Introduction to Computer Programming

Course Notes

Contents

[Introduction to Computer Programming 1](#_Toc522023789)

[1. Introduction 4](#_Toc522023790)

[1.1 The Course 5](#_Toc522023791)

[1.2 The Instructor 5](#_Toc522023792)

[1.3 Requirements 5](#_Toc522023793)

[2. What is computer programming? 5](#_Toc522023794)

[2.1 Computer is a VERY Obedient Servant 7](#_Toc522023795)

[2.2 Similar activities 8](#_Toc522023796)

[2.3 What's Easy - No so physical 9](#_Toc522023797)

[2.4 What's Not Easy - No so physical 9](#_Toc522023798)

[2.5 Do I care Even if I don't Actually Program? 10](#_Toc522023799)

[3. Let's take a look! 10](#_Toc522023800)

[4. Comments – what we want 12](#_Toc522023801)

[5. Variables – Places to store values 13](#_Toc522023802)

[6. Python Language Guidelines 14](#_Toc522023803)

[6.1 An instruction (or statement) is on one line 14](#_Toc522023804)

[6.1.1 One-line exceptions 14](#_Toc522023805)

[6.2 A group of statements 14](#_Toc522023806)

[6.3 Multi-part statements 15](#_Toc522023807)

[7. Adding a few things 16](#_Toc522023808)

[7.1 Printing – send stuff to the screen 17](#_Toc522023809)

[7.1.1 Exercise: 17](#_Toc522023810)

[7.2 Text – We speak, usually in words 18](#_Toc522023811)

[7.2.1 Text strings – 18](#_Toc522023812)

[7.3 Decisions – Without them choices are few 19](#_Toc522023813)

[7.3.1 if – Is it true 19](#_Toc522023814)

[7.3.2 if is true else: false 19](#_Toc522023815)

[7.3.3 while – Loop Till not True 20](#_Toc522023816)

[7.3.4 for – Loop over a group of values 20](#_Toc522023817)

[7.3.5 More Decision Details 20](#_Toc522023818)

[7.3.6 Exercise: Sum the numbers 22](#_Toc522023819)

[7.4 Input – Get the users input 23](#_Toc522023820)

[7.4.1 Exercise: Let the user tell 23](#_Toc522023821)

[7.5 Encapsulation Part 1 – Functions 25](#_Toc522023822)

[7.5.1 Exercise – special product 26](#_Toc522023823)

[7.5.2 Exercise – staircase function 26](#_Toc522023824)

[7.6 Encapsulation Part 2 – Classes 26](#_Toc522023825)

[8. How to Write a Program 26](#_Toc522023826)

[8.1 Goals – What's it going to do? Or at least, what would be nice? 27](#_Toc522023827)

[8.2 Elaborate the goals of our program 27](#_Toc522023828)

[8.3 Prioritize – e.g. No 3D graphics display first round 27](#_Toc522023829)

[8.4 The Plan – How are we going to make it do what we want? 27](#_Toc522023830)

[8.5 Or at least, some of what we want? 27](#_Toc522023831)

[8.6 Displaying Results/Request 27](#_Toc522023832)

[9. A program we can do 27](#_Toc522023833)

[9.1 Game Description 27](#_Toc522023834)

[9.1.1 Traditional "20 questions" 27](#_Toc522023835)

[9.1.2 Program picks number and player guesses, using questions like "Is it greater than 5". 27](#_Toc522023836)

[9.1.3 Main player choices are "> N", "<N", "=N" where N is a number, and "Q" for quit 27](#_Toc522023837)

[9.2 Plan - Simplify – Iterate 28](#_Toc522023838)

[10. More Python to Come… 28](#_Toc522023839)

[10.1 A great Start – www.python.org 28](#_Toc522023840)

[10.2 Continuing Education – Judicious Online Search 29](#_Toc522023841)

[11. Structures to aid in implementing tasks 30](#_Toc522023842)

[11.1 Data Structures 30](#_Toc522023843)

[11.1.1 Primitive Data Structures 30](#_Toc522023844)

[11.1.2 Compound Data Structures 31](#_Toc522023845)

[11.2 Programming Structures 33](#_Toc522023846)

[11.2.1 More on Functions 33](#_Toc522023847)

[11.2.2 Class 33](#_Toc522023848)

[11.3 More Modules 33](#_Toc522023849)

[12. A Computer Programming Roadmap 33](#_Toc522023850)

[12.1 Problem – What's bothering you, your boss, … 34](#_Toc522023851)

[12.1.1 Goal(s) – What do we want to do about it 34](#_Toc522023852)

