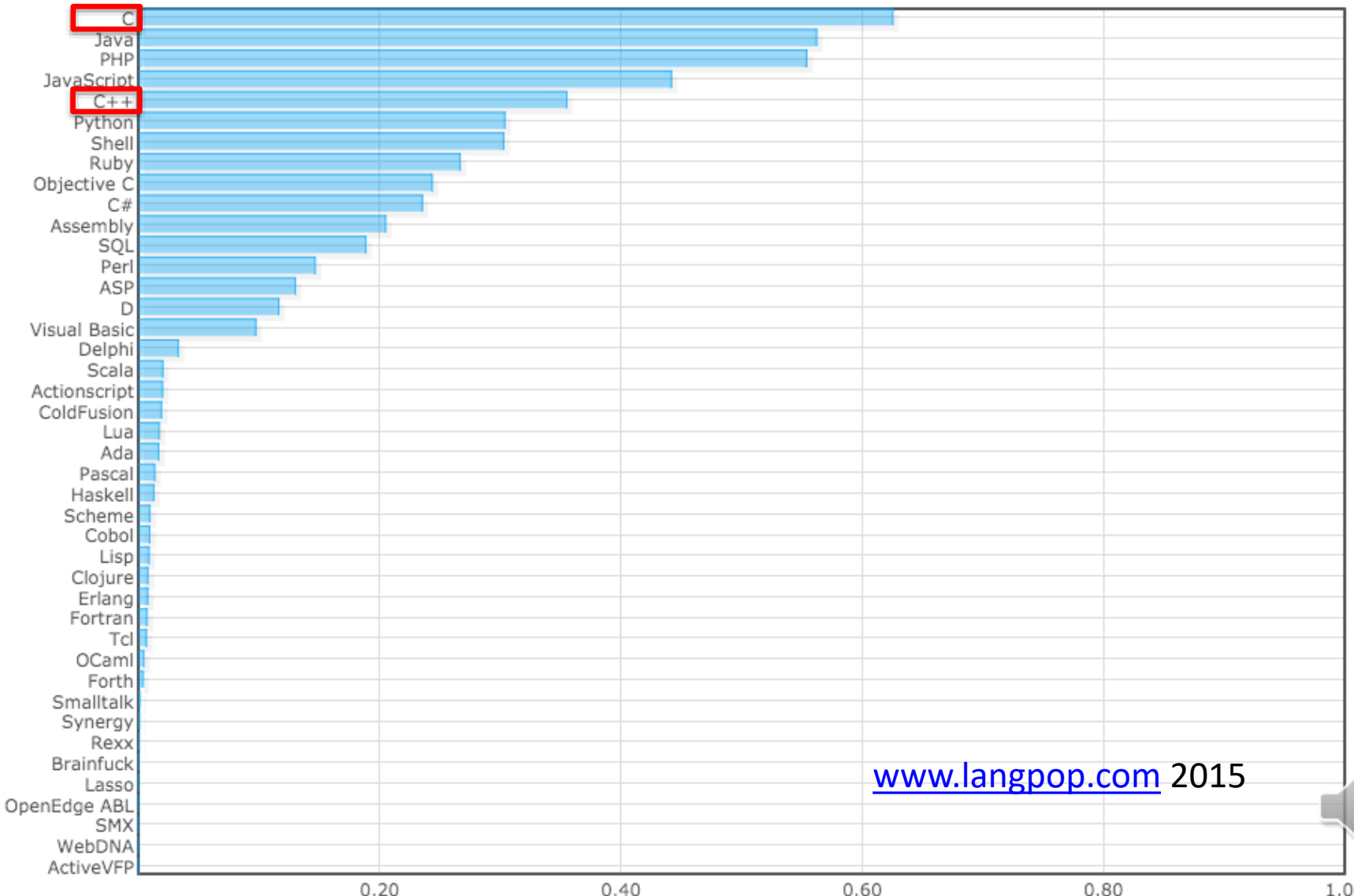
- C: Fast, popular, powerful, dynamic memory management, verbose for multi-dimensional arrays (most popular language in use today)
- C++: superset of C, designed to encourage object oriented style (one of many C related languages)
- Java: slow, portable (similar to C/C++)
- Fortran 77: fast, clean, lacks "modern" features, many math operations available in libraries, many legacy science codes Fortran (e.g. most climate codes)
- Fortran 90-2008: Fast, clean, supports object oriented, dynamic memory management, built in fast array operations, directly compatible with old Fortran

