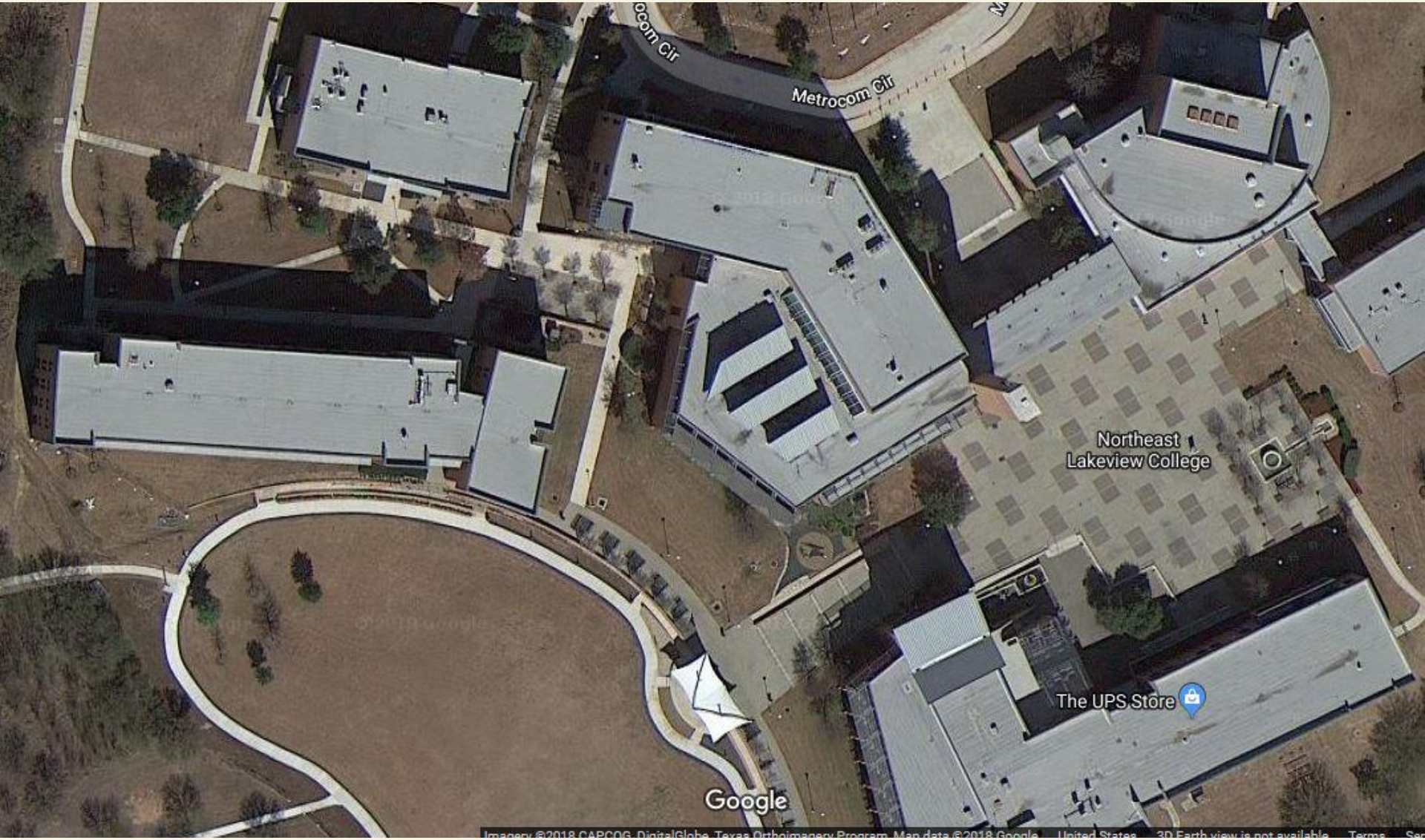
# Python I
# Welcome to Programming

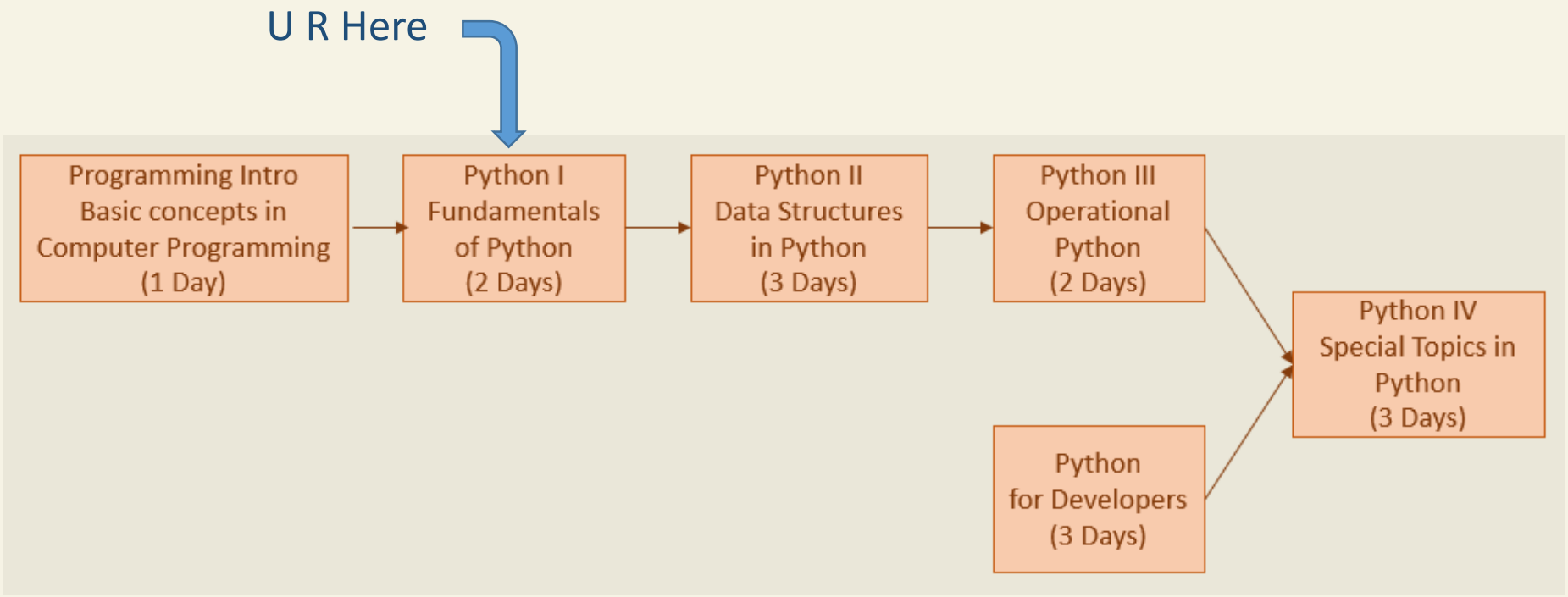

Rev: 1/5/2020

# NLC INFO

- Speeding on campus = easy ticket
- Must go off property to smoke
- Please sign in each day. Class starts at 08:30
- PC's are Windows 10 with Python 3.7 installed
  - Idle is available as an aid in entering and executing programs.
- Class data is available via the cloud.
- Completed labs will be available via the cloud.
- Lab computers are reset every evening.  Save your data!


Metrocom Cir
Northeast
Lakeview College
The UPS Store
Google
Imagery ©2018 CAPCOG, DigitalGlobe, Texas Orthoimagery Program, Map data ©2018 Google
United States
3D Earth view is not available
Terms

# Python Classes




Objective:

- 30% Lecture
- 70% Lab
- Using Version 3.x of Python

# Introductions

- Please mention:
  - First name
  - What company you are with and what your job is.
  - Any programming or scripting experience you may have.
  - What you hope to get out of the class?

