| Week 1 - Monday March 28, 2016 | 3 |
| Week 1 - Wednesday March 30, 2016 | 11 |
| Week 1 - Friday April 1, 2016 | 12 |
| Week 2 - Monday, April 4, 2016 | 16 |
| Week 2 - Wednesday, April 6, 2016 | 18 |
| Topic: Coding standards | 19 |
| Topic: Python operators and operations | 20 |
| Topic: The scope and lifetime of a variable | 21 |
| Topic: Keywords, Built-In Functions, Types | 23 |
| Topic: Command line prompt | 25 |
| Topic: Lists and Tuples | 28 |
| Topic: Looping and Control Constructs | 29 |
| Week 3 - Monday, April 11, 2016 | 34 |
| Week 3 - Wednesday, April 13, 2016 | 35 |
| Model-View-Controller (MVC) | 36 |
| Week 3 - Friday, April 15, 2016 | 38 |
| Week 4 - Wednesday, April 20, 2016 | 44 |
| Procedural Abstraction | 45 |
### Data Abstraction

Getting knowledge into LTM is very time consuming.

| Week 4 - Friday, April 22, 2016 | 53 |
| Week 5 - Wednesday, April 27, 2016 | 55 |
| Week 5 - Wednesday, April 27, 2016 | 56 |
| Week 6 - Friday, May 6, 2016 | 57 |
| Week 8 - Monday, May 16, 2016 | 58 |
| When to use OOP versus Procedural | 59 |
| Exceptions | 62 |
| Week 9 - Monday, May 23, 2016 | 69 |
| More on inheritance using super( ) | 70 |
| The “diamond problem” in a class hierarchy. | 71 |
| Week 9 - Wednesday, May 25, 2016 | 75 |
| Week 9 - Friday, May 27, 2016 | 76 |
| __del__( ), del, smart pointers, and garbage colletion | 77 |
| Week 10 - Friday, June 3, 2016 | 79 |
Week 1 - Monday March 28, 2016

Agenda:
__ Start Class
__ Overview of Topics
__ What you Should Already Know
__ Important Policies
__ Attend Labs to Start P1:
    Monday 4P, Tues 11A, 2P, 5P
__ Project 1
    __ Project Description
    __ Project Code in PyCharm

Start Class

Please be so kind as to put away your laptops and turn off your smartphones and we will get started. Thank you.

Instructor and GTFs
Anthony Hornof, Associate Professor
Jeremy Sigrist, Graduate Teaching Fellow
Sabin Kafle, Graduate Teaching Fellow

Overview of Topics
Object-oriented design and implementation.
    Class hierarchies and inheritance, multiple inheritance.
    Polymorphism, data hiding, data abstraction, modularity.
Linear data structures such as lists, stacks, queues.
Namespaces and scope.
How to systematically test modular object-oriented programs.
How to think like a computer scientist.
How to be rigorous.

You are Expected to Know
Most of the topics in Chapters 2-14 of Downey (2012) “Think
Program: How to Think Like a Computer Scientist”.
Version 2.0.16. Specifically:

1 The way of the program
1.1 The Python programming language
1.2 What is a program?
1.3 What is debugging?
1.4 Formal and natural languages
1.5 The first program
1.6 Debugging
1.7 Glossary
1.8 Exercises

2 Variables, expressions and statements
2.1 Values and types
2.2 Variables
2.3 Variable names and keywords
2.4 Operators and operands
2.5 Expressions and statements
2.6 Interactive mode and script mode
3 Functions
3.1 Function calls
3.2 Type conversion functions
3.3 Math functions
3.4 Composition
3.5 Adding new functions
3.6 Definitions and uses
3.7 Flow of execution
3.8 Parameters and arguments
3.9 Variables and parameters are local
3.11 Fruitful functions and void functions
   (“Fruitful” is the author’s name for functions that return values. This is not standard jargon.)
3.12 Why functions?
3.14 Debugging
3.15 Glossary
3.16 Exercises

(Nothing from Chapter 4: Interface design)

5 Conditionals and recursion
5.1  Modulus operator
5.2  Boolean expressions
5.3  Logical operators
5.4  Conditional execution
5.5  Alternative execution
5.6  Chained conditionals
5.7  Nested conditionals
   (You are not expected to know recursion yet.)
5.12  Debugging

6  Fruitful functions
   (This is not standard jargon.)
6.1  Return values
6.2  Incremental development
6.3  Composition
6.4  Boolean functions
   (You are not expected to know recursion yet.)
6.9  Debugging
6.10  Glossary
6.11  Exercises

7  Iteration
7.1  Multiple assignment
7.2  Updating variables
7.3  The while statement
7.4  The break statement
...
8 Strings
8.1 A string is a sequence
8.2 The \texttt{len} statement
8.3 Traversal with a loop
8.4 String slices
8.5 Strings are immutable
8.6 Searching
8.7 Looping and counting
8.8 String methods
8.9 The \texttt{in} operator
8.10 String comparison
8.11 Debugging
8.12 Glossary

9 Case study: word play
9.1 Reading word lists
9.2 Exercises
9.3 Search
9.4 Looping with indices
9.5 Debugging
9.6 Glossary
9.7 Exercises

10 Lists
10.1 A list is a sequence
10.2 Lists are mutable
10.3 Traversing a list
10.4 List operations
10.5 List slices
10.6 List methods
10.7 Map, filter and reduce
10.8 Deleting elements
10.9 Lists and strings
10.10 Objects and values
10.11 Aliasing
10.12 List arguments
10.13 Debugging
10.14 Glossary
10.15 Exercises

12 Tuples
12.1 Tuples are immutable
12.2 Tuple assignment
12.3 Tuples as return values
12.4 Variable-length argument tuples
12.5 Lists and tuples
12.6 Dictionaries and tuples
12.7 Comparing tuples
12.8 Sequences of sequences
12.9 Debugging
12.10 Glossary
12.11 Exercises

14 Files
14.1 Persistence
14.2 Reading and writing
14.3 Format operator
14.4 Filenames and paths
(Nothing about exceptions)

Note that this is only about 120 pages in the book.

Let me know by this Friday if you need me to review any of these topics in class.

This is a programming class but computer programming is changing the world. There is very little done in any contemporary workplace, field, or discipline that does not rely heavily on programming.

• New Internet and cloud computing services continue to change the way people communicate and collaborate.
• Smart grids, smart buildings, and smart transportation are fundamentally changing the way people live.
• Most fields of science now rely heavily on computer simulation to explore and understand phenomena.
• Computer programming has dramatically changed medicine and how treatment are administered and tracked.

Computer programming is at the core of these advances.

I want you to become great programmers.

Important Policies
Please do not use your laptops or smartphones in class. It is very distracting for everyone.
There is no “pair programming” in this class. Your projects must be your own work. You need to cite any code that you submit that is not your own.

We will have quizzes every Monday. This week on Friday.

**Agenda:**
- ✓ Start Class
- ✓ Overview of Topics
- ✓ What you Should Already Know
- ✓ n Important Policies
- ✓ Attend Labs to Start P1:
  - Mon 4P, Tues 11A, 2P, 5P
- __ Project 1
  - __ Project Description
  - __ Project Code in PyCharm
Week 1 - Wednesday March 30, 2016

Agenda
__ Hand out syllabus. Go over a few important points.
__ Project advice: Read the project documents carefully.
__ Quiz - (On your preparedness for this class, and P1)
__ Announcements: New links on CIS course web page.
__ Read for Monday from docs.python.org:
   (1) 4.7.1 https://docs.python.org/3/library/stdtypes.html#string-methods
        If this is too hard, first read ThinkPython Chapter 8.
   (2) The specification for the file object open( ) function at
        https://docs.python.org/3/library/functions.html?highlight=open#open
        If this is too hard, start by reading ThinkPython Section 9.1.
   (3) The specification from 4.6 to 4.7.1 for sequences lists and tuples in
        https://docs.python.org/3.5/library/stdtypes.html
        If this is too hard, start by reading ThinkPython Chapter 10 and 12

Also Read:
(4) The Syllabus
(5) P1 handout and P1 starter code.
(6) Downey (2010) *ThinkPython 2e* Chapters 1 through 8 as needed to catch up on topics you did not not master in previous classes.
How to write code for Project 1.

