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

* 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 about Computer Programming

* 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 about Computer Programming

* No so physical
* Can’t “see” it
* Everything is connected – unexpected consequences
* Often not easy to tell how close to done

## 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

# Let’s Do Some Programming!

* Seeing is believing
* Lots left unsaid
* Try, Try, Try
* Think arithmetic / algebra

Using online Python: <https://www.python.org>

After clicking the Launch Interactive Shell (the >\_ button)

Note that the “>>> ” and “… ” are prompts displayed by the Python interpreter as guides and NOT typed by you.

Python 3.6.0 (default, Jan 13 2017, 00:00:00)

[GCC 4.8.4] on linux

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

>>> 2+2

4

>>> a=2

>>> b=3

>>> a+b

5

>>>

Python shell (try <http://pythonfiddle.com>)

* Enter the following in the top (code) window
* then click the **Run** button

a = 4

b = 3

print(a+b)

The following should be displayed in the lower (output) window

5

* How many ways can you make interesting calculations?

# Adding a few things

## Variables – Places to store data/values

* Facilitates using our data/results
* Data stores have names
* Names are of the form:
* Start with a letter (a-zA-Z\_)
* Followed by zero or more digits (0-9) or letters
* examples: a, a1, my\_variable, a2or3
* NOT: 1, 1a, my-variable, “ray,smith”
* Storing a value into a variable:
* *variable\_name* = *value*
* Examples:

a = 1

a1 = 5.7

my\_variable = my\_variable + 1

## Printing – send stuff to the screen

* **print(***value***)** - *value* goes to the screen
* **print(***value1, value2, …***)** - *values* go to the screen
* End of line added after value(s)
* Example:
* (Using <https://www.jdoodle.com/python3-programming-online>):

print("Hello World - all on the same line")

print("""Multi-line string

on multiple lines.

Another line goes here.

""")

Output:

Hello World - all on the same line

Multi-line string

on multiple lines.

Another line goes here.

* NOTE: Older Python versions(2.x) used print “……” without parentheses

## Text – We speak, usually in words😊

### Text strings –

“…..stuff….” placed all on one line

OR

“””…stuff may be multiple lines “”” (three adjacent “)

Example:

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

prompt = "Please Enter your answer:"

print("Prompt:", prompt)

help\_message = """

We are glad you are participating

Your choices are as follows

1. Go home

2. Stay here

3. Speak up

"""

print("help\_message:", help\_message)

Output:

Prompt: Please Enter your answer:

help\_message:

We are glad you are participating

Your choices are as follows

1. Go home

2. Stay here

3. Speak up

### Exercise:

Calculate 5 years total interest on $1M, given an annual interest of two percent.

* Name the variables
* What’s the calculation
* Extra Credit: 10 years ?, 100 years?

Hint (for lesser ammounts):

# $100, 3 years, %1 interest, very short variable names

p = 100

i = .01

mult = 1 + i # Multiplier == 1 + interest

tot = p \* mult \* mult \* mult # Three years

ti = tot - p

print("Total interest for 3 years:", ti)

Output

Total interest for 3 years: 3.0301000000000045

## Decisions – Without them choices are few

The primary Python testing / decision constructs are:

**if** – Either it is or it isn’t

if is true **else:** false

**while** – Loop Till not True

**for** – Loop from beginning value till ending value

Decision code follows the form:

*Testing-construct boolean-value-expression* **:**

*Indented-code-lines*

Boolean-value-expression examples:

*Value* compare-operator *Value*

Where compare-operator can be:

**>**, **<**, **==**, **!=**, **>=**, **<=**

*Indented-code-lines* – code lines indented to the same amount

* One or more lines
* Can include additional decision constructs
* Special Cases:
* **break** – quit the indented list
* **continue** – quit the indented list and do loop again
* **pass** – do nothing – just a place holder

Example (Using: <https://www.jdoodle.com/python3-programming-online>)

if 2 < 3:

print("if true=>", "2<3", 2<3)

if 2 > 3:

print("if true=>", "2>3", 2>3)

print("After if 2 > 3")

if 2 > 3:

print("if true=>", "2>3", 2>3)

else:

print("else false=>", "2>3", 2>3)

Output:

if true=> 2<3 True

After if > 3

else false=> 2>3 False

Example:

n = 1

max = 3

while n <= max:

print("n:", n, "max:", max)

n = n+1

print("After loop")

Output:

n: 1 max: 3

n: 2 max: 3

n: 3 max: 3

After loop

Example:

print("for loop")

for i in range(1,3):

print("i:", i)

Output:

for loop

i: 1

i: 2

### Exercise – Sum the numbers from 1 to N

Sum the squares.

# The Computer – Does the program’s work

## 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)

# Programming Language

Computer languages are the languages in which we tell the computer what and how to do what we want.

Our programs are written in these languages.

The parts of the computer language correspond to instructions to control specific 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 parts in less detail.

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

Most often it is productive to concentrate on the programs 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

## Distinction between integer (whole number) and floating point (fractions)

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

## Encapsulation Part 1 – Functions == Subroutines

* Divide and Conquer
* Parts of the problem placed in functions
  + Called (invoked/started) with *name*(comma-separated list of parameter values
* Function Definition – what it is going to do
  + Name – same style/restrictions as variable name
  + Parameters – data which is passed to the function
    - Comma-separated list of parameter specifications
    - Simplest is a set of variable names
  + Body – what the actions are
    - Regular code
    - Uses data passed in parameters by parameter names
    - Example:

def add2(value1, value2):

sum = value1 + value2

return sum

* + Function Calling / Invocation
    - How do we call the function?
    - Called with the function name
    - Including parenthesized list of values
    - The values are passed to the function
    - Example:

sum1 = add2(1,2) # sum1 gets 1+2

sum2 = add2(3,4) # sum2 gets 3+4

### Exercise – special product

Write a function product(factor1, factor2, factor3) that returns the product of the values factor1, factor2, factor3. Test it on the following: .5, .4, .3; 1, 2, 3; -1, -1, -1;

### Exercise – staircase function

Write a function staircase(first\_n, last\_n) that returns the sum of the numbers first\_n, first\_n+1, …, last\_n. Test it on the following: 1,10; 1,100; 5, 10; 5,5; 10,1;

Did you use:

**for** – why?

**while** – why?

neither – why?

neither – why?

## Encapsulation Part 2 – Classes

* Data and Function
* Facilitate Object Oriented Programming

## Compare

# 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 / functions
* 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.

Before we expose our work to others, we probably should examine it.

The bigger the exposure, the more important the examination.

The more complicated - the more testing is useful.

Divide and conquer.

### Distribute - giving it out

* Packaging
* Documentation – user manual, online help

### Celebrate – 😊

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

## Computer Programming Languages

* Many popular languages in use today
* Most similar in form / function

### 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 Bigger 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

## Learning Python

### [Instant Python](http://www.hetland.org/python/instant-python.php) A minimal crash course by Magnus Lie Hetland.

Note that this nice account uses Python2.x so, for the code to work on Python3.x, change the print “……” instances to print(“…..”) instances.