# PAPERWORK

- Grant registration
  - Fill in at least the fields for gender, birth date and your name at the top and the signature line at the bottom. The rest is optional, but the ACCD would appreciate having it.
- Grant daily sign-in and sign-out sheet
- If you have questions, my e-mail is:
  - weastridge@gmail.com
  - Make the subject: Python

# Agenda

Review for background purposes

Define / Set up lab environment

Basics of sequence, selection and iteration

Functions and code reuse

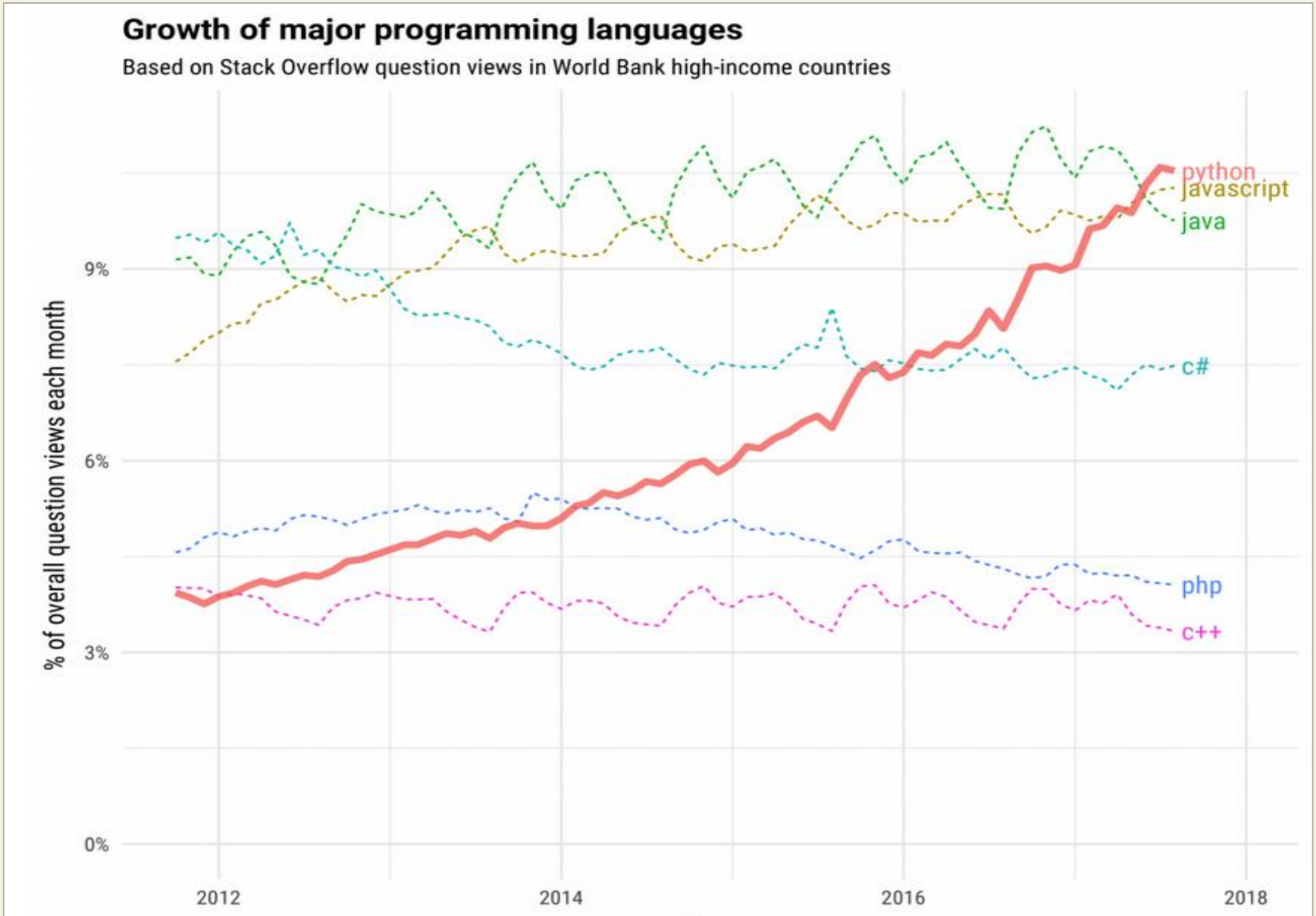
Intercepting errors

Reading and writing files

# Resources

- Two books in PDF format oriented to Python version 3:
  - Python for Everybody – this is a practical book for beginners, but in chapter 11 he transitions to some more advanced material.
  - Think Python – a larger somewhat more comprehensive book with more difficult exercises.
- Python Notes – collection of useful information in one place.
- A copy of the slides used in this class.
- A set of screen shots showing samples of basic Python code.
- Data to be used in lab exercises.
- Completed labs will be posted on the cloud after they are reviewed.

# Why Python?


Growth of major programming languages
Based on Stack Overflow question views in World Bank high-income countries
% of overall question views each month
9%
6%
3%
0%
2012
2014
2016
2018
python
javascript
java
c#
php
c++