Work step by step, one problem at a time, save your code.

Work with accurate, reliable and well-written sources.

Implement get_inputfile_object()

Implement read_first_line()

Adding a command line argument in a PyCharm project.

Read for Monday’s quiz

How to write the code for Project 1.

Work step by step, solving one problem at a time.

• Read the specification, the starter code, practice files, and addendum.
• Start small, making sure you understand all the services and functions that you need to use.
• Work with each function separately, fulfilling its specification.
• Keep a documented code diary of the things you figured out along the way:  practice-code-<yourname>.py
Work with accurate, reliable and well-written sources. The official Python documentation: https://docs.python.org/3/
Downey (2010) ThinkPython 2e
http://greenteapress.com/wp/think-python-2e/
http://www.qtrac.eu/py3book.html - source code

Implement get_inputfile_object( )
• Read the project document, Addendum, and other docs.
• Create a to-do list of the things you need to do.
• Start as small as possible, just working with the few commands you need to use in a tiny little source code file.
• Test to make sure everything works after every step.
• Save your practice code after you get it working.
• Look up all of the Python components that you need. “file object” in the index of https://docs.python.org/3/
• Create the object. Confirm what it is. Use additional sources such as Summerfield if necessary.
• Implement get_inputfile_object( ) to fulfill its docstring but only for the default input filename.

Implement read_first_line( )
• Read the project document, Addendum, and other docs.
• Create a to-do list of the things you need to do.
• Start as small as possible, just working with the few commands you need to use in a tiny little source code file.
• Test to make sure everything works after every step.
• Save your practice code after you get it working.
• Look up all of the Python components that you need.
  
  https://docs.python.org/3/library/stdtypes.html#string-methods

Adding a command line argument in a PyCharm project.
This is an issue with every Integrated Development Environment (IDE). You can usually set it somewhere in the project settings.

Two ways:

1. Run your program from the command line using "python3 <program_name>.py arg1 arg2" in “View / Tool Window2 / Terminal” or a command line terminal (such as Terminal.app in the Mac).
   or

2. “Run / Edit Configuration...” “Script Parameters” or

Read for Monday’s quiz
From docs.python.org:

(1) 4.7.1 https://docs.python.org/3/library/stdtypes.html#string-methods
    If this is too hard, first read ThinkPython 2e Chapter 8.
(2) The specification for the file object open( ) function at https://docs.python.org/3/library/functions.html?highlight=open#open
    If this is too hard, start by reading ThinkPython Section 9.1.
(3) The specification from 4.6 to 4.7.1 for sequences lists and tuples in https://docs.python.org/3.5/library/stdtypes.html
    If this is too hard, start by reading ThinkPython Chapter 10 and 12

Also Read:
(4) The Syllabus.
(5) P1 handout and P1 starter code.
(6) Downey (2010) *ThinkPython 2e* Chapters 1 through 12 as needed to catch up on topics you did not master in previous classes.

Agenda

- √ How to write code for Project 1.
- √ Work step by step, one problem at a time, save your code.
- √ Work with accurate, reliable and well-written sources.
- √ Implement `get_inputfile_object()`
- √ Implement `read_first_line()`
- √ Adding a command line argument in a Pycharm project.
- √ Read for Monday’s quiz
Agenda
__ Quiz.
__ P1-Initial Submission - partial solution available on Canvas
__ Review of student-submitted Project 1 - Initial Submission
__ How to explicitly check if a file is available.
__ How to figure out how to explicitly check...
__ Use the global constants.
__ Use the debugger.
__ Adding a command line argument in a Python program.
__ Wednesday - Go over the Preparedness Quiz

Adding a command line argument in a Python program.
• Read the project specification and the starter code. Make sure you understand everything you are reading, such as:
  import sys # for argv
• If you do not know what this is, look it up in a reliable source. You know what import is from CIS 210 (import math). If not, read in the reliable sources. Look up sys module in the index of [https://docs.python.org/3/](https://docs.python.org/3/)
• Look up all of the Python components that you need.
  “command-line argument” in Summerfield (2010) index
Code Review of Project 1 - Initial Submission

• If you are not testing your code, there is little point in submitting it to be graded.

• Set up your test environment. Get the sample I/O files from the project website.

• To be an effective programmer, you need a good “diff” tool. A tool that compares the contents of two text files, and shows you where they are different. On the Mac, TextWrangler / BBEdit has one.

• Use your “diff” to compare your program’s output to the target output.
Week 2 - Wednesday, April 6, 2016

Agenda
__ Review of Quiz #2
__ Review of Preparedness Quiz Part 1
__ Mini Lectures on Critical Topics in Computer Programming, and Python
   __ Coding Standards
   __ Python operators and operations
   __ Scope and lifetime of a variable
   __ Keywords, built-in functions, types
   __ Command line prompt
   __ Lists and tuples
   __ Looping and Control Constructs
__ Read for next Monday:
Computer programs must be readable both by computers and by humans. Well-written code runs better. It is less likely to have errors now, and less likely to lead to errors in the future.

Q1: If you may have gotten this question wrong and if you want to do better in this class, please read or review: https://www.cs.uoregon.edu/Classes/16W/cis210/handouts/styleguide.php

Q2: If you may have gotten this question wrong and if you want to do better in this class, please read or review ThinkPython2e “Chapter 3: Functions” to see how functions are structured and called. “Read” includes trying all the Python code provided in the reading, studying the glossary, and doing the exercises at the end of the chapter.
Evaluation Order and Operator Precedence
These are critical concepts for any programming language.

https://docs.python.org/3/reference/expressions.html#evaluation-order

Python evaluates expressions from left to right. When evaluating an assignment, the right side is evaluated before the left side.

\[ a[i + 1] = f(j) \]

Order of evaluation:
expr1, expr2, expr3, expr4
(expr1, expr2, expr3, expr4)
{expr1: expr2, expr3: expr4}
eexpr1 + expr2 * (expr3 - expr4)
eexpr1(expr2, expr3, *expr4, **expr5)
eexpr3, expr4 = expr1, expr2

(from docs.python.org)

Q3: Read 6.15 & 6.16, study Operator Precedence at:
https://docs.python.org/3/reference/expressions.html#operator-precedence

Q3: Read ThinkPython2e Chapter 2: Variables, expressions and statements. 2.5: “Order of operations”

• Q4 & Q5: 7.2.1. Augmented assignment statements
https://docs.python.org/3/reference/simple_stmts.html#augmented-assignment-statements

Note how it discusses the order of operations of \*= .
**Topic: The scope and lifetime of a variable**

`scope` defines what part of a program has access to a variable. As opposed to `lifetime`, which defines the time period of time in which a variable has memory allocated to it. Very important programming terms and concepts.

If `twice()` called another function without passing `m`, `m` would not be in scope in that other function, but it `m` would still be within its lifetime.

```python
def f2(y):
    # y in scope; m, x not
    y *= 1  # Though m,x still alive.
    return y  # Lifetime of y ends here

def twice(x):
    m = 2  # x, m in scope
    m = f2(m)
    return m * x  # end of life of m,x

# m, x, y are out of scope and dead.
twice (2)
```

Watch it all happen in the debugger.
The LEGB scope lookup rule. When a variable is referenced, Python searches for it in this order:

1. In the local scope.
2. In any enclosing functions’ local scopes.
3. In the global scope.
4. Lastly, in the built-in scope.

The first occurrence wins. The place in your code where a variable is assigned usually determines its scope.

Q6 & Q7: Read Python docs 4.1 and 4.2.1, and 4.2.2
https://docs.python.org/3/reference/executionmodel.html#execution-model
Generate some practice code to test the LEGB lookup rule.
Every text-based programming language will have a set of keywords and built-in functions that form that language and the nature of that language.