Programming Languages

- PHYS 2G03 will use C/C++: Essentially the common parts of modern C and C++
- We chose C/C++ because
  1) C/C++ are the most popular languages in use today
  2) C is easy to learn and helps you directly understand how computers work
  3) Hardware is programmed in C (e.g. GPUs, Unix, phones)
  4) C++ contains modern, object oriented features
  5) C/C++ looks good on your resume
  6) Popular languages are based on C/C++ (Java, C#, PHP)

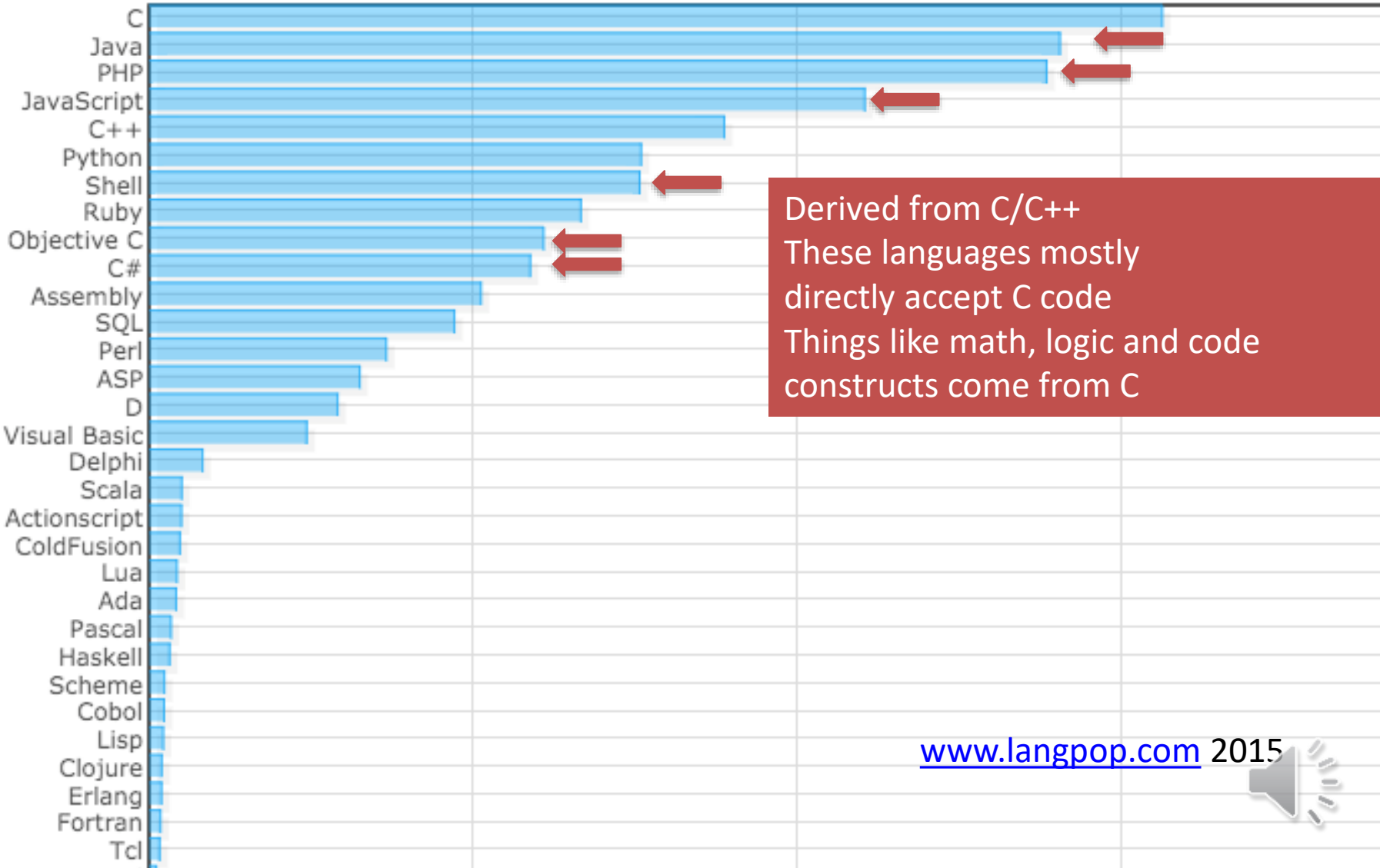
Note! A well structured program should look similar in any language. It is also relatively simple to translate code into other languages once you know what you are doing.

Programming Language Popularity



www.langpop.com 2015

Programming Language Popularity



Derived from C/C++
These languages mostly directly accept C code
Things like math, logic and code constructs come from C

www.langpop.com 2015