[12.1.2 Plan(s) – Exactly, how do we want to do this? 34](#_Toc522023853)

[12.1.3 Create 34](#_Toc522023854)

[12.1.4 Test – Let's see if it does what we wanted. 35](#_Toc522023855)

[12.1.5 Distribute - giving it out 35](#_Toc522023856)

[12.1.6 Celebrate – 35](#_Toc522023857)

[12.1.7 Do it again better? On to the next? 35](#_Toc522023858)

[12.2 Computer Programming Languages 35](#_Toc522023859)

[12.2.1 Python – 35](#_Toc522023860)

[12.2.2 Java 35](#_Toc522023861)

[12.2.3 C / C++ 35](#_Toc522023862)

[12.2.4 Assembly Language – Closest to the computer hardware 36](#_Toc522023863)

[13. Computers / Programming an Overhead Look 36](#_Toc522023864)

[13.1 The Computer – Does the program's work 36](#_Toc522023865)

[13.1.1 Output – what we get: see/hear/feel 36](#_Toc522023866)

[13.1.2 Input – what we give: tell/ask 36](#_Toc522023867)

[13.1.3 CPU (Central Processing Unit) – the actor 37](#_Toc522023868)

[13.2 Programming Languages 37](#_Toc522023869)

[13.3 Output 38](#_Toc522023870)

[13.4 Input 38](#_Toc522023871)

[13.5 Computation - Arithmetic 38](#_Toc522023872)

[13.6 Distinction between integer (whole number) and floating point (fractions) 39](#_Toc522023873)

[14. Appendix 39](#_Toc522023874)

[14.1 Joining Zoom, as a new member 39](#_Toc522023875)

[14.2 Joining the Class Meeting in Session 41](#_Toc522023876)

[14.3 Using PythonAnywhere 42](#_Toc522023877)

[14.4 Online Python Resources 42](#_Toc522023878)

[14.4.1 python.org https://www.python.org 42](#_Toc522023879)

[14.4.2 PythonAnywhere 43](#_Toc522023880)

[14.4.3 Other Online Python Resources 43](#_Toc522023881)

[14.4.4 Learning Python 43](#_Toc522023882)

# Introduction

## The Course

This is a short course for those who have little or no experience in computer programming. The students will gain insight into programming ideas and goals. We will be using Python, a very popular and powerful programming language. My aim is to help the participants decide if further programming study might be for them. I will introduce some of the central ideas, techniques, and styles I've gained over my career. Our method will be to get the students to do some actual programming. This is where the actual "feel" of programming can be experienced. Participants must bring their laptop computers and have basic computer skills such as powering on, creating and modifying text files, and browsing the Internet.

## The Instructor

Ray (Charles **Ray**mond Smith) - [raysmith@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 for the very young (3-8 years)

## Requirements

* Interest in programming
* Laptop computer

# What is computer programming?

* Controlling the computer
* Telling the story that gets done
* The computer language IS the computer
* Providing the instructions to achieve the goal
* Making the story reusable
* Like physical construction but "magical"

## Computer is a VERY Obedient Servant

Does exactly what you tell it – hopefully what you want

Example: Car

* Most cars are alike
* Does exactly what you tell it – hopefully what you want
* Inputs/controls
* Steering wheel, Pedals, Signals, …
* Outputs / Signals
* Wheels, Lights, Windows
* Defined actions
* Same action every time – almost – e.g. forward / backward
* Car gets the blame…

## 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 - 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
* Often not easy to tell how close to done

## Do I care Even if I don't Actually Program?

* 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 take a look!

* Seeing is believing
* Lots left unsaid
* Try, Try, Try
* Think arithmetic / algebra
* Python as a super calculator

Using python.org's IDLE: https://www.python.org

Please refer to the Appendix for detailed instructions on how to down load python's latest release.

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

>>>

Exercises:

1. Assuming: salary: 1000, New salary: 1250
2. What's the raise?
3. What's the percent raise?
4. If the new salary was 1500, 1600, 1700?
5. How many seconds:
6. In a day?
7. In a week?
8. in a year of 365 days?
9. Between 8AM Monday and 5PM Friday?
   * Using online Python: <https://www.pythonanywhere.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

7

* + - How many ways can you make interesting calculations?