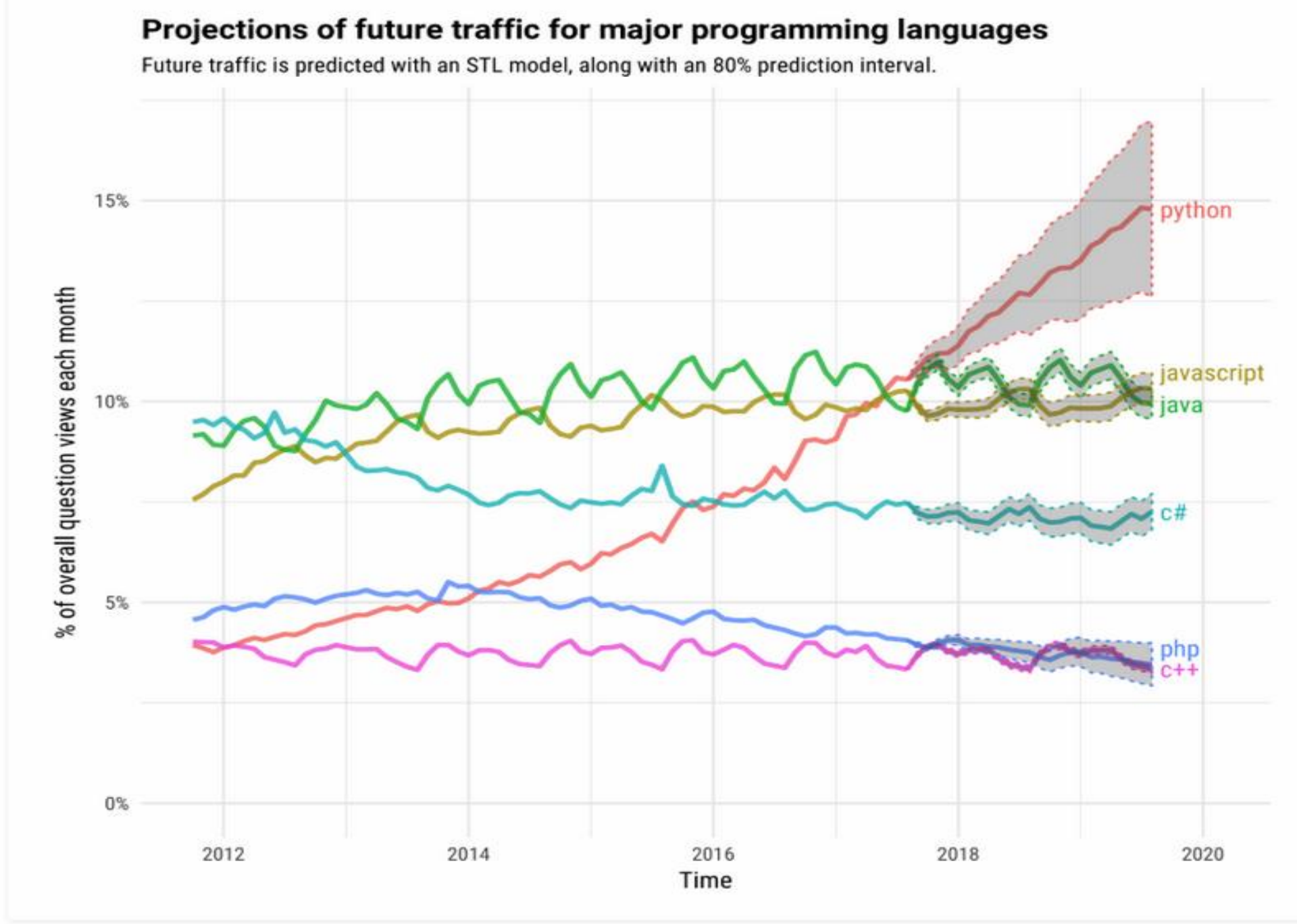
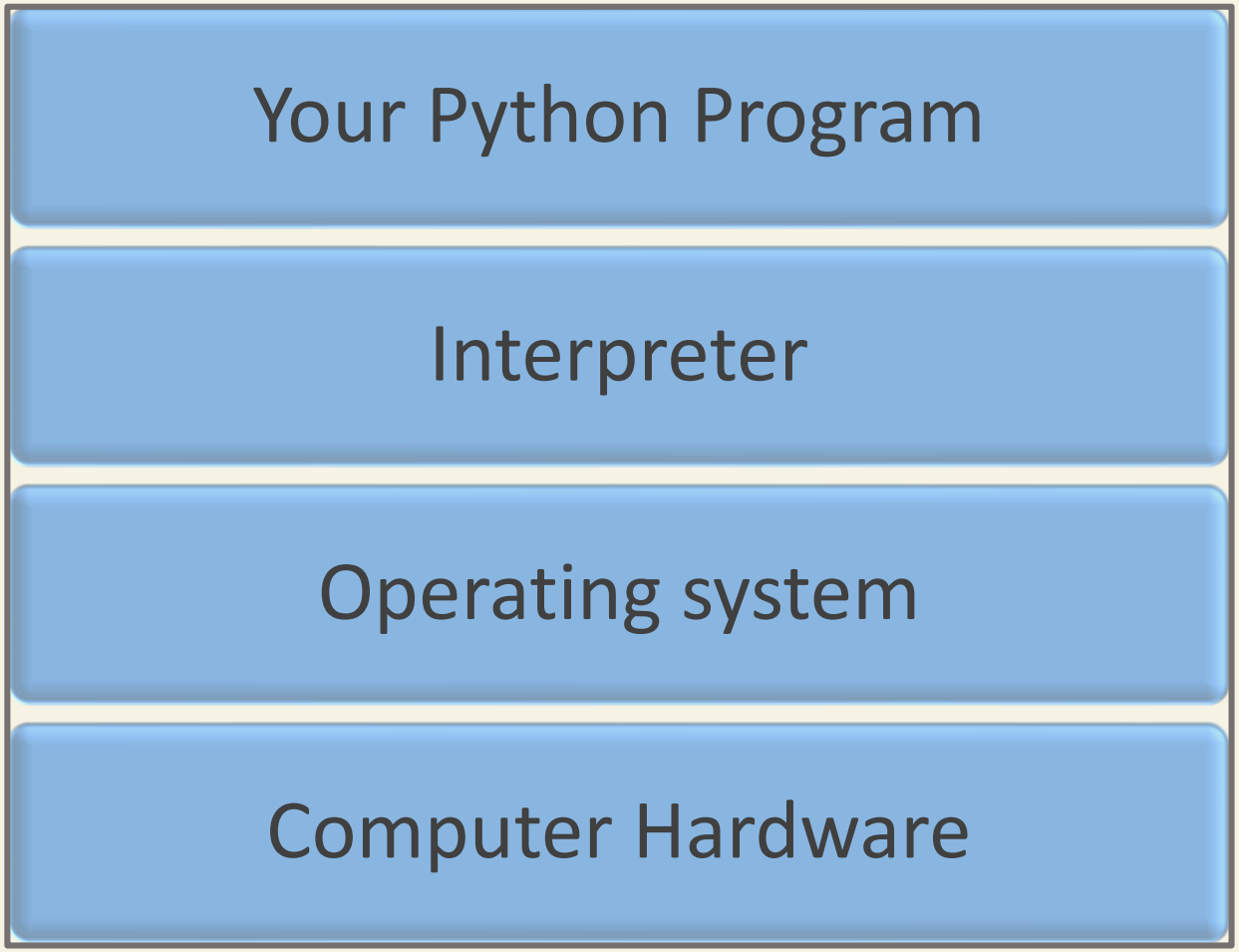
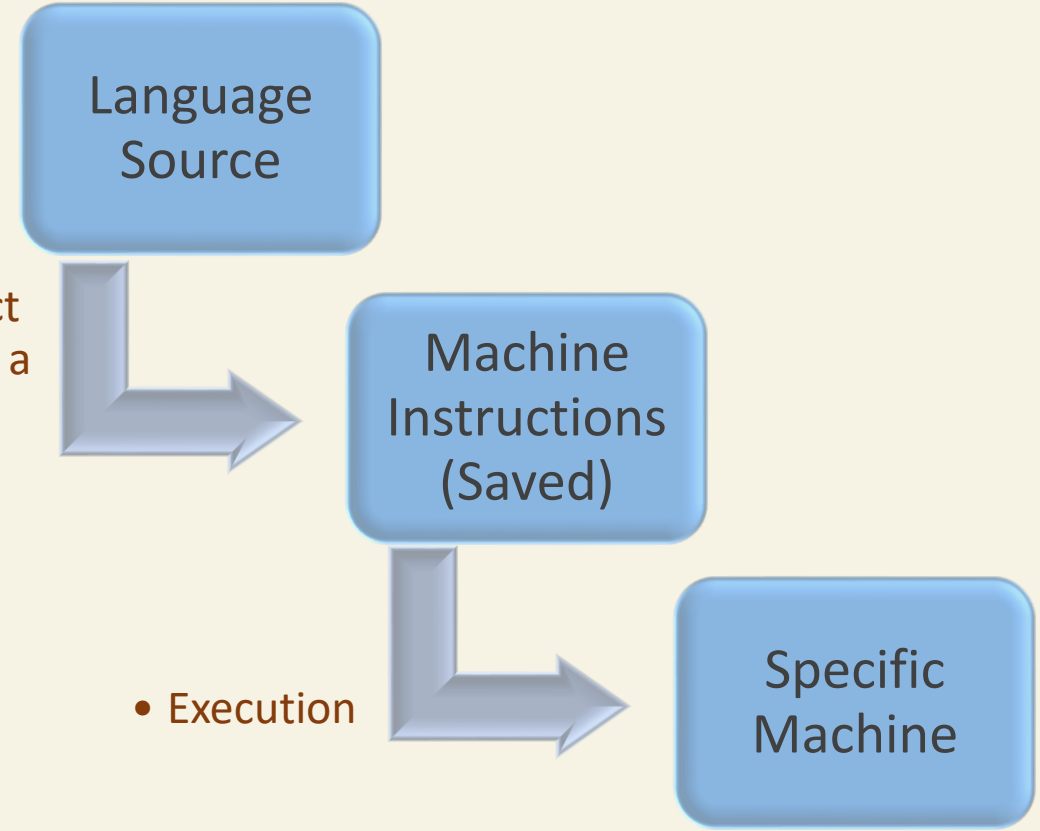



*Image Source : Stackoverflow*

# THE NOTIONAL MACHINE



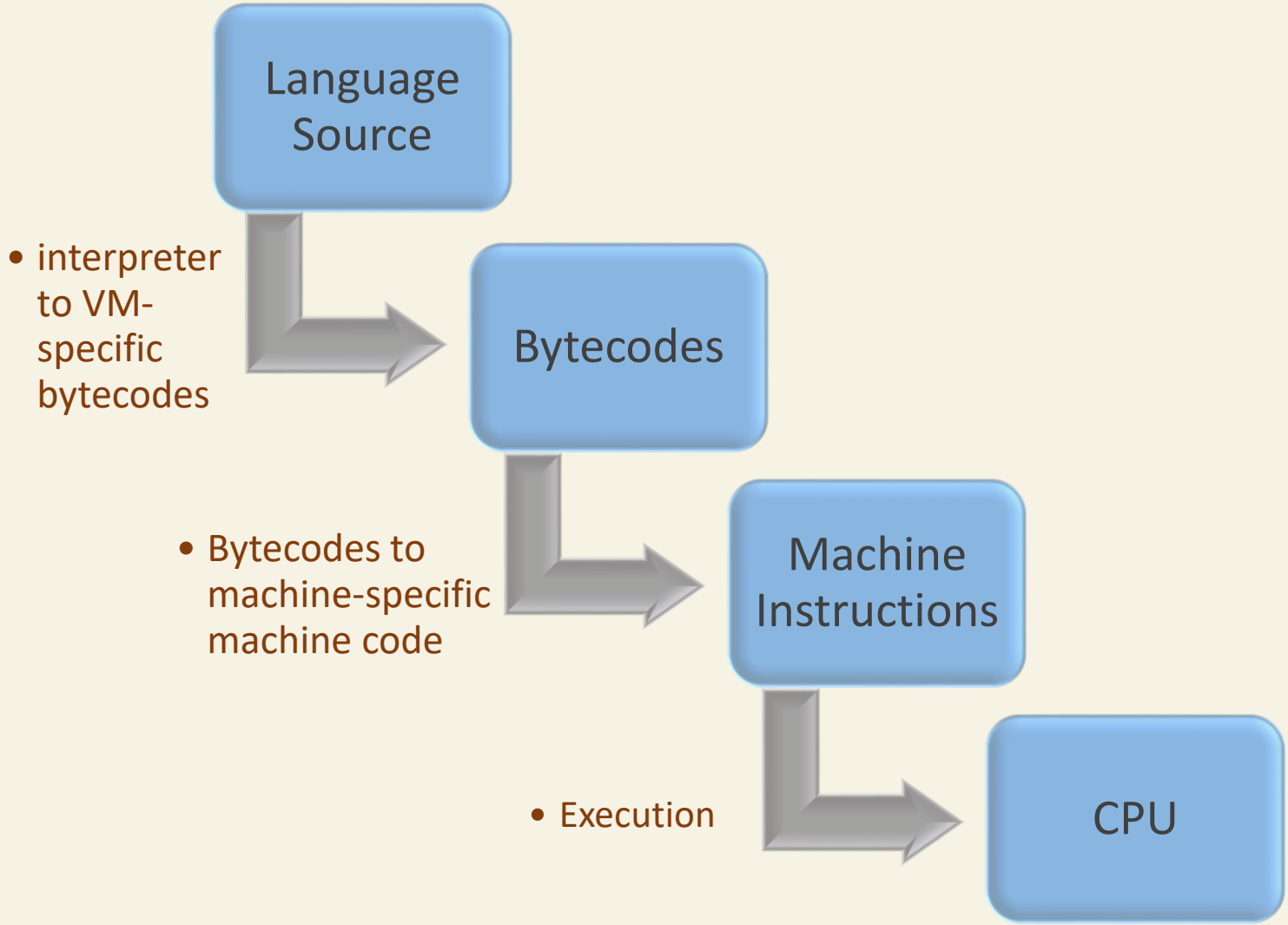
# Translating The Source (Compiling to Object Code)




- Produce object code to run in a specific environment
- Execution

Disadvantage – this code can run only in a specific environment.
Advantage – the code will run faster / more efficiently.