The keywords in the C programming language:

<table>
<thead>
<tr>
<th>auto</th>
<th>double</th>
<th>int</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>else</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>register</td>
<td>typedef</td>
</tr>
<tr>
<td>char</td>
<td>extern</td>
<td>return</td>
<td>union</td>
</tr>
<tr>
<td>const</td>
<td>float</td>
<td>short</td>
<td>unsigned</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>signed</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>goto</td>
<td>sizeof</td>
<td>volatile</td>
</tr>
<tr>
<td>do</td>
<td>if</td>
<td>static</td>
<td>while</td>
</tr>
</tbody>
</table>

All of the type specifiers are in **bold italics**. Many pertain to specifying variable types. C is “strongly typed”

The keywords in the Python programming language:

<table>
<thead>
<tr>
<th>False</th>
<th>class</th>
<th>finally</th>
<th>is</th>
<th>return</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>continue</td>
<td>for</td>
<td>lambda</td>
<td>try</td>
</tr>
<tr>
<td>True</td>
<td>def</td>
<td>from</td>
<td>nonlocal</td>
<td>while</td>
</tr>
<tr>
<td>and</td>
<td>del</td>
<td>global</td>
<td>not</td>
<td>with</td>
</tr>
<tr>
<td>as</td>
<td>elif</td>
<td>if</td>
<td>or</td>
<td>yield</td>
</tr>
<tr>
<td>assert</td>
<td>else</td>
<td>import</td>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>break</td>
<td>except</td>
<td>in</td>
<td>raise</td>
<td></td>
</tr>
</tbody>
</table>

None pertain to specifying or revealing variable types.
The built-in functions in Python:

abs()  dict()  help()  min()  setattr()
all()   dir()   hex()   next()  slice()
any()   divmod() id()   object()  sorted()
ascii() enumerate() input()  oct()  staticmethod()
binary() eval()  int()  open()  str()
bin()   exec()  isinstance() ord()  sum()
bool()  filter()  issubclass() pow()  super()
bytearray()  float()  iter()  print()  tuple()
callable()  format()  len()  property() type()
chr()    frozenset()  list()  range()  vars()
classmethod()  getattr()  locals()  repr()  zip()
compile()  globals()  map()  reversed()  __import__()  
delattr()  hash()  max()  round()  
delattr()  hash()  memoryview()  set()

Many pertain to specifying or revealing variable types.

Python is different from other programming languages that require you to specify a type for each variable as in:

```python
int i = 3;
i = 'd' //i==100, still an int. Why?
```

In Python, type is often implied by the value of the variable. But the importance of a variable’s type matters tremendously in Python, just as it does in other programming languages.

• Q8: Read ThinkPython2e Chapter 1.
• Q9/10: Memorize to recognition the Python keywords https://docs.python.org/3/reference/lexical_analysis.html#keywords
• Q9/10: Memorize to recognition the built-in functions https://docs.python.org/3/library/functions.html?highlight=open#built-in-functions
These are always available, not just for some classes.
• Q9/10: Study open ( ) and note that ‘r’ is the default. https://docs.python.org/3/library/functions.html?highlight=open#open
Here is a simple and classic user interface to a computer’s OS, or operating system, the computer program that runs the computer: a prompt with a flashing cursor:

> 

You can recreate it in a Mac terminal by typing

> PS1='>'

and, in Preferences, setting the cursor to a blinking underline.

You type commands from your human memory, hit enter, and the computer responds to the commands. No response means either the command executed with no problem, or simply has nothing to report.

> help
-bash: hello: command not found
> ls
> _

(if the current directory is empty)

Some commands take “options” and “arguments”

> ls -l x.txt
-rw-r--r-- 1 hornof staff 2175 Jul 5 2014 x.txt
> _
Commands are either built-in to the OS or are found in the PATH environment variable (on Mac, Windows, and Unix).

You are always “in” one specific current working directory. A directory is analogous to a folder in windows-based access to a computer’s operating system. Directories are specified by strings separated by slashes (backslashes on Windows):

```bash
>pwd
/Users/hornof
>_```

Yes, it is limited, but you also have great power, such as the ability to write scripts that access and process multiple files in a directory.

Windows, Unix, and now Mac all have command line interfaces. It is important to have some basic fluency with the command line interpreter used by your operating system.

For example, this is how you install modules in Python:

```bash
>pip3 install numpy
```

This is how you run a Python script at the command line:

```bash
>python3 your_program arg1 arg2
```
Remedy for errors on Q11, Q12, & Q13:

**Basic Windows Command Line commands:**
- [http://people.cis.ksu.edu/~schmidt/200f07/setpath.html](http://people.cis.ksu.edu/~schmidt/200f07/setpath.html)

**Basic Macintosh Terminal Unix commands:**
- [https://tidbits.com/article/7003](https://tidbits.com/article/7003)
- [http://mally.stanford.edu/~sr/computing/basic-unix.html](http://mally.stanford.edu/~sr/computing/basic-unix.html)

**Setting environment variables**
[https://geosci.uchicago.edu/~rtp1/PrinciplesPlanetaryClimate/Python/ShellsNStuff.html](https://geosci.uchicago.edu/~rtp1/PrinciplesPlanetaryClimate/Python/ShellsNStuff.html)

(Note that I prefer to include “site:.edu” in my searches but even then I always ask myself “What is the source?”)

**In the command line interpreter, you should learn how to:**
- View and change the current working directory.
- List the files and directories in the current directory.
- View the contents of text files, one screen at a time if it is a long file.
- Create, rename, move, and delete both files and directories.
- Type a string that goes into a newly created text file.
- Run a program and have the output go into a text file.
- See and set the current path.
Lists and tuples are very similar, but very different. An example of each:

```
li = [1, 2, 3]
tu = (1, 2, 3)
```

Note that the [ ] and ( ) are *type specifiers*.

**Lists**
- Lists use square brackets [ ].
- Lists are mutable.
  - Can be changed in place.
    ```
    li[0] = 4
    ```
- Some functions change in place and return `None`.
- Use if contents will change.
- A little slower. Must support multiple kinds of access.

**Tuples**
- Tuples use parentheses ( ).
- Tuples are immutable.
  - *Cannot* be changed in place.
    ```
    tu[0] = 4  # Error!
    ```
- No functions will change in place.
- Use if contents won’t change.
- Faster. Can be optimized for fast-access reading only.

Q14/15: If you did not do well on Q14/Q15, you should read *ThinkPython2e* Chapters 10 “Lists” and 12 “Tuples”.
Try to learn to think like a computer. Think about the *flow of control*, also known as the *flow of execution*. In a computer program, there is typically a single point of control that moves through the code, like a single ball rolling down

The flow of execution is like a ball—one ball—moving down a series of ramps, with switches that decide very deliberately which path the ball will take at each decision point.

You need to learn how to follow that flow of execution as you read code. You can use the debugger to confirm that the flow of execution is what you think it is.

Learn how to write down the values of variables as you step through a section code.

This is a critical tool in understanding what your code is doing. Even moderately complex code presents more variables values than you can keep track of in your short term memory (“in your head”) and you need an organized way to track the values.
Practice tracing through the code keeping track of variable names, but by writing them down. Write down with pencil and paper how the variables change.

