# Table of Contents

(Click on a day to go to that lecture.)

<table>
<thead>
<tr>
<th>Week 1 - Monday March 28, 2016</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1 - Wednesday March 30, 2016</td>
<td>10</td>
</tr>
<tr>
<td>Week 1 - Friday April 1, 2016</td>
<td>11</td>
</tr>
<tr>
<td>Week 2 - Monday, April 4, 2016</td>
<td>15</td>
</tr>
<tr>
<td>Week 2 - Wednesday, April 6, 2016</td>
<td>17</td>
</tr>
<tr>
<td>Topic: Coding standards</td>
<td>18</td>
</tr>
<tr>
<td>Topic: Python operators and operations</td>
<td>19</td>
</tr>
<tr>
<td>Topic: The scope and lifetime of a variable</td>
<td>20</td>
</tr>
<tr>
<td>Topic: Keywords, Built-In Functions, Types</td>
<td>22</td>
</tr>
<tr>
<td>Topic: Command line prompt</td>
<td>24</td>
</tr>
<tr>
<td>Topic: Lists and Tuples</td>
<td>27</td>
</tr>
<tr>
<td>Topic: Looping and Control Constructs</td>
<td>28</td>
</tr>
<tr>
<td>Week 3 - Monday, April 11, 2016</td>
<td>33</td>
</tr>
</tbody>
</table>
Week 1 - Monday March 28, 2016

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

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 `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 `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
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
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.
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

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

---

### Read for Monday’s quiz

From docs.python.org:

1. 4.7.1  
   ![Link to 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
   ![Link to specification](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
   ![Link to specification](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 \textit{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:
  \begin{verbatim}
  import sys # for argv
  \end{verbatim}
• If you do not know what this is, look it up in a reliable source. You know what \texttt{import} is from CIS 210 (\texttt{import math}). If not, read in the reliable sources. Look up sys module in the index of \url{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:
Topic: Coding standards

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}`
- `expr1 + expr2 * (expr3 - expr4)`
- `expr1(expr2, expr3, *expr4, **expr5)`
- `expr3, 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:

(Figure 17-1 from Lutz, 2013)

<table>
<thead>
<tr>
<th>Built-in (Python)</th>
<th>Names preassigned in the built-in names module: <code>open</code>, <code>range</code>, <code>SyntaxError</code>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global (module)</strong></td>
<td>Names assigned at the top-level of a module file, or declared <code>global</code> in a <code>def</code> within the file.</td>
</tr>
<tr>
<td><strong>Enclosing function locals</strong></td>
<td>Names in the local scope of any and all enclosing functions (<code>def</code> or <code>lambda</code>), from inner to outer.</td>
</tr>
<tr>
<td><strong>Local (function)</strong></td>
<td>Names assigned in any way within a function (<code>def</code> or <code>lambda</code>), and not declared <code>global</code> in that function.</td>
</tr>
</tbody>
</table>

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](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:

auto  double  int  struct
break  else   long  switch
case   enum   register  typedef
char
const  float  short  unsigned
continue  for  signed  void
default  goto  sizeof  volatile
do  if  static  while

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:

False  class  finally  is  return
None   continue  for  lambda  try
True   def     from  nonlocal  while
and    del     global  not  with
as     elif    if    or    yield
assert else    import  pass
break  except  in    raise

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()
bin()     eval()    int()     open()    str()
bool()    exec()    isinstance() ord()     super()
bytearray() filter()   issubclass() pow()     tuple()
bytes()   float()   iter()    print()    type()
callable() format()   len()     property() vars()
chr()     frozenset() list()    range()    zip()
classmethod() getattr()  locals()   repr()    __import__()
delattr()  hasattr()  max()     round()    __import__()
delattr()  hash()     memoryview() set()
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):

```plaintext
>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:

```plaintext
>pip3 install numpy
```

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

```plaintext
>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:

```python
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.

```python
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.

```
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”.
Topic: Looping and Control Constructs

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

![Diagram of a ball rolling down ramps with switches deciding the path.]

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

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

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)
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.
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.
__ 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.
OOP

**Important concepts**
Data types
(Namespaces)
Classes
Objects
Instantiation
Member variables, attributes
Class variables
Member functions
Getters and Setters
Data hiding
Encapsulation
Base class
Derived class
Constructor / Destructor / Initializer

**More advanced:**
Inheritance
Polymorphism

Using the debugger:
Day 3: How I be productive. Keyboard, mouse, headphones, two large displays, display height. Show them numerous tasks completed in PyCharm.

Namespaces.
Scope

Keeping track of your variables as you execute your code. Scope starts to matter as well, such as the contents of lists in different scopes.

This leads to using a debugger.

**Future topics:**
- Tracing values in loops.
- How to use the debugger.
BNF grammar notation

family ::= P (P) | p (p)*
P ::= M | F | N
p ::= m | f | n

family ::= M F (m | f)*

https://docs.python.org/3/reference/introduction.html#notation

Please disregard all notes below.

__ Read from docs.python.org:
(1) https://docs.python.org/3/tutorial/classes.html
stopping at the start to 9.3.

You need to become power users.
Stop using your mouse or keypad. Use keyboard shortcuts for everything.
Get a large external monitor, 2560x1440. These can be had for as little as $350. You need this to be a serious programmer.
Set up a good ergonomic work environment

Set up an ergonomic and quiet workspace, with a comfortable chair with lumbar support and elbow supports. Position your keyboard

18–28 in. (45–70 cm)

Top of the screen at or slightly below eye level (You may need to adjust the height of your display by raising or lowering your work surface.)

Shoulders relaxed

Forearms and hands in a straight line

Forearms level or tilted slightly

Lower back supported

Thighs tilted slightly

Screen positioned to avoid reflected glare

Clearance under work surface

Feet flat on the floor or on a footrest

More information about ergonomics is available on the web:

www.apple.com/about/ergonomics


You need to become proficient system administrators.
Topics

Day 1: Project 1 introduction

Day 2: Quiz and Syllabus

Mutable versus immutable.