# Translating The Source (Interpreting)



# Introduction to Programming

- Objective - Learn the basics of programming using the Python language.
  - Has its own set of rules.
  - Basic principles are the same as any other language.
- Recognize that a programming language is actually a language to be mastered.
  - As with any language, it requires constant use and study to attain fluency and fluency is the goal.
  - Taking a class is only a start.

# Semantics

Bit – Binary digit - the smallest unit of computer storage.

Byte – Grouping of 8 bits.  Can store a character (e.g., letter a)

Binary – Numbering system used by computers to do math.

- Example – decimal number 147 in binary is 10010011
- Numbers used in math usually occupy multiple bytes.

There are two types of numbers in a computer:

- Integers and numbers with decimal points (called floats)

ASCII – A system for representing human-readable characters in a computer.  See chart in Python Notes

In programming:

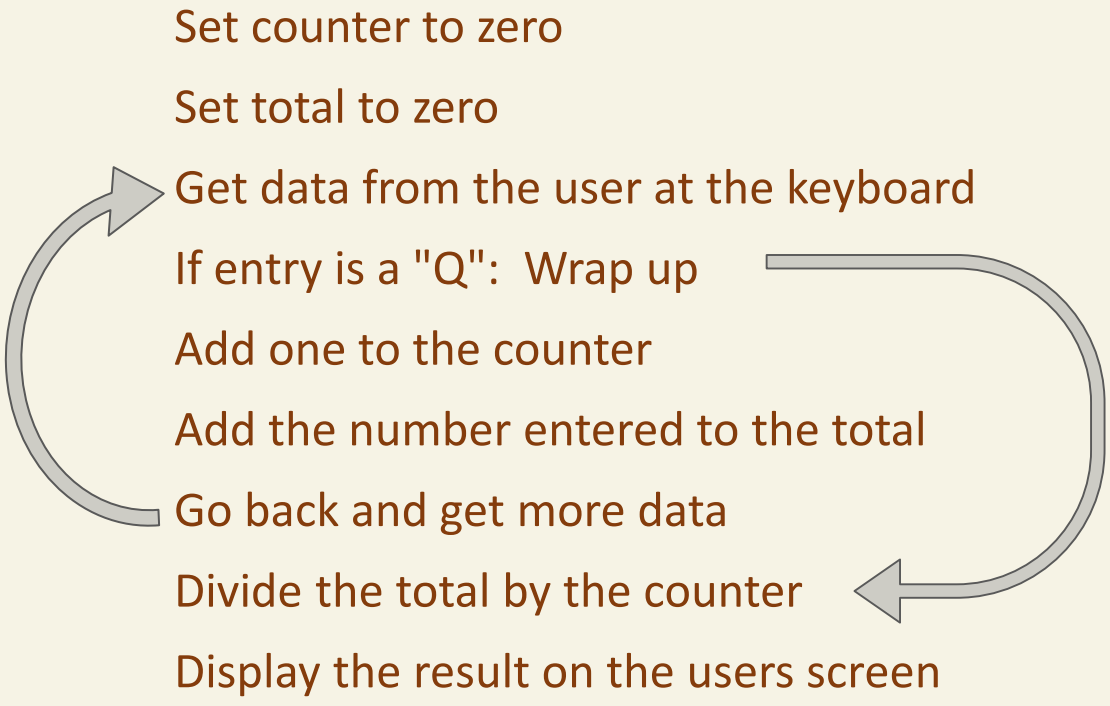
- The apostrophe (') is called a quote or single quote.
- The quote (") is called a double quote.

# Planning

- Planning is essential to writing a program.
- Always understand how you are going to approach a problem.
- For example:
  - What calculations will you have to perform?
  - What data will you need?
  - Where will it come from?
  - How will you preserve/save it?
  - What will you do with the results?
- These are just a few of the questions you need to answer.
- As the class problems become more complex, planning becomes essential.

# Onboard Lab Solution

I want to get numbers from the keyboard and take an average when the entry is a "Q"



# From Concept to Code

- Create a plan.
- Determine how to go from your plan to actual code.
- Always have a working program.
- Resolve problems as you encounter them.
  - Review the reference material you get in this class.
  - The web is loaded with information (much of it confusing to beginners).
  - Be sure to do everything you can before asking your team members.
  - ASK AS MANY QUESTIONS AS YOU WANT IN CLASS.

# Our Lab Environment

- Python 3.4 or above (aka CPython)
- IDLE
  - Rudimentary IDE
  - Really primarily a learning tool allowing editing and executing of Python programs.
  - Shell operations vs program creation windows
- Command line / Vim
- Do "Hello world" both ways

# LAB
# Programming Samples - Exercise 1

In your data you will find the document Programming Samples.pdf. Do the above exercise in that document. It demonstrates a number of programming features we will discuss in more detail as the class progresses. The primary purpose here is to expose you to basic Python code and Python interpreter error messages. That assumes you will make some errors entering the program statements.

# Identifiers/Variables

- In a program you can save data by giving it a name. Such data is called a variable since it can be changed.
- A variable is one type of identifier. We will see others later.
- The variable points to the memory location where the data is stored.
- All variable names:
  - Must start with a letter or underscore
  - Are case sensitive
  - Can only have letters, numbers, and underscores
  - Can be any length, so use a descriptive name
  - Cannot be a reserved word. See Python Notes for a list of reserved words.
- Variables are created with assignment statements.

# Assignment Statements

- Exactly what is an assignment statement? Assignment statements take the following form:

  variable = expression

- Here, the expression can contain literals, variables, functions and math operators
  - x = 10 (a simple assignment)
  - x = x + 1 (The original value of x is incremented by 1)
  - x = sqrt(y) + z / 15 (a more complex assignment)
- All assignments have these attributes:
  - If the variable on the left side of the equal sign does not exist, it is created, initialized with a value and assigned a type.
  - If it already exists, the value in it is replaced with the value provided by the expression and the type updated if necessary.
  - Nothing to the right of the equal sign (the expression) is changed.
- See a0Assignments.jpg in Samples

# Variables & Type

- In Python, the only way to define a  variable is to initialize it.
- This can be done with an assignment statement
- Once initialized a variable has a type and a value.

  x = 10  x is an integer (int)

  y = 13.7  y is a decimal number (float)

  z = 'Some text'  z is a string (str).

  Strings are enclosed in single or double quotes

- Unlike other languages, you can freely mix integers and floats in calculations and get the correct answers (Python V3 only).

# Basic Data Types

integer

float

string

boolean

None

# Math Operators

| | |
|---|---|
| ** | Exponentiation |
| / | Division (works differently in Python 3) |
| * | Multiplication |
| + | Addition |
| - | Subtraction |
| % | Modulus (Remainder) |
| // | Integer/Floor Division |

**Do you have to worry about mixing integers and floats in a math operation?**
**No! Not in Python V3!**

# Printing

- Printing is done using the print function.
- The general format of a function is as follows:
  - funcname(arg1, arg2, … , argN)
- The print function is used to output one or more items. These can be literals, variables or expressions
  - Each argument is separated by one space. The sep= key word can override this default.
  - Each use of a print function causes the next printing to be done on a new line. The end= keyword can override this default.
  - A print function with no arguments just forces a new line. i.e, print()
- See the sample a0Print.jpg in the Sample folder.

# LAB
# Programming Samples - Exercise 2

In your data you will find the document Programming Samples.pdf. Enter and execute the program shown in the above exercise. It demonstrates a number of features related to printing. It includes ways to enclose a string. Be sure to read all of the text associated with the code shown in the exercise. Pay special attention to the definition of whitespace.

# Expressions

- An expression is a combination of one or more variables, operators, literals, and/or functions.
- Expressions compute something.
- Expressions can occur on the righthand side of the '=', as arguments, and many other places.
- Boolean expressions are either True or False.
- In Python, it is considered proper style to leave a space on either side of an equal sign and most math operators. If you don't, the interpreter will not generate an error. Use your judgement as to what makes your code most readable.
  - This topic is covered in detail in PEP8.