# Comments – what we want

* + - What we want, think, expect
    - Not "code" but more important
    - Comment 🡺 what you want vs. code 🡺 what you did
    - **#** text to indicate what you want, think, …end of line
    - **"""** Any text you choose, really used as a code text string, but often used as a programming comment. This comment often goes on for several lines ending at the next **"""**

Example:

"""

Program Goals

See web site: www.our.place.here

"""

# Setting up initial variables

# These values are updated as necessary

max = 10 # Maximum value allowed

n = 1 # Set initial value

# Variables – Places to store values

* Eases the use/reuse our data/results
* Variables have names
* Names have rules:
  + - Start with a letter (a-zA-Z\_)
    - Followed by zero or more digits (0-9) or letters
  + examples:
  + Valid names: a, a1, my\_variable, a2or3
  + Invalid names: 1, 1a, my-variable, "ray,smith"
  + Storing a value into a variable:
  + variable\_name = value
  + Code:

a = 1

a1 = 5.7

my\_variable = my\_variable + 1

**Demonstrate A program we can do**

# Python Language Guidelines

## An instruction (or statement) is on one line

YES:

sum = 1 + 2

NO:

sum = 1 +

2

### One-line exceptions

* Parenthesized grouping may be on multiple lines
  + Sum = (1

+ 2)

## A group of statements

* + - Are executed in order
    - Must start at same indentation (from left)

YES:

start = 1

last = 3

increment = 1

NO:

start = 1

**last = 3**

increment = 1

## Multi-part statements

* + Start with a statement ending with a "**:**"
  + Followed by an indented group of statements
  + Example:

# Simple test

**if** gas < gas\_minimum**:**

print("Check for money")

print("Add gas")

* + Example:

# Filling gas tank

# Setting basic definitions

gas\_can = 2.5 # Capacity

gas\_tank = 13.5 # tank capacity

# Setting initial values

initial\_level = 5 # Initial amount

filled\_amount = 0 # amount filled

gas\_level = initial\_level

# Go through the process until filled

**while** gas\_level < gas\_tank**:**

gas\_level = gas\_level + gas\_can

filled\_amount = filled\_amount + gas\_can

print("filled amount", filled\_amount)

print("gas\_level:", gas\_level)

# ??? Did we leave something out?

# Adding a few things

## 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 online Python: <https://www.pythonanywhere.com>

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

### Exercise:

1. 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 amounts):

# $100, 3 years, %1 interest

# very short variable names

p = 100

APR = .01

mult = 1 + APR # Multiplier == 1 + interest

tot = p \* mult\*\*3 # Three years

ti = tot - p

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

Output

Total interest for 3 years: 3.0301000000000045

## Text – We speak, usually in words😊

### Text strings –

"…. stuff…." placed all on one line

OR

"""…stuff may be multiple lines """ (three adjacent ")

Example:

Using python.org: [www.python.org](http://www.python.org) IDLE

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

**Show A program we can do**

## Decisions – Without them choices are few

The primary Python testing / decision key words including: **if**, **else**, **while**, **for**.

### if – Is it true

**if** gas\_level + gas\_can > gas\_tank**:**

print("We're going to spill!")

### if is true else: false

**if** gas\_level + gas\_can > gas\_tank**:**

print("We're going to spill - or do something!")

**else:**

gas\_level = gas\_level + gas\_can

filled\_amount = filled\_amount + gas\_can

### while – Loop Till not True

**while** cash\_on\_hand > price\_of\_gas \* gas\_can**:**

print("We can afford another can of gas")

purchase = price\_of\_gas \* gas\_can

cash\_on\_hand = cash\_on\_hand - purchase

### for – Loop over a group of values

Supports looping over a specific set of values

for i in range(1, 11): # Does 1 through 10

print("i: ", i)

Same results as the following:

i = 1

while i < 11:

print("i: ", i)

i = i + 1

### More Decision Details

#### Some Definitions:

***Value*** – a variable or constant that is or contains a single value.

Examples: 5, sum, "a text string"

***Expression*** – a group of terms that expresses a value

Examples: 5, sum, sum+5

***Boolean*** – a value or expression that is or contains only one of two values – True or False

#### General Decision Code Pattern:

*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 loop
  + **continue** – stop doing the indented portion and start next go-round at the beginning of this portion
  + **pass** – do nothing – just a place holder
    - Often used, when there is nothing to do here, but the language rules say we have to have something here

Example:

Using online Python: <https://www.pythonanywhere.com>

if 2 < 3:

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

if 2 > 3:

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

print("After test for 2 > 3")

if 2 > 3:

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

else:

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

Output:

tested true=> 2<3 True

After test for 2 > 3

tested 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

1. Sum the numbers from 1 to 100
2. List the numbers from 1 to 50, that are evenly divisible by 2, 3, 4, and 5.

Hint: I found the following by an Internet search for

python evenly divisible

n % k == 0

evaluates true if and only if n is an exact multiple of k. In elementary math this is known as the remainder from a division.