```python
sum = 0
anum = 1
while anum < 5:
    sum += anum
    anum += 1
print(sum)
```

<table>
<thead>
<tr>
<th>where you are in the code</th>
<th>sum</th>
<th>anum</th>
<th>while anum &lt; 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>initially</td>
<td>0</td>
<td>1</td>
<td>out of scope</td>
</tr>
<tr>
<td>entering while loop</td>
<td>0</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>at start of loop</td>
<td>0</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>at end of loop</td>
<td>1</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>at start of loop</td>
<td>1</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>at end of loop</td>
<td>3</td>
<td>3</td>
<td>T</td>
</tr>
<tr>
<td>at start of loop</td>
<td>3</td>
<td>3</td>
<td>T</td>
</tr>
<tr>
<td>at end of loop</td>
<td>6</td>
<td>4</td>
<td>T</td>
</tr>
<tr>
<td>at start of loop</td>
<td>6</td>
<td>4</td>
<td>T</td>
</tr>
<tr>
<td>at end of loop</td>
<td>10</td>
<td>5</td>
<td>F</td>
</tr>
<tr>
<td>at print( )</td>
<td>10</td>
<td>5</td>
<td>F</td>
</tr>
</tbody>
</table>

A critical tool in understanding what code is doing. Confirm your belief of what the code is doing using what?

Q16/17: If you did not do well on Q16 or Q17, read ThinkPython2e Chapters 3, 5, 7.
Python has some unique sequence evaluation and iteration constructs, and idioms. Try to learn them.

`in` is a operator is a membership operator. It takes two sequences and returns True if the first sequence appears within the second sequence.

- Q18/19: Read *ThinkPython2e* Chapters 8 “Strings”.

`for <x> in` is a compound statement that returns an “iterable object” that is used to step through a sequence (such as a string, tuple or list), making each element of the sequence available one time to a block of code.

- Q18/19: Read “8.3 The for statement”
  [https://docs.python.org/3.5/reference/compound_stmts.html#the-for-statement](https://docs.python.org/3.5/reference/compound_stmts.html#the-for-statement)

`range(<y>)`

`range(<x>,<y>)`

creates an “immutable sequence type” that can be used to step through a sequence of integers. If one argument is supplied, it ranges from 0 to one before argument. If two arguments are supplied, it ranges from the first argument to just before the second argument.

- Q18/19: Read “4.6.6 Ranges” at [https://docs.python.org/3/library/stdtypes.html#ranges](https://docs.python.org/3/library/stdtypes.html#ranges)

`for i in range(len(L))`

A common idiom for iterating over a list \( L \), but \( i \) starts at 0.
import math  # This is the best and safest way to import.
s = math.sqrt(99)  # Module specified every access. This provides safe, fully-qualified access.

import math as math
math.sqrt(99)
This presents a risk of name collisions with math such as:
math could have been previously assigned a value, and is overwritten with <module 'math' from ...>.
sqrt could be set to a new value after math is imported but before sqrt is called, and it will no longer be callable.
Even worse: sqrt could be redefined as a function that returns a number.

from math import sqrt
s = sqrt(99)
This presents a risk of a name collision with sqrt
sqrt is now <built-in function sqrt>

from math import *
This presents a risk of a name collision with every single function in the module.
sqrt, exp, log are all <built-in function sqrt>

Q20: Docs 7.11  https://docs.python.org/3/reference/simple_stmts.html#the-import-statement (adapted from Summerfield, 2010)
Mini Lectures on Critical Topics in Computer Programming, and Python

- Coding Standards
- Python operators and operations
- Scope and lifetime of a variable
- Keywords, built-in functions, types
- Command line prompt
- Lists and tuples
- Looping and Control Constructs

Summary of Results from Preparedness Quiz:

1. Roughly a third of the class appears to have pretty good mastery of the CIS 210 material, which is required for this class (a score of 11 or higher on the quiz)
2. For many, if you want to do well in this class, you will need to go back and revisit a lot of the material covered in CIS 210, and very diligently complete all of the assigned reading.
Week 3 - Monday, April 11, 2016

Agenda
__ Quiz
__ Finish review of Critical Topics in Computer Programming, and Python
  __ Lists and tuples
  __ Looping and Control Constructs
__ Walkthrough Project 1 Solution
__ Note readings for the week: Namespaces, classes, objects.
__ Review Project 2 Handout

Command interpreter loop
While (looping)
  Display a prompt.
  Wait for user input.
  Examine the first word, which is typically the command.
  If the command is valid then
    Send to subroutine to process the rest of the line.
    If the command was “quit”, set looping to false.
  Else if command was not valid then
    Output error.
Loop.
Week 3 - Wednesday, April 13, 2016

Agenda
__ Review Quiz
__ Walkthrough Project 1 Solution
__ Project 2 Handout

Lecture Topics
__ Model-View-Controller

Reading Topics
• Namespaces, classes, objects.
Model-View-Controller (MVC)

MVC is the prototypical (best single example of a) “software design pattern”
A “software design pattern” is a standard established solution to a recurring problem in computer programming.

**The Model**
Holds the data.
Reports data to any module that requests data.
Updates data in the model when other modules request that.
Example: A list of cities and temperatures.

**The View**
Provides a view of the data in the model.
Requests data from the model as needed.
Responds to requests from the controller to change the view.
Example: A spreadsheet view of the data, with scroll bars, or a bar graph of the data, with fields to enter city names.

**The Controller**
Provides a way for the human user to enter commands.
Could be text-based, mouse-clickable buttons, or other.
Sends data updates to the Model, view updates to the View.
Responds to user interactions captured by the View and sent.
Example: Tabbing through a spreadsheet, entering data, clicking on bars in a bar graph.
In Project 2, there is a Model class and a Controller class. Each is in a separate file. (There is no View class yet.) Read Chapters 15, 16, and 17 in ThinkPython2 to learn how to create and access member variables, member functions, how to create “private” member variables, and how to access these components from within a class, and from outside of a class.
Object-Oriented Programming (OOP)
__ Key idea: a self-contained collection of data and methods.
__ Important concepts associated with OOP.
  (1) data abstraction,
  (2) encapsulation (data hiding)
  (3) inheritance
  (4) polymorphism
__ Examples of how the ideas are implemented in Python.
  Python syntax for creating classes and objects.
  Python convention for imposing data hiding.

Object-Oriented Programming (OOP)
One key idea of OOP is that objects are self-contained collections of both data *and the methods* that operate on that data.

Important concepts in OOP include: (1) data abstraction, (2) encapsulation (data hiding), (3) inheritance, and (4) polymorphism.

Let’s just look at the first two...
1. **Data abstraction**: The process and result of isolating the representation details of a component from the rest of the program. In OOP, this is accomplished in part by using access functions (such as constructors, getters and setters, or readers and writers). These create a **data-abstraction barrier**.

Example: In a simulation of city busses, a central bus dispatcher could determine whether empty seats are available on a bus by either looking at every seat on the bus, or by calling a function `empty_seats()` that returns the number of seats that are currently available.

2. **Encapsulation (Data Hiding)**: The process and result of the bundling data attributes, and the methods for accessing and changing those data attributes, inside of a component. And of making some of those elements visible, and other elements invisible, to clients outside of the component.

Example: A bus object might not only provide an `empty_seats()` function; it also might make it specifically impossible for other components to see what specific seats are available.

“Nothing in Python makes it possible to enforce data hiding – it is all based upon convention.” (docs.python.org) Other languages provide specific mechanisms in the language to enforce dating hiding, such as the keyword “private” in C++ and Java. This is a shortcoming in Python. The convention in Python is to precede member variables intended to be private with two underscores.
A simple class in Python. A self-contained collection, but little or no data abstraction or encapsulation.

# Define the class
class A:

    # Class variables, set for the entire class.
    name = "class A"

    # Member functions.

    # a "setter" or "writer"
    def set_x(self, x_in):
        # Member variables can only be referenced
        # within a function that has access to
        # the calling object.
        
        # Member variable, set for the one instance.
        self.x = x_in

    # A "getter" or "reader" member function.
    def get_x(self):
        return self.x

# Use the class
# Show contents of class variable.
# No instances have been created yet.
print()
print ("A.name =", A.name)

# Create two objects, two instances.
a1 = A()
a2 = A()

# Set values
a1.set_x(111)
a2.set_x(222)

# Get values
print ("a1.get_x() =", a1.get_x())
print ("a2.get_x() =", a2.get_x())
What is “self” in these functions?