# Math Operators

| | |
|---|---|
| ** | Exponentiation |
| /,//,% | Division |
| * | Multiplication |
| + | Addition |
| - | Subtraction |

**Order of Execution (Precedence)**

- Exponentiation
- Multiplication/Division (left to right)
- Addition/Subtraction (left to right)

Only parentheses change the order of execution.

# Lab

**Order of  Execution (Precedence)**

Exponentiation

Multiplication/Division (left to right)

Addition/Subtraction (left to right)

Only parentheses change the order of execution.

Operators: + addition, - subtraction, * multiplication, / division, ** exponentiation

What results would these formulas produce?

10 + 8 / 2 * 6 - 4 = ?
(10 + 8) / 2 * 6 - 4 = ?
10 + 8 / 2 * (6 - 4) = ?
10 + 8 / 2 * 6 - 4**2 = ?
Note - spaces omitted in 4**2 as a matter of personal preference

# Completing Labs

- We will be doing a lot of labs that you have to do from scratch.
- Many people going through this class are unable to complete all of these labs.
- Actually completing the lab is not as critical as making a serious effort to do so.
  - This will make the explanation of the completed lab make more sense to you.
- An inability to finish the labs is no reason to become discouraged. This happens to most people going through this class.

# Lab01 - Formulas

- Create a program to solve these problems and print the results. Place each value specified by the problem into a variable before doing any calculations. Do not worry about the formatting of the answers.
  - You bought 125 shares of a stock at $25.32 and you sold it at $48.97. What is the profit?
  - A product with a price of $127.99 is going on sale. If the price is reduced by 16%, what will the new price be?
  - Answer:

```
Profit is 2956.25
Sale price is 107.51159999999999
```

# Syntax, Semantics and Failure

- There are three categories of problems you can encounter when writing and running a program.
  - You may have experienced some of these already
- Syntax – basic structure (e.g., missing punctuation, misspelled statement)
- Semantics – you misuse a capability in a way that Python accepts as valid. (e.g. x = 1,000,000)
- Failure – your program fails when you run it. You are provided with an error type (e.g., ValueError). You have to determine why the problem occurred.

# Formatting – Newer Style

- In the Introductory classes, we previously used the old-style formatting (%).
- Many other sites still use the old style.
- You will see both the older  and newer styles used in Stack Overflow as well.
- It is important to recognize and know both.  They are both covered in Python Notes.  Most completed labs show how to use both.
- The next several slides and the labs in the rest of this class use the newer style.
- Python 3.6 introduced yet another formatting capability (newest?).
  - It is called f-strings and is also covered in Python Notes.

# Formatting Data into Strings

General example:

'insert text here with {0} {1}'.format(variable, "literal string")

Abbreviated general format of a formatting sequence:
{[seq#] ":" [width] ["," ] ["." prec] [type]} - See Python Notes for more detail

- The sequence number [seq#] is optional. If missing, variables/literals are formatted in the order given (python 2.7+).
- Width is used to expand a formatted item beyond the default. In the expanded width, numbers are right justified, text left justified.
- The ',' is used as a thousands separator in larger numbers.
- The ".prec" specifies the number of decimal places to display.
- The short list of valid types are s, d and f .
  - s – strings, d – integers, f – floating-point numbers

# Formatting Data into Strings

Abbreviated general format: {[seq#] ":" [width] ["," ] ["." prec] [type]}

Examples:

a = 12

b = 17.426

x = 'Some text {} more text {}'.format(a, b)

Result stored as x – 'Some text 12 more text 17.426'

x = 'Some text {0} more text {1}'.format(a, b)

Result stored as x – 'Some text 12 more text 17.426 '

x = 'Some text {1} more text {0}'.format(a, b)

Result stored as x – 'Some text 17.426 more text 12 '

x = 'Some text {0} more text {1:.2f}'.format(a, b)

Result stored as x – 'Some text 12 more text 17.43' (Note rounding)

See examples in Sample folder - a2Formats.jpg and a3Formats.jpg

# Formatting Data into Strings

Abbreviated general format: {[seq#] ":" [width] ["," ] ["." prec] [type]}

General examples:

a = 1234567.889

x = 'I would like to have ${}'.format(a)

Result stored as x – 'I would like to have $1234567.889'

x = 'I would like to have ${0:,.2f}'.format(a)

Result stored as x – 'I would like to have $1,234,567.89 ' (Add separators – note rounding)

x = 'I would like to have ${0:15,.2f}'.format(a)

Result stored as x – 'I would like to have $ 1,234,567.89'

When the width specified is larger than necessary to accommodate the result, numbers are right justified and everything else is left justified.

# Lab02 - Formulas

Redo the last lab formatting the output.

Sample output:

```
Profit is $2,956.25

Sale price is $107.51
```

# Strings vs Numbers

- So far, the numbers we have assigned to a variable were converted to internal format for us.
  - x = 10  stores in memory a binary 10 as an integer.
  - Whenever we refer to x, Python retrieves the current value for us. e.g, x = x + 1 retrieves the value of x, adds 1 to it and stores it again.
- A string cannot be used in a mathematical operation even if it contains all numbers.
  - x = "10" stores in memory a string 10 as two characters.
  - x = x + 1  results in an error message from the interpreter.
- Numbers entered from the keyboard or from text files stored on a hard drive have to be converted to internal format before being used in mathematical operations.

# Getting Keyboard Input

- The input() function
  - Gets data from the keyboard in the form of a string stripping the newline character.
- strings vs. floats and integers
  - Numbers entered from the keyboard need to be converted before you can use them in mathematical formulas
- float() and int() functions
  - These two functions convert strings to numbers (integers or floats) if all the characters are valid. Int() requires all digits. float() has the same requirement and allows a decimal point. Both functions strip leading and trailing white space.
- See bInput-Float-Int.jpg in Samples

# LAB 03

- In your data you will find the document Programming Samples.pdf. Do Exercise 3 in that document by entering and executing the program shown.
- Create a program that reads a temperature in degrees Fahrenheit from the keyboard, then prints the equivalent Centigrade temp.
- Remember:
  Fahrenheit = 9/5*Centigrade + 32
  Centigrade = 5/9*(Fahrenheit – 32)
- Example:

```
Enter a temperature: 72
72.0 degrees Fahrenheit is 22.2 degrees Centigrade
```

Note – Make sure you keep all copies of your programs from now on. You will be re-using this code often. My copies will be available through the cloud.

# Selection

- Prior examples and exercises used only sequence; executing one instruction after the other until the program is finished.
- Selection allows us to selectively execute instructions based on a condition.
  - The decision is based upon the result of a comparison (true or false)
- A single-branch selection allows you to excecute or skip a set of instructions based on a condition.
- A multiple-branch selection allows you to select a particular group of statements to execute.

# If … (Single Branch Selection)


The instructions given consider situations other than the basic one.

"if you are travelling with a child, secure your mask first, then secure the mask of your child.

## Putting on an airline oxygen mask

1. secure your mask
2. if you are travelling with a child then
3. secure the mask of the child
4. sit back

**Sequence– with child**

**1->2->3->4**

**Sequence – without child**

**1->2->4**

# Dual-Branch Selection

## Reading a book

If I have an unread book on my bookshelf:
  Pick a book that you have not read.
  Read the book.
  Put the book back
Otherwise:
  Go out and buy a new book.
  Read it.
  Put it on the bookshelf.

One of these sets of instructions will be followed but not both.

# Making Decisions

- All languages make decisions, usually  with an if statement.
- if or if/else.
- Each option (if, else) is followed by a suite of statements
  - At most, only one suite is executed
- For a simple if statement (single branch), the suite of statements following it are executed if the comparison is true.  Otherwise, they are skipped.
- For if/else (dual branch), one and only one suite will be executed. In any case, something will be executed.

# Comparison Operators

- == Equal a == b
- != Not Equal a != b
- > Greater Than a > b
- < Less Than a < b
- >= Greater Than or Equal a >= b
- <= Less Than or Equal a <= b
- not, and, or
  - compound condition examples:
    if x >= 0 and x <= 10:
    if x == 1 or x == 2:  Note how the variable name is repeated
    if not(x == 1 or x == 2):

# Python Indentation

- The way Python uses indentation is unique
- Indentation shows where blocks/suites of code begin and end.
- The standard is to use 4 spaces per indentation level (not tabs).
- IDLE will do this for you automatically
  - If it gets confused, it's probably because of something you have done.
  - It replaces tabs with 4 spaces.  Multi-line indent/unindent capability.
- Other editors (e.g., Notepad+, Bluefish) can be configured to replace tabs with spaces.
- Demo with sample : (c1If-Else.jpg)

# Lab 04

Create a new program from the previous lab to print out a description along with the centigrade temperature. For Fahrenheit temperatures that are above 90 degrees, print “It’s hot outside”. Otherwise, print “It’s not hot outside”. This printing should follow the printing showing the conversion from Fahrenheit to Centigrade.