Example:

The Following code:

if 10 % 2 == 0:

print("10 is divisible by 2")

if 10 % 3 == 0:

print("10 is divisible by 3")

Outputs:

10 is divisible by 2

1. Sum the numbers whose sum is less than 1000000, listing the sum and the last number added.

## Input – Get the users input

With input we can reuse the program for different situations without making changes to the program.

* + - **input()** – returns line user typed on keyboard
    - **input(**"Please enter your answer: "**)** – prompts user, then returns text entered by the user (usually on the keyboard)

### Exercise: Let the user tell

1. Tell the users you are going to sum numbers
2. Ask the user to enter the sum limit (Limit).
3. Sum the numbers whose sum is less than Limit, listing the sum and the last number added.

## Encapsulation Part 1 – Functions

* + Divide and Conquer
  + Hide the particulars
  + Delay consideration
  + 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?

## Encapsulation Part 2 – Classes

* + - Data and Function
    - Facilitate Object Oriented Programming
    - For later discussion

# How to Write a Program

How should we write a program? How should we write a good program? Does it need to be the best?

## 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 program we can do

## Game Description

### Traditional "20 questions"

### Program picks number and player guesses, using questions like "Is it greater than 5".

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

## Plan - Simplify – Iterate

* + - Simplification sometimes helps the user
    - Accept number and tell if greater, less, equal
    - Iteration allows us to learn and test along the way
    - First:
    - loop asking number
    - print number entered
    - Second:
    - Set target value
    - Quit if number entered number equals target
    - Third:
    - Announce if guess greater, less or equal
    - Fourth:
    - Announce goals, rules before start
    - Fifth:
    - Ask user if they want to play again
    - Sixth:
    - Set the target to a random number

# More Python to Come…

## A great Start – [www.python.org](http://www.python.org)

1. Download/Install your Python3.7.0
2. IDLE from the Start Programs menu
   1. Help
   2. Python docs
   3. The Python Tutorial
      1. 3. An Informal Introduction to Python…4…

## Continuing Education – Judicious Online Search

* Immense information – much of it good
* Pair down
  + Start search string with "python" or something assured to relate to python
  + Follow with as precise a word(s) as you can
  + Don't make long phrases unless certain all words are relevant e.g., cut/past error message
* E.g., python random number between

Result:

Then the third line: print **random**.randint(1,100) will automatically select a **random**integer **between** 1 and 100, inclusive, for you. The process is fairly simple. Basically this code will generate a **random number between** 1 and 20, and then multiply that **number** by 5.Feb 15, 2017

*NOTE that this answer did not mention that one needs to include the "import random" earlier in one's program*

* Some very good sites, e.g., Stackoverflow.com, often follow the form of presenting the question, which often is "why doesn't this work", followed by a number of suggested solutions with the most popular (probably the best) preceded with a green check mark.

# Structures to aid in implementing tasks

## Data Structures

### Primitive Data Structures

#### Integer

Examples: 1,2, 10000

#### Float

Examples: 1.2, 1.234 1.2E3

#### String

Examples: "ABC", 'def', """GHI\nKLM"""

#### long\_string = ("first"

#### + "second"

#### + "third"

#### )

Strings can be indexed

name = "Ray"

name[0] == "R"

name[1] == "a"

name[2] == "y" == name[-1]

name[0:1] == "Ra" # [*included* : *excluded*] – called a slice

**Attempting to use an index that is too large will result in an error**

#### Boolean

Examples: True, False, 1>2

### Compound Data Structures

#### List

names = ["tom", "dick", "harry", "sue", "jill", "mabel" ]

evens = [ 0, 2, 4, 6, 8, 10]

names[0] == "tom"

names[-1] == "mabel"

names[-1] = "jane" # change mabel to jane

len(names) == 6

names.append("joe") # Add "joe" to end of list

len(names) == 7 # Length increased by append

names.extend(["fred", "sue"]) # add list to end of list

names.insert(1, "tom2") # insert tom2 before 2nd entry

t2 = names.pop(1) # remove 2nd entry

last\_insert = names.pop()

del names[1] # delete 2nd entry

#### Dictionary – associative array

Elements are key/value pairs

Examples:

name\_ages ={'tom' : 10, 'joe' : 11, 'jane' : 12 }

name\_ages['charles'] = 1.1 # Add Charles

name\_ages.pop('charles') # remove 'charles'

# Error if not present

Name\_ages.pop('charss', None) # remove 'charles'

# returns None

# if key not found

#### File

##### File Output

file\_name = "data.txt"

**with** open(file\_name, 'w') **as** foutp**:**

foutp.write("#" + file\_name + "\n")

**for** i **in** range(10)**:**

foutp.write("data line " + str(i+1) + "\n")

##### File Input

**with** open(file\_name) **as** finp**:**

**for** line **in** finp:

print(line, end="")

##### Exercises – Notes program

Write a "Notes" program. The program will display lines

from a text file, containing a given text string.

