Introduction to Computer Programming

Course Notes

# The Course

This is course for those who have little or no experience in computer programming. The students will gain insight into programming ideas and goals to help decide if further study might be for them. The instructor will introduce some of the central ideas, techniques, and styles he has gained over his career. A primary goal will be to get the students to do actual programming. This is where the actual “feel” of programming is experienced. Students must bring their laptop computers and have rudimentary skills such as powering on, creating and modifying text files, and browsing the Internet.

# The Instructor

Ray Smith (Charles Raymond Smith) - [raysmith619@alum.mit.edu](mailto:raysmith619@alum.mit.edu)

* Years of Programming – engineering, scientific, financial
* Programming environments – embedded, systems, application
* Languages – C, C++, Perl, Java, Python, Assembly, shell, Fortran, PL1, …
* Currently investigating computer graphics/game programming

# Introduction

## Goals

### Understanding, empathy

## Requirements

* Interest in programming
* Laptop computer

## What is computer programming?

* Making the computer do what you want it to do
* Making reusable plan
* Like physical construction but “magical”

### Similar activities

* Mental / Abstract / Indirect
* Cooking recipes
* Travel directions
* Assembly instructions
* Physical – What you see is what you get
* Construction - Roads, Buildings
* Like Computer Programming
* Build once – use many times
* Builder and user(s) often different
* Building errors show up many times
* Building errors don’t always show up immediately
* Not so like Computer Programming
* Physical materials

Hard to copy

Hard to manipulate

Easy to see

## What’s Easy

* No so physical
* Creation / copying / moving is a snap
* Things don’t wear out – works today, tomorrow, forever
* Easy to change
* Working on new one, while the old one is running

## What’s Not Easy

* No so physical
* Can’t “see” it
* Everything is connected – unexpected consequences

## Why should I care about computer programming?

* We depend on programs, even if written by others
* Better knowledge of what is possible / difficult / easy
* To more easily deal with, instruct, cope with results.
* Understand alternatives – e.g. get someone else to do it

# A Computer Programming Roadmap

## Problem – What’s bothering you, your boss, …

In most endeavors, the most important thing is to decide what needs “fixing”, creation, …

### Goal(s) – What do we want to do about it

* What part or parts do we want fixed, created, …
* Which things are most important?
* Which need to be done first, last, before others.

### Plan(s) – Exactly, How do we want to do this?

* Has this or part of this, already been done?
* Can/Should we do part of this first?
* Is this a programming task?
* Can we divide this up in to smaller parts?
* What’s the most fun part?

### Create

* More detailed planning - choose program layout
* modules
* Subroutines
* Variables
* major calculations
* Write the code – text of the program
* May include checking reference material, e.g. Googling
* Possible experiments about how to do *X*
* Visual inspection – can save overall time / effort / pain

### Test – Let’s see if it does what we wanted.

As anything we may want do show to others, we probably should examine it before we expose our work.

Our computer programs often fall in to this category. The bigger the exposure the more important this is.

The more complicated the program the more testing is useful.

Dividing the program into parts is often helpful.

### Distribute - giving it out

* Packaging
* Documentation – user manual, online help

### Celebrate – 😊

### Do it again – Better?, On to the next?

# Program building blocks

### Computer(s) Parts

### Output – what we get: see/hear/feel

#### Display

* Dynamic
* Multi-dimensional
* Complex

#### Printer

* Time honored
* Simple

### Input – what we give: tell/ask

#### Keyboard

* Simple
* Easy to use
* Easy to program

#### Mouse / Touch screen

* Two dimensional
* Smart phones 🡪 becoming more accepted / expected

#### Other cool stuff

### CPU (Central Processing Unit) – the actor

#### Calculator – adding, subtracting, …

#### Storage – where we store stuff (data)

##### Memory – Fast

##### Storage – Large (Rotating Memory – Disks)

## Language Parts

Computer language(s) correspond to instructions to control computer parts.

As computers are evolving, the link between the physical (computer) and logical (programming language) changes.

As programming techniques evolve the programmer needs to think about the physical in less detail.

Still often good to keep in mind that the computer program execution, at some level, will execute on actual hardware.

Most productive to concentrate on the program goals

In the early stages (specification, planning), often ignore the physical computer attributes.

### Computation - Arithmetic

* Heavy use of arithmetic and algebraic notation
* Slight relaxation in semantics
* “=” 🡪 data on right travels to place on left
* “<”, “>”, “==” 🡪 compare value on left with that on right
* Code – try typing in these lines into a Python Shell

2+2

3\*4

2\*\*10

### Variables – Named Storage

* So we can use data by name
* Names are of the form:
* letter{one or more}digit\_or\_letter{zero or more}
* examples: a, a1, my\_variable
* Storing a value into a variable:
* variable\_name = value

a = 2 # Store 2 in a

b = 3 # Store 3 in b

a + b # The contents of a with the contents of b, printing the results

Done in online Python interpreter:

23:18 ~ $ python

Python 2.7.6 (default, Oct 26 2016, 20:30:19)

[GCC 4.8.4] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> a = 2

>>> b = 3

>>> a+b

5

>>> a\*\*b

8

>>>

### Move

### “=” 🡪 data on right travels to place on left

### Compute

* Arithmetic + Algebra

### Encapsulation – Divide and Conquer

* Parts of the problem placed in functions / subroutines
  + Called (invoked/started) with *name*(comma-separated list of parameter values)

### Compare

### Decide

## Computer Programming Languages

### Python –

* Highly popular programming language today
* This course will use Python
* Facilitates Object Oriented Programming

### Java

* Facilitates Object Oriented Programming

### C / C++

* Highly efficient
* Often the basis for computationally intense modules / packages
* C++ - Object Oriented version of C

### Assembly Language – Closest to the computer hardware

* Different for every hardware
* Still not the hardware
* Little used today
* Even, when used, usually only when necessary

# Let’s Write a Program

## Goals – What’s it going to do? Or at least, what would be nice?

## Elaborate the goals of our program

## Prioritize – e.g. No 3D graphics display first round 😊

## The Plan – How are we going to make it do what we want?

## Or at least, some of what we want?

## Displaying Results/Request

# A “Real” Program - I’m thinking of a number…

## Traditional “20 questions”

## Program picks number and player guesses: “Is it greater than 5” continually.

## Main player choices are “> N”, “<N”, “=N” where N is a number, and “Q” for quit

# Appendix

## Online Python Resources

### python.org

<https://www.python.org>

### pythonanywhere

<https://www.pythonanywhere.com>

* Need to create an account
* Provides a persistent Linux-like account
* Bash shell

### Jdoodle

<https://www.jdoodle.com/python3-programming-online>

* Need to create an account to save file/project

### pythonfiddle.com

<http://pythonfiddle.com>

### onlinegdb

<https://www.onlinegdb.com/>

* Need to create an account to save project
* Has a debugger

### paiza.io

<https://paiza.io/en/projects/new?language=python3>

<https://paiza.io/en/languages/python3>

### [amrdraz.github.io/python-debugger/](http://amrdraz.github.io/python-debugger/)

<http://amrdraz.github.io/python-debugger/>

* Simple example- not yet investigated