Example:

```
Enter a temperature: 72
72.0 degrees Fahrenheit is 22.2 degrees Centigrade
It's not hot outside
```

# Multi-Branch Selection

## Reading a book

If I have an unread book on my bookshelf:
    Pick a book that you have not read.
    Read the book.
    Put the book back
Else if I have enough money:
    Go out and buy a new book.
    Read it.
    Put it on the bookshelf.
Else:
    Get a book from the library.
    Return it before you incur fines.

Only one of these sets of instructions will be executed.

# Making Decisions

- The if statement has an expanded option for decision making.
- if or if/else or if/elif/else. (Use elif for more than two options)
- Each option (if, elif, else) is followed by a suite of one or more statements
  - At most, only one suite is executed
- There can be multiple elif statements.
- Regardless of the number of elif statements, one (and only one) suite will be executed.
- Demo with samples. : (c2If-Else.jpg)

General Form:

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
      :
      :
      :
elif expressionN:
    statement(s)
else:
    statement(s)
```

# Lab 05

Create a new program from the previous lab to print out a description along with the centigrade temperature. For Fahrenheit temperatures in the following ranges, print the corresponding description along with the centigrade conversion:

Temperature:

| | |
|---|---|
| Over 95 | It's very hot! |
| Over 80 to 95 | It's hot |
| Over 60 to 80 | It's nice out |
| Over 40 to 60 | It's chilly |
| 40 or less | It's cold! |

Example: keyboard entry is 72. Program output should be something like:

```
Enter a temperature: 72
72.0 degrees Fahrenheit is 22.2 degrees Centigrade
It's nice out.
```

This exercise requires multiple elif statements following the if statement.

# Play it Again Sam (Conditional Iteration)

## Reading all books on my bookshelf

Repeat while books on shelf I have not read:
  Pick a book that you have not read.
  Read the book.
  Put the book back
Declare "I have read everything!"

# Looping Until a Condition is Met

- One way to loop (or iterate): while statement
- Executes a set of statements in a loop:
  - until the while expression turns False, at which time the loop terminates, or
  - until an instruction in the loop explicitly terminates the loop
- The while statement specifies a condition or True
- If True is used, somewhere in the loop itself, a break statement must be used to terminate the loop.
  - This technique is employed when a condition cannot be defined in the while statement. See d1While.jpg in Samples.
  - Break statements work only in the loop in which they are executed.

# LAB 06a

Re-implement the previous lab asking the user for temperatures to convert in a loop.
Stop if input() returns 'q' or 'Q'

Sample output:

```
Enter a temperature: 72
72.0 degrees Fahrenheit is 22.2 degrees Centigrade
It's nice out.
Enter a temperature: 32
32.0 degrees Fahrenheit is 0.0 degrees Centigrade
It's cold!
Enter a temperature: q
Conversions ended
```

# Loops Within Loops

- Loops can contain other loops just as if statements can contain other if statements. (Sample c1If-Else2)
- In this example, we will use a condition rather than True.

```
x = 0
while x < 5:
    y = 0
    while y < 3:
        print(x, y)
        y += 1
    x += 1
```

Results

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
4 0
4 1
4 2
```

# LAB 06b

- Using the technique described on the previous slide, create a times table.
- It should be a matrix of the results of all possible single-digit multiplications.

Sample output:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 |

- Don’t worry about getting numbers aligned unless you finish early.
- If this assignment confuses you, just do one loop and print the value of one variable as it increases. Then, add another loop within the one that is working. Remember, always have a working program.

# Importing Modules

- Much of Python's capability is included in modules.
- You import only what you need.  There are a large number of modules.
- Use help to show modules. Interactive shell – help('modules') or pydoc3 modules from command line. python –m pydoc (Windows)
- Some modules contain code written in C++ (e.g., math).  That code is compiled as an executable and cannot be inspected.
- Many modules contain code written in Python.  This code can be inspected just like one of your own programs.

# Importing Modules

- There are several ways to get a module or part of a module into your program.
  - import random  (imports the entire random module)
    In this case you have to use dot notation to get to the function you want. (e.g, x = random.randrange(1, 10))
  - from random import randrange
    No need for dot notation  (e.g., x = randrange(1, 10))
  - from random import randrange as rr:
    Use a smaller name for less typing (e.g., x = rr(1,10)
  - Importing variables works the same (e.g., from string import punctuation)
- We will be using the randrange function in the random module.
  - This function picks a random integer in the range you specify excluding the end point.
- Review hImports.jpg in Samples

# LAB 07 - Game

Create a game. Have the computer select a random integer between 1 and 100 inclusive. Then, have the operator take successive guesses until they guess the correct number. At each try, advise them whether the guess is too high or too low. If the guess is correct, tell them they won and tell them the number of attempts they took. Use the same format as the output sample to the right. This exercise requires you to use everything we have learned so far.

```
Enter your guess: 50
Your guess is too high
Enter your guess: 25
Your guess is too low
Enter your guess: 37
Your guess is too high
Enter your guess: 31
Your guess is too high
Enter your guess: 28
Your guess is correct!
You succeeded in 5 attempts.
```

# Functions

- What Python functions have you used already? How did you access them?
- You can build your own functions.
  - def/return statements. The return statement is implied if omitted.
- Functions are just named groups of statements that return an answer.
- The name you give a function follows the same rules as variables. See PEP8 for standards.
  - This is the second type of identifier we have seen. The first was a variable. We will see more in future Python classes.
- Functions are 'called' with arguments (not the verbal kind).
- Functions reduce repetitive code and simplify the main program.

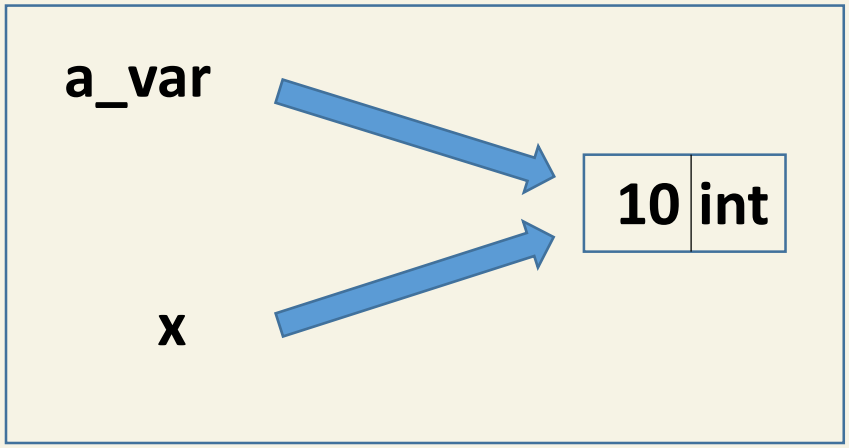
# Functions vs Main Programs

- You should consider functions and main programs separate programs.
- A function can "see" the variables in the main program.
  - It should not use them unless they were explicitly passed to the function.
  - Under no circumstances should the function try to change a variable in the main program even if the variable were passed to the function.
- In the function, variables passed to the function may have any valid variable name.
  - Assume that functions and main programs are being written by different people.
  - The  names between the main program and the function do not have to match.  Regardless, they point to the same object. (Next slide)
- Sample - f1Functions.jpg - Note where the main program begins in Python versus other languages.

# Functions vs Main Programs

```
def testfunc(a_var):
    return a_var**2

x = 10
z = testfunc(x)
print(z)
```




**Think about a built-in function such as float(x). Do we care what variable the float function uses to receive the argument?**
**See testfunc.py in DemoProgs**

# LAB 08a

Re-implement a previous lab to use a function to do just the temperature conversion formula.  Call the function from the main program, and return the centigrade temperature from the function.

# LAB 08b

Create a program that reads a temperature from the keyboard. It should then read a second input, a single character, that determines what type of conversion to perform. A 'c' causes a fahrenheit-to-centigrade conversion while 'f' causes the opposite conversion. Create separate functions to provide the conversions as well as print statements showing the result of the conversion. The functions should be void (return None). After a conversion, the main program should request a new set of inputs from the keyboard until a 'q' is entered in place of a temperature. Also, remember:

Fahrenheit = 9/5*Centigrade + 32

Centigrade = 5/9*(Fahrenheit – 32)

See plan on the next slide

Sample Output:

```
Enter a temperature: 72
Enter a c or f: c
72 degrees Fahrenheit is 22.2 degrees Centigrade
Enter a temperature: 32
Enter a c or f: f
32 degrees Centigrade is 89.6 degrees Fahrenheit
Enter a temperature: q
```

# LAB 08b Plan

**Main Program**

Read a temperature from the keyboard and save it.

Read a control character form the keyboard.  It must be a 'c' or 'f'.

If the control is a 'c', call the function that converts the temperature to centigrade and prints the result.

If the control is a 'f', call the function that converts the temperature to fahrenheit and prints the result.

Keep getting keyboard inputs until a 'q' is entered in place of a temperature.

**Functions**

There should be two functions; one that converts the number to centigrade and one that converts the number to fahrenheit.