Test it with the following:

file name = "people.notes"

text = "Watertown"

Implement with the following iterations, saving each iteration in a separately named program:

0. Setup test file(s): "test.notes", "people.notes"

1. Read specific file e.g. "test.notes", printing out all lines
2. Print only lines containing "student"
   * + How to match lines ? Google "python search for substring" ?
     + Support case insensitive match (Student, STUDENT)
3. Prompt for, then accept file name, pattern
   * + Repeat file prompt if file open error (can't find file)
4. [Extra Credit] Support multiple text patterns
   * + What do we mean by multiple patterns? Contains all patterns? Contains any patterns?

## Programming Structures

### More on Functions

#### Keyword parameters

* facilitate parameter recognition
* support optional parameters
* Calling Format:

*function\_name***(**[*arg1***,** arg2**,** …,]*key1\_name***=***value*

**,***ke2\_name***=***vaue*…**)**

Example: myprint("text line", end="") # no new line

* Definition:

**def** myprint**(**text, end**=**None**):**

print(text, end=end) # Has an end parameter

### Class

## More Modules

# 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

# Computers / Programming an Overhead Look

## 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 Languages

* 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.

## Output

* print – Text to the screen or file
* write – bulk output to file
* Graphics – pictures, etc. – more complicated

## Input

* input() – get a line from user / keyboard
* read – bulk input from file

## 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)

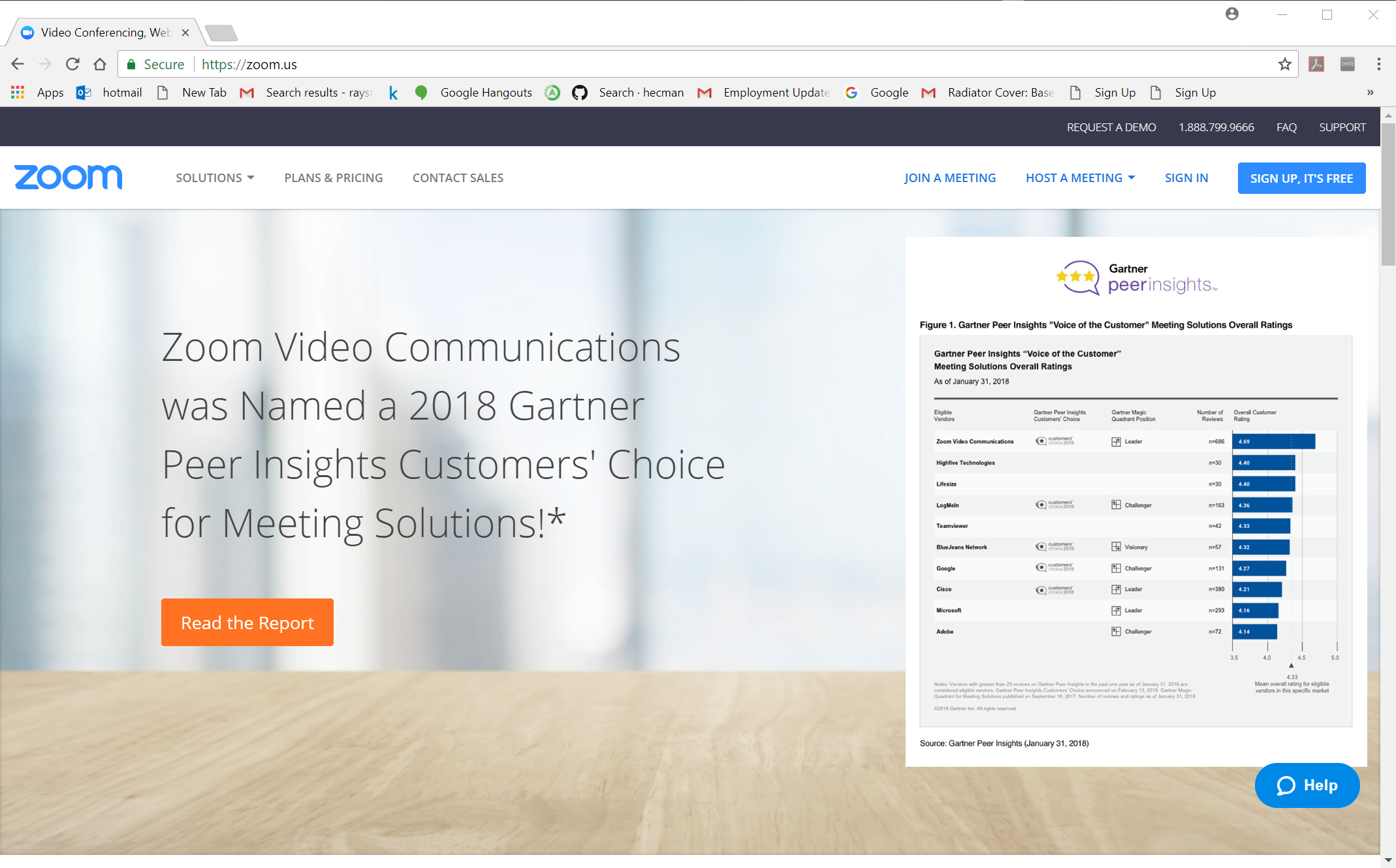
# Appendix

## Joining Zoom, as a new member

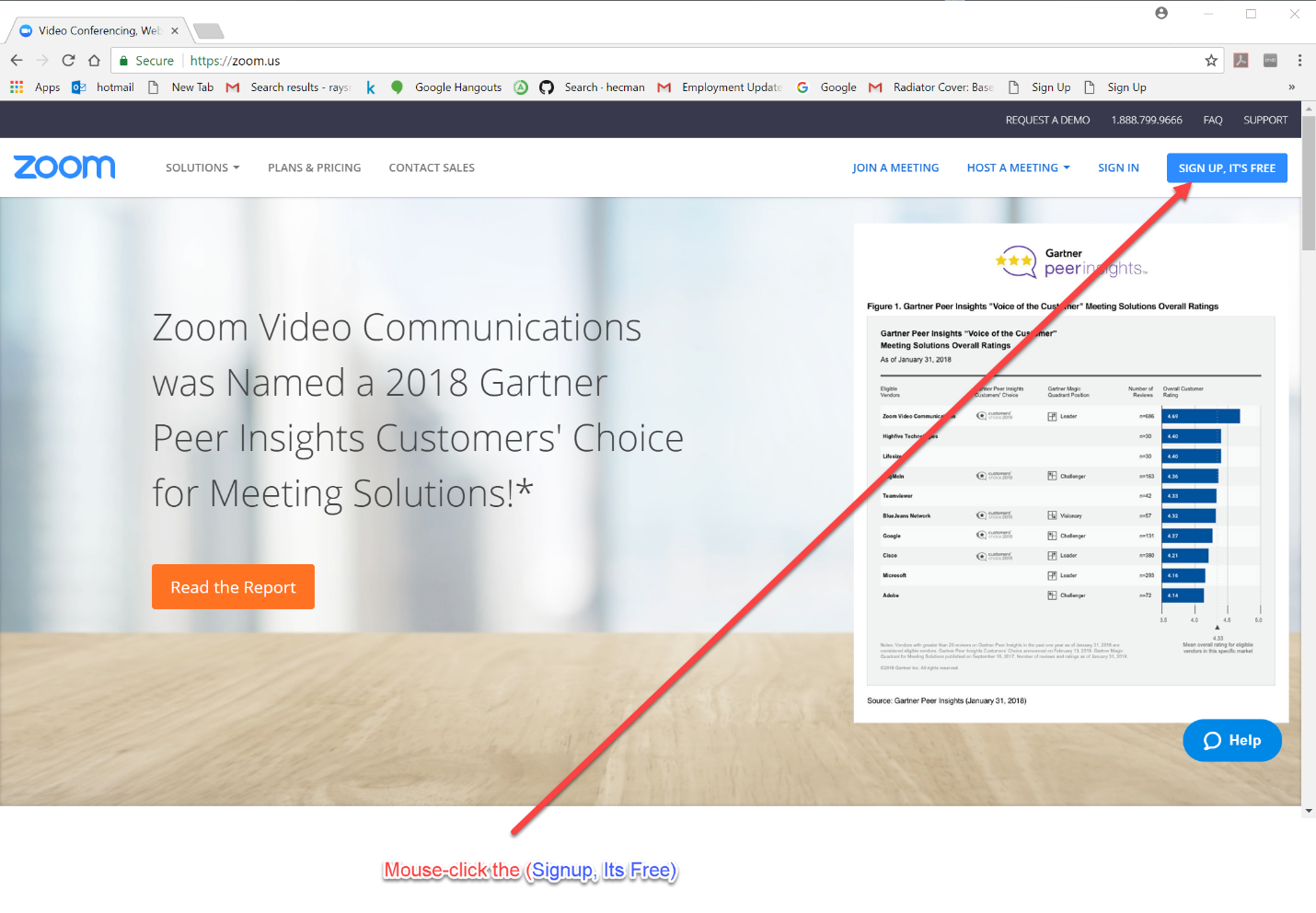
1. Click the following or paste the following address in to your browser address:

