Introduction to: Computers & Programming: Variables, Functions, Modules & Scripts

Adam Meyers
New York University
Outline

• More function IO:
  – Assignment of multiple values to variables
  – Keyword parameters
  – Returning multiple values from a function
• Filename – just one more type of Identifier
• Two types of .py files
  – Programs (scripts)
  – Library Files (modules)
• Loading programs
  – To run
    – Into Idle for editing
• Loading and using modules
• Writing a python script
• Writing a python module
Multiple Variable Assignment

• It is possible to use one assignment with multiple variables
  
    chicken, egg, one, two = 'chicken', 'egg', 1 2

• The variables are assigned in order
  – chicken is assigned the value 'chicken'
  – egg is assigned the value 'egg'
  – etc.

• This is equivalent to writing several assignment statements
Parameters of Functions

- Local variables represent input to functions
  - def print_pattern(char1, char2, len1=1, len2=2, repeat=3):
    for rep in range(repeat):
      print(char1*len1, char2*len2, sep='')
      if len2 > 1:
        if len1 > len2:
          len1 = len1 - 1
          len2 = len2 + 1
        else:
          len1 = len1 + 5
      else:
        len1 = len1 + 1
        len2 = len2 + 1
  - print_pattern('*', '+', len1=5, len2=4, repeat=10)

- Normal Parameters are required
- Keyword Parameters include defaults
- Parameter values can be changed because they are local variables
Return Multiple Values

- A function can return multiple values
- `def div_w_remainder(dividend, divisor):
  whole = dividend // divisor
  remainder = dividend % divisor
  return(whole, remainder)
- `cookies_each, xtra_cookies = div_w_remainder(50, 20)`
Filenames are identifiers

- Files used in python include: program, input and output files
- This lecture only concerns program files (modules and scripts)
  - These should have filetype .py
- When creating program files, it is advisable to follow the same principles as with variables and function names:
  - Use letters, numbers and underscore
  - Begin with a letter
  - Pick filenames that tell you something about the file contents
  - Not abiding by these rules can result in bugs, e.g., when loading files with “import”
Finding Files on Your Computer

- Lets find scripts in the `turtledemo` directory on your computer
  - Apple: `/Library/Frameworks/Python framework/Versions/3.4/lib/python3.4/turtledemo`
  - Windows: `C:\Python34\Lib\turtledemo`

- Searching for `turtledemo` (or any python file)
  - For Apple to find it, you must set “kind” in the search window to include “systems” files
  - For Windows, you must search from `C:`
    - This worked on my Windows computer, if you have difficulty, you may need to change settings to search for more types of files (and search `C:`, not a subdirectory)

- Next 2 slides: A quick introduction to file systems on your computer
Computer Files Form a Directed Graph

/  

Applications  Library  Home  System  var

........

Application Support  Frameworks  Quicktime

Python Framework

-  

-  

turtledemo

bytedesign.py  colormixer.py  yinyang.py
The Structure of File Systems

- Special Symbol for root of graph and separator
  - Unix-compatible (linux, Apple, …) = /
  - Windows (DOS) only = \n- The root corresponds to a either a disk drive or all disk drives
- For Windows, C: is usually the “main” disk drive
- The terminal nodes (nodes with no children) of the graph are regular files
- The internal nodes (nodes with children) of the graph are “directories” or “folders”, i.e., they refer to sets of files
- It is possible to have files or directories “shared” by multiple parents
  - These are handled via “file linking” or “short cuts”
- In many operating systems, certain classes of files are hidden from the users: hidden files, system files, etc.
  - This reflects a philosophy that users should be protected from understanding the details of what is going on
  - Computer science professors don't share this philosophy
  - We recommend making all files visible, so you can understand how your computer works and so you can find program files
What is a Script?

• A script is a file that contains a single program
  – Functions defined in other files (modules) are loaded with the keyword `import`

• Scripts can be run in several ways:
  – Double-clicking if the `python3 launcher` is the default program for files of type `.py`
  – By using `open with` and choosing the `python3 launcher`
    • Recommended as python2 is likely to be the current default
  – Typing `python3 filename.py` on a command line (in a shell)

• These can also be loaded, edited and run in IDLE

• For example, we will use the colormixer (`colormixer.py`) script from the `turtledemo` directory
  – We can run it
  – We can open it with Idle, examine the code and run it
The Example colormixer

• When we run it
  – There are 3 sliders corresponding to red, green and blue
    • primary colors for light (magenta, yellow and cyan are primary colors for pigment)
  – Moving the sliders show the result of mixing these colors differently

• When we look at the code
  – This program imports parts of the *turtle* module
  – It creates some of its own object types (details omitted)
  – The real action is in the *main* function
The Example colormixer

• When we run it
  – There are 3 sliders corresponding to red, green and blue
    • primary colors for light (magenta, yellow and cyan are primary colors for pigment)
  – Moving the sliders show the result of mixing these colors differently