Each function applies the appropriate formula depending on the control character.

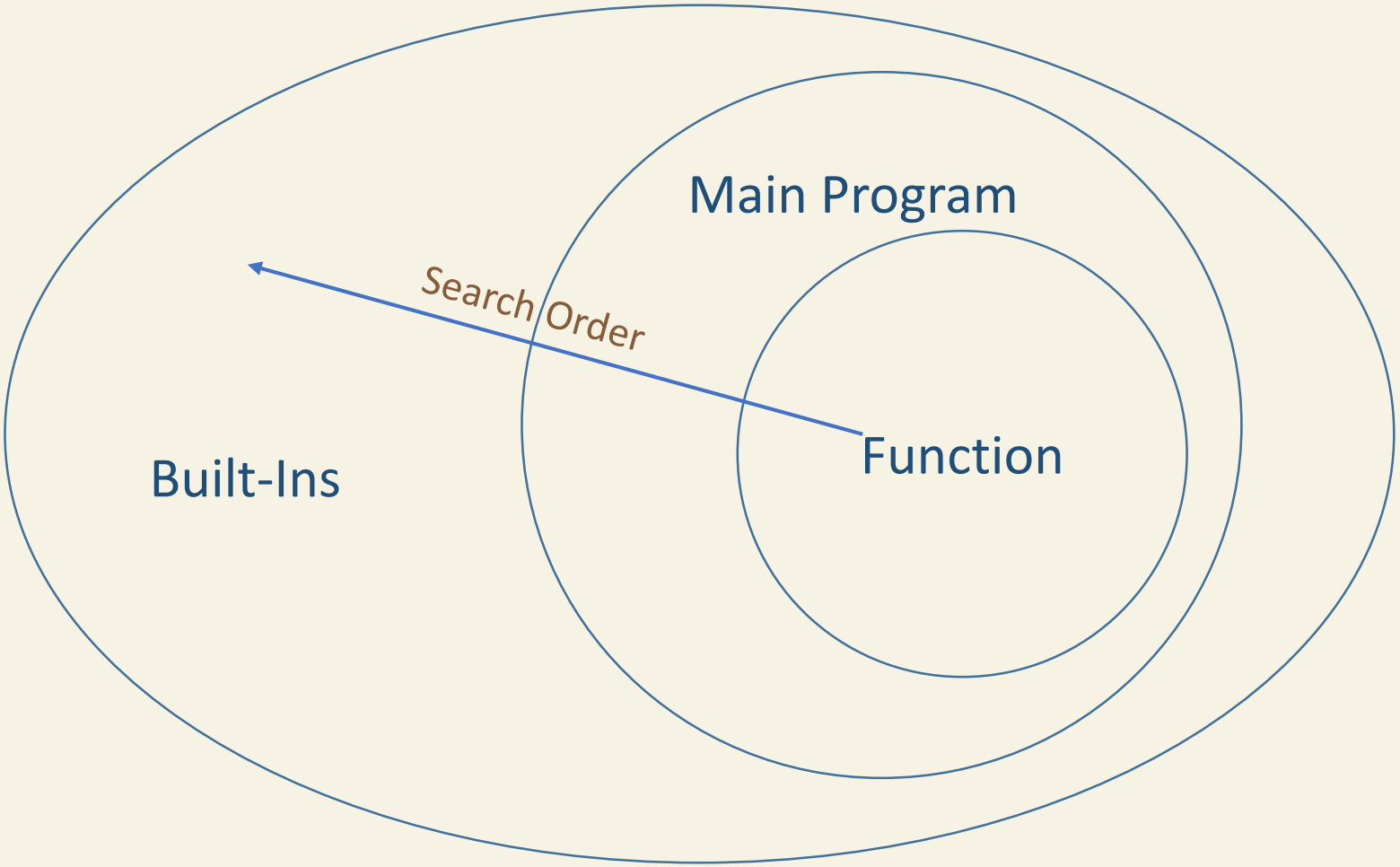
Each function prints the results of the conversion and returns None to the main program.

The formulas to be used are:
Fahrenheit = 9/5*Centigrade + 32
Centigrade = 5/9*(Fahrenheit – 32)

# IDENTIFIER SCOPE

- Scope is how widely the identifier can be seen in other parts of the code
- Local scope
- Global scope
- 'Built-in' scope
- Sample - f2Functions global-local.jpg

# SCOPE (Simplified)



# EXCEPTIONS

- Exceptions are just events: e.g., using an incorrect data type for an operation or using a variable that isn't defined.
- Handling an Exception is optional.
- Catching Exceptions requires using a TRY statement
- Some Exceptions: NameError, ValueError, TypeError.

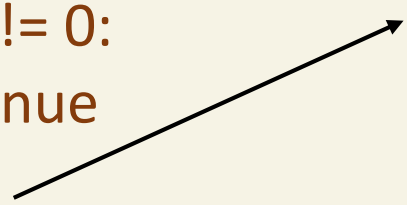
# CATCHING EXCEPTIONS

- **try/except/else/finally**
- Puts a protection ring around a suite of code
- **except** catches one or more named Exceptions
- Don't use 'bare' **except** unless there is a compelling reason**.**
- **else** runs if no Exception occurs.
  - Without **else,** the main program just continues to execute.
- **finally** always runs no matter what
- Sample - gTry_Except.jpg

# Short Circuit a Loop

- The break statement gets you out of a loop.
- What if you just want to short circuit the loop by starting the next iteration prematurely?
- Example - print only the even number in a series:

```
x = 0
while x < 10:
    x += 1
    if x % 2 != 0:
        continue
    print(x)
```



```
2
4
6
8
10
```

- This technique is particularly helpful when you are accepting entries from the keyboard and you get an erroneous entry.

# LAB 09

Re-implement a previous lab to use a try statement to catch ValueError exceptions. Test by entering a non-numeric temperature.

# Play it Again Sam (Fixed Iteration)


At school as a punishment you might have to write lines: writing the same thing over and over again.

"Write out 100 times, 'I must not throw chewing gum at the teacher' "

"Write out 30 times, 'I must not break up my desk and pass the bits out of the window behind the teacher's back' ",

## Writing Lines

1. Repeat 100 times:
2. write punishment line
3. swear silently

**With fixed iteration we know in advance the number of times an instruction needs to be repeated**

# Looping N Times

- The for statement is another way to iterate.
- General form:  for *some_variable* in *some_collection:*
  - Each element in *some_collection* becomes available in *some_variable.*
  - There is a new element for each iteration of the loop.
  - A set of statements is executed in a loop until *some_collection* is exhausted.
- We know what a variable is, but what is a collection?
  - We haven't studied collections yet, but we have been exposed to one.  It is the string data type.  Put the following code in the shell:

```
word = 'himalayas'
for ltr in word:
    print(ltr)
```

  - We will read data from files later in the course.  A file is a collection.
  - In Python II, we will study a broad range of collections: strings, lists, tuples, dictionaries and sets.
  - As shown earlier, the continue statement can be used to short-circuit any iteration.

# Looping N Times

- There is a built-in function that produces a collection of integers that it delivers one at a time.
  - We can use it to loop a given number of times; something we will do later when we simulate rolling a pair of dice a large number of times.
  - In the above case, we don't use individual numbers from the collection. We just want to stop when a certain limit is reached.
  - In the next lab, we will actually use each number as you will see.
- This function is *range* which takes 1 to 3 arguments.
- The general form is *range(start, stop, increment)*.
  - *start* defaults to zero and *increment* defaults to 1.
  - If only one number is supplied, it is the stop number.
  - Two numbers represent the *start* and the *stop* values.
  - Three numbers  represent all three options.
  - See the sample - d2For.jpg.  Note: You do not get the stop number.

# LAB 10

Re-implement the previous lab using a for statement with a range function to convert to Centigrade the Fahrenheit temperatures from -40 to 110 in increments of ten degrees (i.e., -40, -30, -20, …, 110). Skip the Fahrenheit temperatures zero and 50 when doing the conversions. Use a continue statement to make this happen.

# Loops Within Loops (while)

- Loops can contain other loops.
- In this example, we will use a condition rather than True.

```
x = 0
while x < 5:
    y = 0
    while y < 3:
        print(x, y)
        y += 1
    x += 1
```



```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
4 0
4 1
4 2
```

- Break statements work only in the loop in which they are executed.

# Loops Within Loops (for)

- Loops can contain other loops.

```
x = 0
while x < 5:
    y = 0
    while y < 3:
        print(x, y)
        y += 1
    x += 1
```



```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
4 0
4 1
4 2
```



- We can replace the while loops with for loops with less code.

```
for x in range(5):
    for y in range(3):
        print(x, y)
```

# LAB 11

- Using the for-loop technique described on the previous slide, create a times table.
- Use the results of Lab 06b and modify these using for loops.