```python
class A:
    def set_x(self, x_in):
        self.x = x_in  # Creates a member variable.

a1 = A()
a1.set_x(111)
print("a1.x =", a1.x)
```

You can call `set_x()` for `a1` two different ways:

A.set_x(a1)
a1.set_x()

When you call a member function using the object-dot notation, the object gets passed in as the first argument. “self” is a reference to the calling object, `a1`.

```
a1,
\downarrow
a1.set_x(a1, 111)
```

```
a1,
\downarrow
a1.get_x()  # transforms into a1.get_x(a1)
```

Calling it “self” is just a convention. You can call it anything (that is not a Python keyword). “this” would be consistent with how the keyword is used in Java and C++.

Note that you need access to the calling object in order to create a member variable. Hence, member variables must be created inside of member functions. But note in the previous example how we can also access the member variables directly from outside of the class. They are all “public”. There is no data hiding.

`print("a1.x =", a1.x)` works fine unless you impose the Python convention for considering member variables as private.
A class with data abstraction and encapsulation.

# A city bus in a simulation.
class Bus:

    # __init__() is called automatically with every instantiation.
def __init__(self, num_seats):
        # "Private" member variables, by convention
        # Total number of seats on the bus.
        self.__num_seats = num_seats
        # A list of the availability of each seat
        self.__seat_taken_map = []
        # Fill the seat map with empty seats.
        i = 0
        while i < self.__num_seats:
            self.__seat_taken_map.append(False)
            i += 1
        print("Created a bus with", self.__num_seats, "seats.")

    # return the number of seats that are available.
def empty_seats(self):
        # initialize a counter
        i = 0
        for s in self.__seat_taken_map:
            # If the seat is not taken, increment the counter.
            if not s:
                i += 1
        # Return the total
        return i

    # Marks the next seat available as taken (True)
    # If no seats are available, output an error.
def take_a_seat(self):
        for i in range(len(self.__seat_taken_map)):
            # If the seat is available (False)
            if not self.__seat_taken_map[i]:
                # Then take it (make it True)
                self.__seat_taken_map[i] = True
                # And then exit.
                break # (Only take the first seat, not all.)
        else:
            print("No seats are available.")
Create a bus with a capacity of 4.

```python
b = Bus(4)
print("b.empty_seats()=", b.empty_seats())
b.take_a_seat()
print("b.empty_seats()=", b.empty_seats())
b.take_a_seat()
print("b.empty_seats()=", b.empty_seats())
b.take_a_seat()
print("b.empty_seats()=", b.empty_seats())
b.take_a_seat()
print("b.empty_seats()=", b.empty_seats())
b.take_a_seat()
b.take_a_seat()
```

The convention in Python is to precede private member variables with two underscores. Note how `print(b.__num_seats)` generates an error. But if we change the member variable to `num_seats`, it works.

**Summary**

Key ideas and motivations of OOP: data abstraction, encapsulation (data hiding), inheritance, polymorphism. Important OOP concepts that support these key ideas: class, object, instantiation, member variables (or attributes), private member variables, member functions (or methods), getters and setters (or readers and writers).
Week 4 - Wednesday, April 20, 2016

Agenda
__ Questions
__ Review Monday’s quiz.
__ Procedural Abstraction
__ Data Abstraction
__ How to read for retention

Take to class:
__ My OS notes and textbook.
Procedural Abstraction
From Winston (1994) On to C++ (with C++ examples replaced with Python code)

Procedure abstraction increases your efficiency and makes your programs easier to maintain.

When you move computational detail into a function, you are said to be doing **procedure abstraction**, and you are said to be hiding the details of how a computation is done behind a **procedure-abstraction barrier**.

The key virtue of procedure abstraction is that *you make it easy to reuse your programs*. Instead of trying to copy particular lines of program, you—or another programmer—arrange to call a previously defined function.

Another virtue of procedure abstraction is that *you push details out of sight and out of mind*, making your programs easier to read and enabling you to concentrate on high-level steps.

Another virtue of procedure abstraction is that *you can debug your programs more easily*. By dividing a program into small, independently debuggable pieces, you exploit the powerful divide-and-conquer problem-solving heuristic.

Still another virtue of procedure abstraction is that *you easily can augment repetitive computations*. For example, you can start with a function that computes the volume of a railroad car:
def box_car_volume(l, w, h):
    return l * w * h

And you can easily add a line that displays the volume every time that the volume is computed:

```python
def box_car_volume(l, w, h):
    print ('The volume is:', l * w * h)
    return l * w * h
```

Thus, you do not need even to bother to find all the places where a volume is computed, because you need only to change the `box_car_volume` function's definition.

Another virtue of procedure abstraction is that you easily can improve how a computation is done. You might decide, for example, that it is wasteful for your `box_car_volume` function to multiply out height, width, and length twice. Accordingly, you might decide to do the computation just once, using a variable, named `result`, to hold on to the value:

```python
def box_car_volume(l, w, h):
    result = l * w * h
    print ('The volume is:', result)
    return result
```

Again, you do not need to bother to find all the places where the volume is computed using the `box_car_volume` function; you need only to change the `box_car_volume` function's definition.
Still another virtue of procedure abstraction is that *you easily can change the way a computation is done*. If you decide to measure size the way the U.S. Postal Service measures size, instead of by computing volume, you easily can redefine `box_car_volume`.