[https://zoom.us](https://zoom.us/j/2821636776)

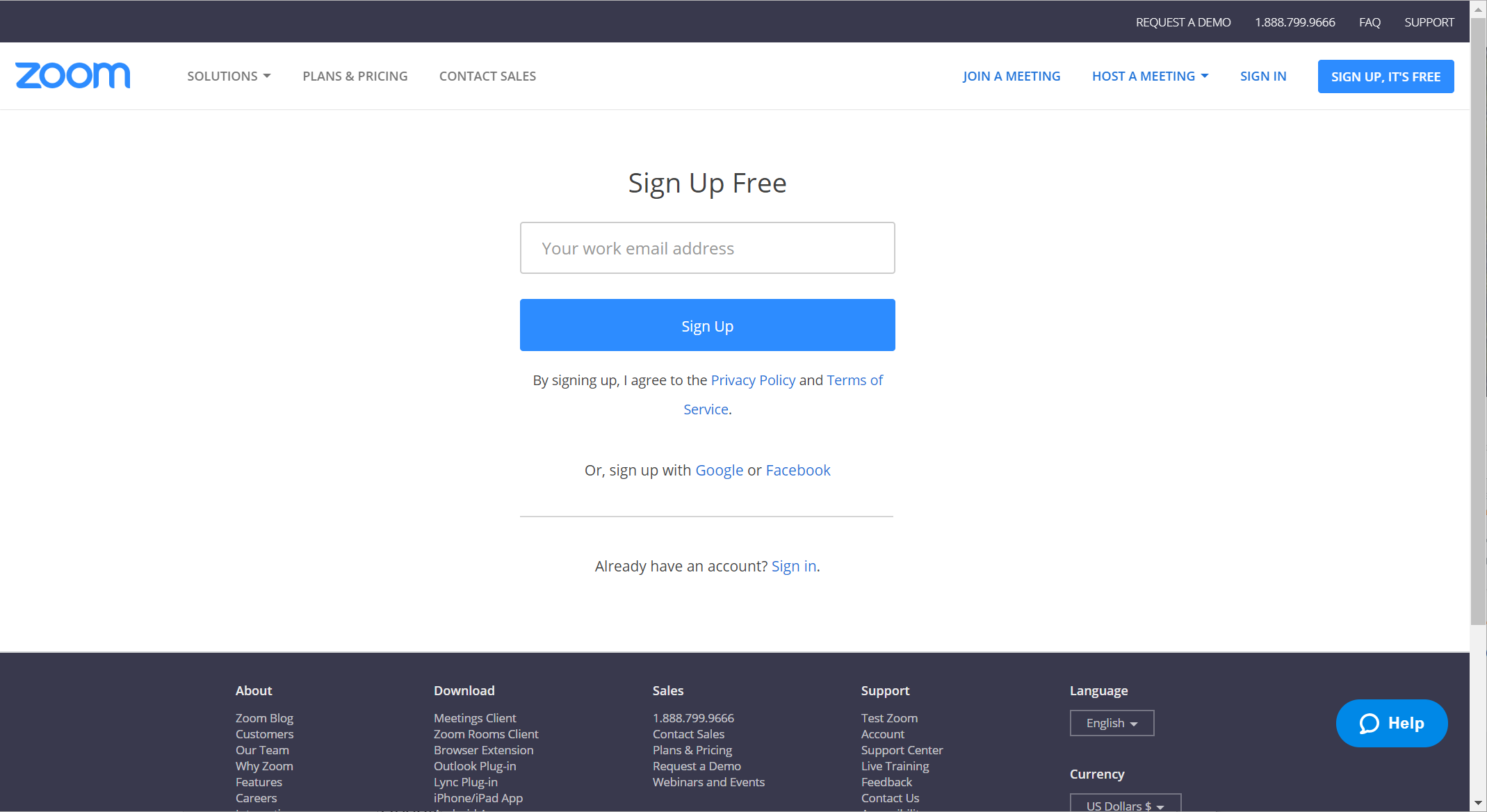
The following window should appear:



Mouse-click the (SIGN UP, IT'S FREE) button in the upper right.