Sample output:

```
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

- Again, don’t worry about getting numbers aligned unless you finish early.

# File Processing

- There are basically two types of files: text and binary.
- They are external to the program; usually on a hard drive.
- Text files contain records that are strings of encoded characters.
  - In our case, the records are mostly ASCII-encoded characters.
  - Records in a text file are delimited by newline characters (\n).
  - We will be dealing exclusively with text files in this class.
- Binary files contain any kind of data.
  - Database records can be a combination of text along with integers and floats in binary form.
  - Graphical items (e.g., jpegs) may contain only binary data.
  - Newline characters may not be present in binary files.

# FILE Processing

1. Open the file
   a) Use the open built-in function and assign the result to a variable
   b) Specify the file and the mode used to access it.
2. Process the file one record at a time
   a) Each record is delimited by a newline character or characters
   b) Use a for loop to process each record (recommended), or
   c) Use the  readline method to access each record
   d) You can read an entire file into memory using the read method
3. When finished with the file, close it using the close method

# Opening a File

- The open function takes two arguments: the file/location and the mode of access.
- Defining the file: absolute or relative
  - An absolute reference specifies the full file location as well as the file name.
  - A relative address assumes a starting point (current directory) and specifies a location based on that assumption.
- Mode of access takes a two-character string.
  - The first character is r, w or a for read, write or append.  r is the default.
  - The second character is t or b for text or binary.  t is the default.
  - If the second argument is omitted, rt is assumed.  Our recommendation is to always be explicit and specify both of them.
- Demo (open, read, close) with samples iRead_File1.jpg, iRead_File2.jpg and iRead_File3.jpg

# LAB 13a

Re-implement Lab 08a to replace the data source with a file instead of **input()**. The file should be the data in the temps.dat file in your data folder. Be sure to account for any bad data that might be contained in this file.

Answers:

```
83.0 degrees Fahrenheit is 28.3 degrees Centigrade
It's hot.
104.0 degrees Fahrenheit is 40.0 degrees Centigrade
It's very hot!
58.0 degrees Fahrenheit is 14.4 degrees Centigrade
It's chilly,
15.0 degrees Fahrenheit is -9.4 degrees Centigrade
It's cold!
Input contains non-numeric data - 'q23\n'
-5.0 degrees Fahrenheit is -20.6 degrees Centigrade
It's cold!
76.0 degrees Fahrenheit is 24.4 degrees Centigrade
It's nice out.
```

# More Complex Programs

- Usually, as you read records, you have to accumulate information.
- The most basic information accumulated would be counts and sums.
  - In this case the initial setting for these variables would be zero.
  - A common use of such data is to calculate averages.
- Somewhat more complex data might be the largest or smallest entry.
  - In this case, the initial settings for the variables would depend on the data in question.
  - For the largest, the initial setting would be ridiculously low.
  - For the smallest, the initial setting would be ridiculously high.
- There are many other statistical ways to deal with data including categories or percentiles of data.

# Lab 13b

Change the last lab to remove the descriptions (in the if/elif/else statements). Instead, read the same data accumulating the information necessary to print the following centigrade temperatures:

- the average centigrade temperature
- the highest centigrade temperature
- the lowest centigrade temperature

Print these results in a neat readable form.

Answers:

```
Input contains non-numeric data - 'q23\n'
Average Temperature 12.9
High Temperature 40.0
Low Temperature -20.6
```

# LAB 14 – Read File

Create a plan and write a program to read the file in your data folder labeled trees.dat. This file contains the measurement in even feet of the height of a sampling of California coastal redwood trees. Your job is to read the data and produce a report showing the following information:

The number of trees in the sample,

The average height of all the trees to the nearest tenth of a foot,

The height of the tallest tree,

The height of the shortest tree, and

The number of trees over 300 feet tall

| | |
|---|---|
| Total Trees | 998 |
| Average Height | 215.4 |
| Tallest Tree | 316 |
| Shortest Tree | 99 |
| Over 300 Feet | 3 |

Format the data so the report has a professional look. Watch for bad data.

# Writing Records

- To write records to a file, you must open the file with a 'w' as the second argument. As before, you should enter 'wt' to be explicit.
  - Opening a new file causes that file to be created.
  - Opening an existing file destroys the data in that file.
- To write a record, use the write method. This method takes one argument which must be a string.
- Whereas the print function adds a newline character to each printed record, the write method writes only what you give it. It adds nothing.
- If you fail to close the file, the CPython interpreter will close it for you. Other interpreters may not. Always close the file.
  - If you are using Idle and fail to close the file. The file will appear empty when you open it with a text editor. That's because the interpreter doesn't automatically shut down when you use Idle.

# LAB 15

See lab15.py in your data. In place of filename in the open statement, substitute the path/filename on your computer. Then simply run the program. When finished, open the file you created with a text editor. Observe what the output looks like. Determine what has to be changed in the program to make the data appear as separate records.

# LAB 16 – Write File

Change the previous lab on tree heights to include the same report in a file that you can open and read with a text editor. Place this file in the same directory/folder as the input file. Call it whatever you like.

# Review

- When writing a program, what is the first thing you do?
- When writing a program, what should you always have?
- What is sequence? What statement is used in selection? What two statements do we use for iteration?
- What is indentation used for in Python? What is the standard indentation?
- What are the three types of errors you can encounter running a program?
- What built-in functions have we used? Imported functions?
- What statement allows us to define our own function? Intercept errors?
- What methods did we use to read a file? Write a file?
- What technique allowed us to avoid using a method to read a file?
- What character in the ASCII chart is produced by using the newline character (\n)?

# Calculating/Printing Percentages

- Let's say I have 30 temperature readings and 10 of them are over 90 degrees Fahrenheit. How do I calculate that percentage?
- One Answer:
  x = 10 / 30 * 100
  print('{0:.1f}%'.format(x)) # result -> 33.3%
- Another way using % in place of f:
  x = 10 / 30 # Multiplying by 100 no longer necessary
  print('{0:.1%}'.format(x)) # result -> 33.3%
- Either way is fine. Using the % eliminates some complexity.

# LAB 17 - Dice Roll

Use the randrange function in the random module to simulate rolling a pair of dice. Simulate 10,000 rolls and calculate the percentage of the rolls that are sevens and the percentage of the rolls that are twos. Express your results to one decimal place.

Mathematically, the percentage of sevens will be 16.7% and the twos will be 2.8%. Your answers should close to these values.

Use Python Help or Pydoc3 to determine what the difference is between the randrange and randint functions in the Random module.

# RESOURCES

- Your humble instructor
- Python for Everybody - book
- tutorialspoint.com/python3
- python-course.eu/python3_course.php
- Think Python - Interactive and the PDF
- greenteapress.com
- PEP 8 – Python style guide

# Python II Overview

- Review including a review of the take-home  lab solution.
  - Does not preclude the need for complete understanding of Python I material
- Data structures – purpose, operations and methods
  - Strings
  - Lists, Tuples
  - Dictionaries
  - Sets
- Concept of mutability
- Concept of tables/arrays
- Concept of iterables and iterators

# Take-Home Assignment

- Review Python for Everybody
  Chapters 1 – 5 and 7 including the exercises – These chapters should be understood completely.
- Preview chapters 6, 8 and 9. Python II is very concentrated and intense. You should not take this class until you are at least somewhat familiar with the data structures covered in these three chapters. It is not necessary to master the details or do the exercises. Just be familiar with the subjects prior to class.
- Review and understand all of the completed labs from Python I.
- Answer all of the take-home questions in your downloaded data.
- Practice formatting. See Python Notes for details.
- Work on the take-home lab. Do a little coding every day regardless of what you are working on.

# Take-Home LAB

- Create a program that calls a function that simulates the rolling of a pair of dice. Use randrange from the random module to accomplish this. Your main program will deal with the total of the two dice.
- The rules are as follows:
  - On the first roll, a total of 7 or 11 is an automatic win
  - On the first roll, a total of 2, 3 or 12 is an automatic loss
  - Any other number is called the Point. You must roll again. On subsequent rolls the rules are as follows:
    - You roll a 7 which is a loss.
    - You roll the Point number again which is a win.
    - Any other roll has no meaning and another roll is required.
- You start with $100 and bet $10 on each play.
- Print all the rolls and whether you have won or lost on one line.
- Print the funds balance and a request to play again on the next line. A ‘y’ or ‘Y’ means play again. Anything else ends play. A balance of $0 ends play automatically.

# Take-Home LAB

The output from your program should look something like the following:

Beginning Balance = $100
7  You win!
Balance = $110 – Play again?  y/n:  y
10 4 12 8 7  You lose!
Balance = $100 – Play again?  y/n:  y
3  You lose!
Balance = $90 – Play again?  y/n:  y
8 6 3 9 8  You win!
Balance = $100 – Play again?  y/n:  y
4 12 6 9 7  You lose!
Balance = $90 – Play again?  y/n:  n
Number of plays - 5
Ending Balance = $90

# THE END