```python
def box_car_volume(l, w, h):
    result = l + w + h
    print ('The post-office size is:', result)
    return result
```

At this point, of course, you should also rename your function to bring the name into line with what the function does.

**In Summary**

- *Procedure abstraction* hides the details of computations inside functions, thus moving those details behind an abstraction barrier.

- You should practice procedure abstraction so as to take advantage of the following benefits:
  - Your programs become easier to reuse.
  - Your programs become easier to read.
  - Your programs become easier to debug.
  - Your programs become easier to augment.
  - Your programs become easier to improve.
  - Your programs become easier to adapt.

From Winston (1994) *On to C++* (with C++ examples replaced with Python code)
"Abstract data types are a big deal. They lead to a different way of thinking about organizing large programs. When we think about the world, we rely on abstractions. In the world of finance people talk about stocks and bonds. In the world of biology people talk about proteins and residues. When trying to understand these concepts, we mentally gather together some of the relevant data and features of these kinds of objects into one intellectual package. For example, we think of bonds as having an interest rate and a maturity date as data attributes. We also think of bonds as having operations such as "set price" and "calculate yield to maturity." Abstract data types allow us to incorporate this kind of organization into the design of programs."

"Data abstraction encourages program designers to focus on the centrality of data objects rather than functions. Thinking about a program more as a collection of types than as a collection of functions leads to a profoundly different organizing principle. Among other things, it encourages one to think about programming as a process of combining relatively large chunks, since data abstractions typically encompass more functionality than do individual functions. This, in turn, leads us to think of the essence of programming as a process not of writing individual lines of code, but of composing abstractions."
“The availability of reusable abstractions not only reduces development time, but also usually leads to more reliable programs, because mature software is usually more reliable than new software. For many years, the only program libraries in common use were statistical or scientific. Today, however, there is a great range of available program libraries (especially for Python), often based on a rich set of data abstractions, as we shall see later in this book.”

From Guttag (2013) *Introduction to Computation and Programming Using Python*
How to read for retention

Education is concerned with learning. Learning is the process of getting knowledge into your long-term memory such that you can retrieve it when you need it.

To learn something, you need to:
1. Acquire it. Get it into your brain.
2. Retain it. Keep it in your brain.
3. Retrieve it. Get it out of your brain.

You have two kinds of memory:

<table>
<thead>
<tr>
<th>Short-term memory (STM)</th>
<th>Long-term memory (LTM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very small, 7±2 items</td>
<td>Infinite</td>
</tr>
<tr>
<td>Fast in</td>
<td>Slow in</td>
</tr>
<tr>
<td>Fast out</td>
<td>Slow out</td>
</tr>
<tr>
<td>Rapid decay</td>
<td>Slow decay</td>
</tr>
<tr>
<td></td>
<td>Interference</td>
</tr>
</tbody>
</table>

There are two kinds of knowledge:

<table>
<thead>
<tr>
<th>Procedural</th>
<th>Declarative</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to do things</td>
<td>Facts about the world</td>
</tr>
<tr>
<td>Ex: How to start PyCharm</td>
<td>Ex: We are using Python 3</td>
</tr>
</tbody>
</table>

Both can be stored in your LTM.
Getting knowledge into LTM is very time consuming. All knowledge must pass through your STM before it can go into your LTM. Getting knowledge into your LTM requires rehearsal. Rehearsal is the process of actively engaging with material using your STM, during which time some of this material will very slowly make its way into your LTM. Learning some thing after a single trial, or a single exposure, occurs very rarely, typically only when it is easy knowledge with a very high reward or penalty. Such as “The gold is at milepost 368.” “Single trial learning” occurs very rarely.

Rehearsal is required. It can take may forms, such as:
• Solve problems that require you to repeatedly move new, important knowledge in and out of your STM.
• Actively engage with material such as through taking notes during or after a lecture, discussing topics with others.
• Work with instructors who can accurately tell you when you are right or wrong.
• Actively read, which includes taking notes and doing practice problems.

How should you take notes when you are reading?
• Summarize mains point in your own words, first aloud, and then write it down.
• If a concept can be illustrated visually, get a pencil and paper and draw it.
• There is some value in reading text from a book and writing it down with your hands. You have to hold the text in your head. You have to send it to your hand.
• Digitally copying and pasting text from a book into a computer document will not get knowledge into your LTM.

Where and how do you take these notes? Some options:
• If it is your book, and it is paper, in the book itself.
• In a paper notebook.
• On individual pieces of paper that you then scan in.
• For PDFs and eBooks, a separate single word processing file with text notes. Not ideal, but the best solution I have come up with. Make sure the file format will last.

Regardless of how you take your notes, it is critical that it engages your STM in a rehearsal process.
Lots of time is required. There is no shortcut. It takes 10s to 60s to get a single “chunk” into LTM. (Kieras, NGOMSL Guide.)

Revisiting material often will dramatically enhance your ability to read and retain.
If you really want to learn subject X, you need to stop peeking at Facebook every chance you get, and start peeking at subject X every chance you get. Including: immediately after you wake up, just before you go to sleep, and all other obvious times that you whip out your smartphones.
Nobel prize winner Herb Simon: ‘I cured myself of my addiction to newspapers at a very young age.’
Agenda

Object-Oriented Programming (OOP)
__ Key idea: a self-contained collection of data and methods.
__ Procedural Abstraction
__ Data Abstraction
__ Important concepts associated with OOP.
   __ (1) data abstraction,
   __ (2) encapsulation (data hiding)
   (3) inheritance
   (4) polymorphism
__ Examples of how the ideas are implemented in Python.
   __ Python syntax for creating classes and objects.
   __ Python convention for imposing data hiding.

__ Questions from Wednesday
__ New questions
__ Review Monday’s quiz
__ How to read for retention

Questions outstanding from Wednesday
__ What does adding self do?
__ How do you know when to add self?
__ How to update/reassign the value of member variables,
   __ including public vs. private
   __ Where do you initialize public vs. private?
A personalized example of when you might want to use public member functions to access private member variables.

Think of yourself as an object of the Person class.

Class Person:

Class functions:
- add_two_numbers (x, y)  # adds two numbers, does not need private data

Private member vars:
- events_in_life  # a list of all of your memories of personal events
- all_facts_about_myself  # a list of all of the facts you know about yourself
- people_to_text  # a list of everyone you want to text

Public member vars:
- shoe_color(s)  # the color(s) of the shoes you are wearing now
- hair color  # the color of your hair

Public member functions:
- get_event(self, event, who_asking)  # Other people can ask for event data.
  search events_of_last_week  # But they do not have access to all events.
- get_fact(self, fact, who_asking)  # Other people can ask for facts.
  search events_of_last_week  # But they do not have access to all facts.
- please_text_me(self, person)  # Other people can ask to be texted.
  people_to_text  # But they do not know if they made the list.

The list of all events about yourself is not available outside of the class. Public member functions have access to all private data. Class functions do not, and do not need it.

Other Person objects can ask for events or facts. Each Person searches its own internal private data and decides what data, if any, to provide.

Events might be: Have you read anything about Python in the last day? Did you drink coffee this morning? Have you ever <insert personal question here>?

Facts might be: What is your name? Cell phone number? Social security number? Your uoregon email password?

get_event( ) has access to all memories and facts, but determines how to answer the question based on who is asking for the information.
Week 5 - Wednesday, April 27, 2016

Agenda
__ Review Quiz
__ Walk through some code examples.
__ Midterm topics

Midterm Topics

Materials that will be covered:
• Everything in the class lecture notes from the CIS course website, regardless of whether the material was discussed in class.
• All of the quizzes, which are reviewed in the lecture notes on the class Canvas website. But you need to thoroughly understand why each answer is correct, so you can answer similar but different questions, and so that you can provide short handwritten answers.
• All of the assigned reading, especially ThinkPython2 Chapters 1-3, 5, 7-10, 12, 15-17.
• Programming concepts and ideas from Projects 1 and 2, such as Model-View-Controller.
• Using the PyCharm Debugger.

You should understand...
• Order of evaluation and operator precedence.
• Scope and lifetime of a variable.
• The concepts and benefits of object-oriented programming.
• How to use an operating system’s command line interface.

You should be able to write the Python code, using pen and paper...
• To read from files as was done in Projects 1 and 2.
• To process lists and text strings as was done in Projects 1 and 2.
• To create a simple command interpreter.
• To implement classes, objects, “private” member variables, and member functions.
Agenda
__ Project 3 available (not due) this Wednesday
__ Reading on Inheritance will be posted imminently.
__ Any questions on course content or topics?
__ Code walkthrough of Project 2 solution?
Accessing private variables in a class hierarchy cannot be done in any kind of straightforward way with double-underscored "private" variables.

class Animal:
    def __init__(self):
        self._num_wings = 2

class WingedThing:
    def wings(self):
        print("This wingthing has", self._num_wings, "wings.")

class Bird (Animal, WingedThing):
    def bird_wings(self):
        print("Bird has", self._num_wings, "wings.")

b = Bird()
b.bird_wings()
b.wings()
Agenda
__ When to use OOP versus Procedural
__ Exceptions
__ Reading for rest of term
__ Questions
When to use OOP versus Procedural

The overall goal is modularity, self-contained components with well-defined purposes.

Code become easier to reuse, read, debug, augment, improve and adapt. (Winston, 1994, presented in the context of procedural abstraction. It applies to modularity in general.)

What is the criteria to be used in decomposing systems into modules? This is a software design problem that goes back (at least as far as) a 1972 article by Parnas.

One simple distinction is whether the problem you are solving best characterized primarily in terms of (a) services that are made available to well-defined data structures or data sets or (b) a hierarchical data structure with services specific to the data objects?

Is the problem best characterized in terms of the verbs or the nouns?

Is the top-level goal of the problem to process data? Can the top-level process be broken down into well-defined subprocesses? Are the data input to the system common data types such as strings or integers? Is there little hierarchical relationship among the things in your problem?

Then procedual programming might be best.

Is your problem well-defined in terms of a hierarchy of concepts? Do different concepts have unique data characteristics and services associated with them? Is there a natural hierarchy to the concepts? Then object-oriented programming might be best.

Data abstraction. Decompose around the nouns and verbs.
Other considerations:
GUI programming lends itself to OOP. There is a natural hierarchy of screen objects with associated services. Some programming languages do not support OOP extremely well. Python, for example.
Exceptions
This lecture is partly derived from Dave Kieras’ EECS 381 lecture on the same topic.

Exceptions solve a programming problem
Code and system inputs can (and will) have errors, but adding special code in the normal flow of control to handle all possible errors greatly complicates the code.

A traditional way of checking for errors using return values:
```python
def main():
    # Call a function. If it returns True, there was an error.
    if f():
        print ('Error')
        return 1 # Exit code indicating a problem.
    return 0 # Exit code indicating no problem.