The following window should appear:



## Joining the Class Meeting in Session

1. Click the following or paste the following address in to your browser address:

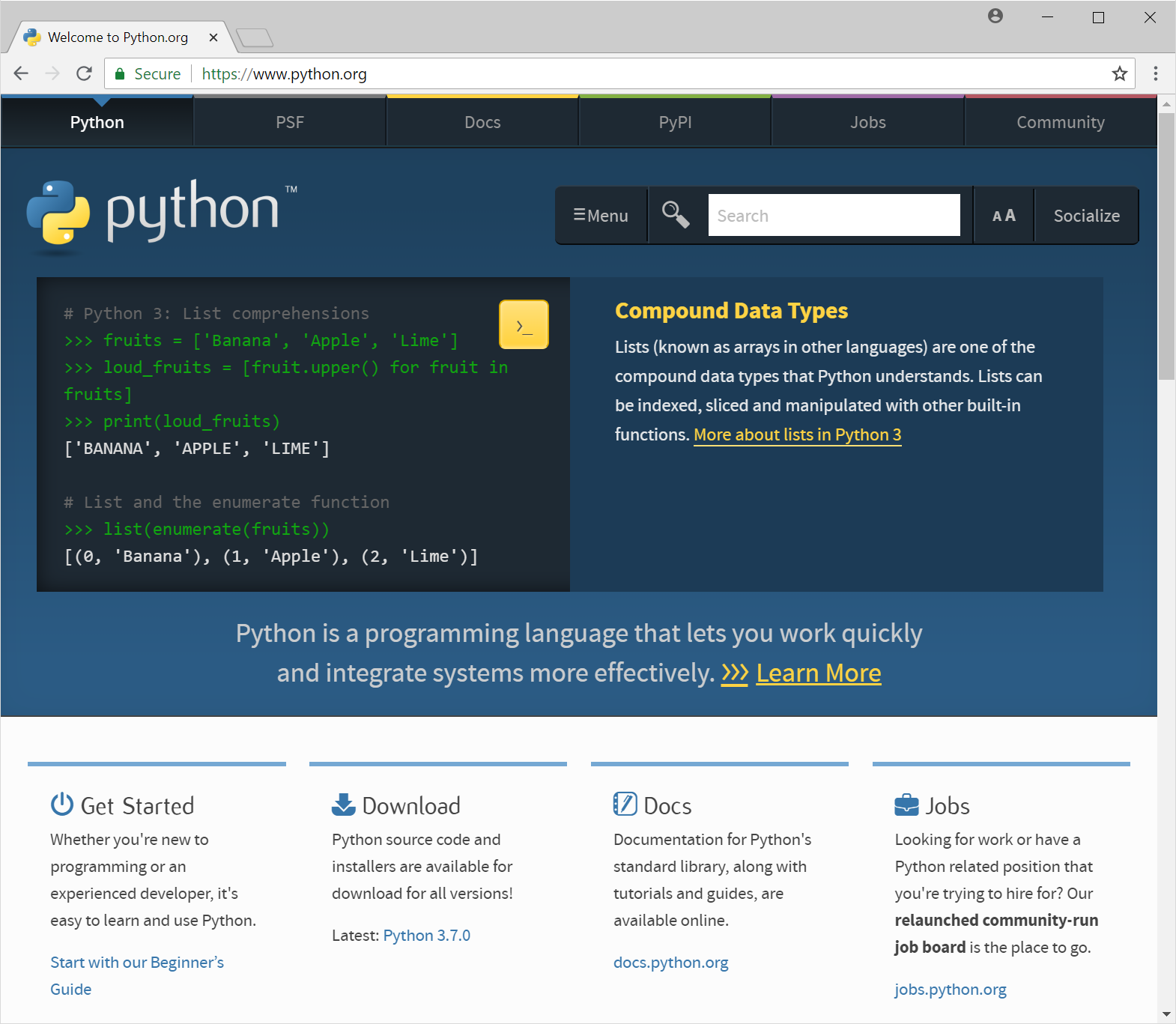
<https://zoom.us/j/2821636776>

1. If you are new to Zoom:
2. After download completes, click Zoom\_Launcher.exe in lower left corner of screen.
3. If you see "Do you want to run this file? …Zoom Launcher…" click the "Run" button.
4. To the prompt: "Please Enter Your Name", enter your name, then click **Join.**
5. If you are in the same room as the speaker, please mute your microphone by clicking the microphone icon in the lower left corner of the Zoom window.

## Using PythonAnywhere

## Online Python Resources

### python.org <https://www.python.org>



Download Latest: Python 3.7.0 – Click Python 3.7.0 in bottom middle.

A more detailed description of the installation is found in my document pythonorg.docx.

### PythonAnywhere

<https://www.pythonanywhere.com>

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

### Other Online Python Resources

### 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.