def f():
    if g():
        return True # Indicate error.
    return False # Indicates no problem.

def g():
    if h():
        return True # Indicate error.
    return False # Indicates no problem.
```

The downsides of this approach:
You lose the return value to error checking.
The code for looking for problems is mixed in with the code that forges ahead doing whatever work needs to be done.
Not a clear division of labor. The code gets kind of sloppy. This is all you were trying to do, presuming all went well:

```python
def main():
    f()
def f():
    g()
def g():
    h()
```

The solution is to provide a separate flow of control for error situations.

Write the majority of the code assuming there will be no problems. Write a little code that checks for trouble.

The basic syntax:

```python
try:
    # Write your code here, assuming all goes well.
except:
    # Write your error-handling code here.
```

Example:

```python
L = []
try:
    print (L[0])
except:
    print ("List index out of range")
```
How exceptions works

The code proceeds normally but if an exception is encountered, the stack is “unwound” (a technical term) up to the try block that checked for the exception. The “except” part of the try block is executed.

“Unwinding the stack” is equivalent to forcing a return from the function where the exception occurred. Control is transferred directly through every function up the stack, and each function call is popped off the stack, all the way up to the try/except block.

This cleans up all of the if/then statements that check the error codes of return values by using a separate flow of control for error states.

The initial example code can be rewritten as such:

```python
def main():
    n = 2
    print('Start h')
    print(1/n)
    print(n)

    def h(n):
        print('Start h')
        print(1/n)
        print(n)
    
    def g(n):
        print('Start g')
        h(n-1)
        print(n)
    
    def f(n):
        print('Start f')
        g(n-1)
        print(n)
```

Perkovic (2015)
try:
    f()
except:
    print (‘Error’)
    return 1 # Exit code indicating a problem.
return 0 # Exit code indicating no problem.

def f():
    g()

def g():
    h()

Let’s return to the example:

L = []
try:
    print (L[0])
except:
    print ("List index out of range")

This code will catch all possible exceptions. You generally want to check for the specific exceptions that you are interested in.

Improved example, checking for one particular exception:
L = []
try:
    print (L[0])
except IndexError:
    print ("List index out of range")
How do you know what exceptions to look for?
If it is covered by a built-in exception, look for clues by running your code in the error state, and seeing what exceptions get thrown.
But verify by studying the built-in exceptions at https://docs.python.org/3/library/exceptions.html

You can look for multiple exceptions in the same block:

```python
L = []
try:
    print(L[0])
except IndexError:
    print("List index out of range.")
except ZeroDivisionError:
    print("Divide-by-zero error.")
```

“Raising” an exception
You can also “raise”, or trigger, an exception in the try block. This causes the particular exception to occur, and it of course can be caught and processed by the try block.

```python
L = []
try:
    raise ZeroDivisionError
    print(L[0])
except IndexError:
    print("List index out of range.")
except ZeroDivisionError:
    print("Divide-by-zero error.")
```

User-Defined Exceptions
You can derive your own exceptions by inheriting from the Exception class, and then you can “raise” these exceptions when appropriate.
class World:

    # Pseudo-constructor
    def __init__(self):
        self.cars = ('audi', 'jeep')  # A list of cars that exist.

    # Drive a car if it exists.
    def drive(self, car_name):
        if car_name in self.cars:
            print(car_name, "is driving")
        else:
            raise BadCarError(car_name)

class BadCarError(Exception): pass  # Typically an empty class.

class CommandLine:

    # Pseudo-constructor
    def __init__(self):
        # Create the instance of the world
        self.w = World()

def run(self):

    # infinite command interpreter loop
    while (True):

        # Start error checking.
        try:
            print('Enter car here > ', end=''
            input_string = input()

            # either quit or...
            if input_string == 'q':
                return False
            else:
                # Try to drive the car.
                self.w.drive(input_string)

        # Process errors.
        except BadCarError as error_string:
            print('Invalid car name:', error_string)

c = CommandLine()
c.run()
A couple details

`try/except/raise` in Python = `try/catch/throw` in Java, C++

`try/except` will only catch exceptions, not logic errors.

**Pass a parameter from a “raise” to an “except”**

When you raise an exception, you can pass arguments to the except statement as follows:

```
raise BadCarError(value1, value2, ...)
```

The except clause can specify a variable after the exception name. The values are bound to the exception instance

```
except BadCarError as inst:
    ...
```

You can access with .args as in inst.args.

`__str__()` is defined to directly print the argument list, so you can simply write `print(inst)`

See Python Docs 8.3 for more info

[https://docs.python.org/3.5/tutorial/errors.html#handling-exceptions](https://docs.python.org/3.5/tutorial/errors.html#handling-exceptions)

**In Summary**

Exceptions help to separate the code for error-checking from the code for the intended flow of control.

Exceptions create a separate flow of control for error situations, which bypasses the normal flow of control.

Exceptions contribute to readability and clarity of code.
Week 9 - Monday, May 23, 2016

Agenda
__ Quiz
__ Questions on P3? Exceptions?
__ More on inheritance using super()
__ The “diamond problem” in a class hierarchy.
  __ The MRO
__ Passing parameters in exceptions.
__ How to test your code using the sample inputs/outputs.
__ Use the debugger to trace your code.
super() delegates method calls to a parent class. What happens when an object has two parent classes, and each of the two parents have the method?

```python
class FlyThing():
    def say_me(self):
        return "I am a FlyThing. 

class GasThing():
    def say_me(self):
        return "I am a GasThing. 

# The order of inheritance dictates which comes first in the MRO.
class Airplane(FlyThing, GasThing):
    def __str__(self):
        return "I am an airplane. " + super().say_me()

a = Airplane()
print (a)
```

Which say_me() method gets called?

Answer: The order in which the base classes are listed when defining the new class dictates the MRO.

MRO = method resolution order, the order in which base classes are searched for a member variable or function. You can see the MRO for an object by viewing its __mro__. 
The “diamond problem” in a class hierarchy.

If D inherits from B and C, and B and C inherit from A, ambiguity can arise if B and C both override a method in A. Which version should D use?

What is A in our example above?

Python addresses this problem by creating a list of all of the classes that a class inherits from. The lists dictates the MRO. The MRO for this diamond would be D, B, C, A.

Python needs this MRO in part because it is interpreted and dynamically-typed. In C++, which is compiled and statically typed, this ambiguity must be explicitly resolved in the code and verified at compile time. C++ has no MRO.
Calling super( ) with arguments

In class D, \texttt{super( )} is the same as \texttt{super(D, self)}

The \texttt{D} argument specifies \textit{after which class in the MRO} the search for the referenced attribute should begin.

If the MRO is \{D, A, B, C, object\}:
- \texttt{super(D, self)} skips the attributes of the referenced object, of type D, the same as just calling \texttt{super( )}.
- \texttt{super(B, self)} starts the search at C.
- \texttt{super(C, self)} starts the search at object.
For example:

\[
\text{return } A.f() + \text{self}.f() + \text{super()}.f() +
\]

Here is what happens:

1. If the function call is attached to a class, as in \(A.f()\),
   the interpreter looks for the function in the namespace for
   that class. Namespaces are implemented as dictionaries.

2. If the function call is attached to \(\text{self()}\), the interpreter
   looks for the function in each class listed in the \(\text{__mro__}\)
   for the current class. The \(\text{__mro__}\) is searched from left to
   right. Classes are typically listed in the \(\text{__mro__}\) in an order
   that would correspond to an upward, breadth-first, left-to-
   right search starting at the current class.

3. If the function call is attached to \(\text{super()}\), the interpreter
   does the same as in case #2, but skipping over the first
   item in the \(\text{__mro__}\), the current class.

What does the Python interpreter do? It looks for the function
using a bunch of dictionaries.

Note that cases 1 through 3 take place in the local scope.

https://docs.python.org/3/tutorial/classes.html#python-scopes-and-namespaces
The interpreter definitely does not look in the current class, and then its superclass, and then its superclass, and so on, to the topmost inherited class. This would look like this:

![Diagram](image)

The numbers indicate search order.

An attribute in class C would not be accessible. Cases #1 and #3 above are not covered.
Week 9 - Wednesday, May 25, 2016

Agenda
__ Review Quiz
__ How time works in Project 4
__ Announcements:
   __ Several addendums for Final submission.
   __ P4-Final sample i/o files are now available.
   __ Test your code using the sample inputs/outputs.
Your Questions.
del(), delete, smart pointers, garbage collection
How time works in Project 4
`del__( )`, `del`, smart pointers, and garbage collection

We have said a lot about creating objects. We have said very little about destroying objects, how to make them go away forever. 

`del__( )` is referred to as a “destructor” but it is not. It is called just before the interpreter destroys an object. Only use it for very simple things. Exceptions that are thrown in a `del__( )` are not caught, for example!

Keep it simple: [https://docs.python.org/3/reference/datamodel.html#object.__del__](https://docs.python.org/3/reference/datamodel.html#object.__del__)

(You can see objects being destroyed at [https://www.youtube.com/watch?v=w_A8htd0fEg](https://www.youtube.com/watch?v=w_A8htd0fEg))

```python
a = A()  # creates an object. a is a reference, a pointer, to it.
b = a    # creates an alias to the same object, another pointer to the same object.
The reference count for that object is now 2.
```

del a  # removes the pointer to the object, and decrements the reference count by 1.

As soon as that count reaches 0, the object is destroyed.

You delete the pointers. The interpreter destroys the object. This makes it a sort of “smart pointer”. A smart pointer in C++ give control over what happens to objects pointed-to when the pointer is deleted, the object copied, and more.

It is the default behavior for Python pointers. The object can be destroyed, and inaccessible, but can still take up memory. How does this memory get freed up?

*Garbage collection* is the process of the runtime environment, going through all of objects in memory, and freeing up the memory that is allocated but no longer used.

Note that destroying objects and garbage collection are related, but independent.
Why do you care about any of this?

1. You need to be better stewards of your environment.
   ‘Young people have very little concern regarding the conservation of resources.’ Bill Gates (circa 2005)
   What resources was he referring to?

   Memory management is something that is integral to all programming languages. It is something you will need to think a lot about if you do any serious programming. It is not discussed much in Python textbooks because of the behavior of smart pointers and garbage collection. But it is something that could cause you great pain if you ever do any serious programming with sizable data.

2. A subtle detail to the Project 4 specification.

**Model class**

*Member functions*

delete_fire( name ) - Delete from the Model and the View a fire with the passed-in name.

**Fire class**

*Member functions*

__del__( ) - Outputs to the console “Fire <name> has disappeared from the simulation.” <name> is the fire name.
Overview of Topics
Object-oriented design and implementation.
  Class hierarchies and inheritance, multiple inheritance.
  Polymorphism, data hiding, data abstraction, modularity.
  Linear data structures such as lists, stacks, queues.
Namespaces and scope.
How to systematically test modular object-oriented programs.
How to think like a computer scientist.
How to be rigorous.

This is a programming class. Computer programming is changing the world. There is very little done in any contemporary workplace, field, or discipline that does not rely heavily on programming.

• New Internet and cloud computing services continue to change the way people communicate and collaborate.
• Smart grids, smart buildings, and smart transportation are fundamentally changing the way people live.
• Most fields of science now rely heavily on computer simulation to explore and understand phenomena.
• Computer programming has dramatically changed medicine and how treatment are administered and tracked.

Computer programming is at the core of these advances. I want you to become great programmers.
Programming can be tedious but it gets a LOT easier with practice—LOTS of practice. If you can do it well, it gives you power. Almost superpower.

“Even though more than 4,600 software development jobs are unfilled in Oregon, there were fewer than 400 computer science graduates in the state in 2014 to apply for them.”

The goal of this class is to prepare you for one of these many positions, or for any position in which a thorough understanding of computer programming will permit you to advance your career and your creative ideas.

I probably spent about 40 hours of programming for each project. I probably spent the same amount time reading reliable sources on the Python programming language. (I don’t love programming. But I love building things. I suspect some good carpenters love building things but not necessarily using power tools.)

But for very, very little of that time I spent programming was there any doubt in my mind what my code was doing.
How can that be?

Good programming practices, such as:

A drive to understand exactly what is going on in your code, and *behind* your code, in the interpreter.

Rigorous use of a debugger.

Good tools, such as:

A good IDE. Look for and learn key functionality for viewing and navigating your code both statically and dynamically (not for how the IDE can give you hints).

An easy-to-use “diff”.

Lots of backups, ideally annotated. My P3 backups:

Spend a lot of time reading reliable sources.

Understand that stackoverflow.com is not a good and reliable resource. On the great occasion that I got a good idea from stackoverflow.com, I always followed up by studying reliable resources.

Don’t use programming constructs unless I fully understand how they work such as by studying a reliable source, and writing very small sample programs to rigorously confirm my understanding.

Don’t use found code unless I fully understand what it is doing (and provide attribution).

Good understanding of basic programming concepts such as flow of control, scope, data structures, inheritance, exceptions, pointers, etc.
Good understanding of behaviors and characteristics specific to the programming language and environment you are currently working with.

- lists versus tuples
- no for (i=0; i<n; i++) construct
- try/raise/except rather than try/throw/catch
- inheritance creates namespaces that are simply collections of everything inherited
- every variable is a pointer
- mutable versus immutable objects; they behave differently. all built-in numeric types are immutable.
- the method resolution order for figuring out, at run time, which member function to call.
- Python is a collection of dictionaries.

The point of the projects:

Code should often first be described in human-readable text that is independent of the code, and then written as code. If I had given you a generic assignment such as “Build a simulation with robots and fires, and the robots put out fires” you still would have needed to decide much of what is in the project specifications. But the specification would have been deeply embedded within the code, and thus much harder to understand or analyze. Addendums and clarifications will be needed for any real programming project.
What to learn from the projects in terms of design, or, why this project is a good design.

• Designs evolve and change. The project was resilient to that.
• Uses all of the potential of procedural and data abstraction.
• Uses all the potential of OOP (that is supported by Python): data abstraction, encapsulation, inheritance, polymorphism.
• Uses well-established software patterns: MVC pattern, and the Observer pattern (in which a subject maintains a list of its dependents, and notifies them of any state changes).

Why is it good design to use patterns?
(1) They are a time-proven construction technique, such as post and lintel. They work. Use them when appropriate.
(2) Experienced builders will recognize them, and immediately understand the intentions of the original builder.

• Many of the design decisions made the code easy to build upon in later projects.
• The process of development was also designed, to provide intermediary stopping points in which specific functionalities and services were completed, and could be tested.

How to learn how to do design
You learn how to design first by studying and working in the context of, and building, other people’s good designs.
For example, MVC is just one software architecture (high level description of a program’s major components and how they interact). But understand that it will not be appropriate for every software project you are asked to design.

At this point, you should be able to take a description of any of the projects, and write the code with no reference materials whatsoever. If you can, you have mastered the Python programming aspects of the class. That is how good programmers write their code. First they learn the their craft. Then they exercise it.

You should also have a much easier time learning Java in CIS 212 if you have mastered the concepts such as inheriting versus overriding, making member variables private, and throwing exceptions. I have tried to frame many of the discussions of how Python works in these sort of general programming concepts and terms.

To be a great scholar, you must improve how you learn. Sometimes you need to just sit quietly with ideas for them to sink in. You might not learn well if you try to intensively “multitask” while learning. Thank you for refraining from your use of computers in the classroom.

Try to be conscious of how your computer and smartphone usage affects your learning and experience, and the learning and experiences of others. You might learn better if you take notes by hand.
Trying out little code snippets on your own in class might not help you to learn big concepts.
Yes, it was a hard class. I encourage you to question the value of classes that are not as hard as this one if your goal is to be a great programmer.