• When we look at the code
  – This program imports parts of the *turtle* module
  – It creates some of its own object types (details omitted)
  – The real action is in the *main* function
The code in colormixer

- By convention, many programmers name their principle function that calls all others \textit{main}.
- The program first defines some variables and creates objects of two types:
  - \textit{screen} and \textit{colorturtle} (the programmer's modified version of a type called \textit{turtle})
  - And writes the message “Drag” to label the window
- The method \textit{shift}, part of the definition of \textit{colorturtle}, maps the y position of a turtle to a numerical value
- The function \textit{setbgcolor} sets the red, green and blue components for the background color of the screen at the end of each call to the function \textit{shift}.
  - These component values are based on the y positions of the red, green and blue turtles
What is the turtle module?

- A file called turtle.py
  - `import turtle` loads this in python
  - `help(turtle)` lists the various functions, variables and objects that are part of the turtle module

- History
  - Turtle Graphics was originally implemented as part of the **LOGO** language
  - To this day, there are implementations for teaching young children about programming (e.g., Microworlds)
  - The turtle module is a python implementation of this environment
The Basic Idea behind Turtle Graphics

• Do graphics by creating 'turtles'

• A turtle is an object on a Cartesian Plane
  – The turtle can look like a turtle, but need not
  – A Cartesian Plane is a grid as in High School Geometry
    • Vertical lines are represented as: X = -1, X = 0, X = 1, etc.
    • Horizontal lines are represented as: Y = -1, Y = 0, Y = 1, etc.
    • Points are (X,Y) pairs where X indicates how far to the left or right and Y indicates how far up or down, e.g., (1,1) is located diagonally up from the middle (0,0)

• Turtles have pens which write when the pen is down, but don't when the pen is up

• The ink color of the pen can be changed by setting their R,G,B values
Basic Components of Turtle Graphics in Python (and elsewhere)

- Object types: **Turtle** and **Screen**
  - In effect, this adds to our list of data types
    - integer, string, float, Turtle, Screen, …
  - These are initialized using functions with no arguments
    - `turtle.Turtle()` and `turtle.Screen()`
    - Use 'turtle.' prefix for commands from the turtle module
      - Or 'math.' for commands from the math module, etc.

- Simple commands that are connected to the Turtle object using dot notation
  - `fd(NUM)` – moves forward NUM units (i.e., moves forward from the turtles' point of view)
  - `left(DEG)` and `right(DEG)` – pivot left/right DEG degrees
  - `pd()` and `pu()` – put pen down (to draw) and up (to stop)
A Simple Turtle Graphics Example

• Loading module, creating a screen and a turtle
  import turtle
  my_screen = turtle.Screen()
  my_turtle = turtle.Turtle()

• Putting the pen down and drawing a square
  my_turtle.pd()
  my_turtle.fd(100)
  my_turtle.left(90)
  my_turtle.fd(100)
  my_turtle.left(90)
  my_turtle.fd(100)
  my_turtle.left(90)
  my_turtle.fd(100)
  my_turtle.left(90)
  my_turtle.fd(100)
Drawing a 2$^{\text{nd}}$ Square Under the 1$^{\text{st}}$ One

```python
my_turtle.pu()
my_turtle.fd(100)
my_turtle.pd()
my_turtle.fd(100)
my_turtle.left(90)
my_turtle.fd(100)
my_turtle.left(90)
my_turtle.fd(100)
my_turtle.left(90)
my_turtle.fd(100)
```
Modules, aka, Library Files?

- Modules are files of functions and variables
  - Designed to be incorporated in other programs
  - Typically on a single theme (math, graphics, astronomy, ...)
  - Some modules are built in, i.e., installed with Python
  - You can download or write others yourself

- To load a module
  - 'import module_name'
    - You can use functions, global variables and objects
      - Use dot notation, e.g., module_name.function()
  - 'from module import functionX' (or objectX)
    - Use functionX without dot notation
    - Overwrite function and global variable definitions if they have the same name
  - 'from module import *' – same as above, except import everything
Modules

- Example (the math module)
  
  ```python
  import math
  help(math)
  math.ceil(5.1)
  help(math.ceil)
  ```

- The 'help' function
  - Lists variables, functions, methods, etc. for a module
  - Also gives function definitions

- Use 'dot' notation for module variables/functions

- Alternatively: from module_name import *
  - Let's you drop the dot notation
  - Can cause problems (name conflicts)
four-squares.py Script

- Uses 2 modules: *turtle* and *time*
- Encapsulates square drawing as a single function which we call 4 times
- The square drawing function puts down the pen; moves forward and turns left three times each; and then puts down the pen
- The main function draws four squares, (redundantly) puts down the pen in between squares and sleeps for 15 seconds at the end
  - Note that the redundancy insures that the function works properly in all environments
four-squares.py Script 2

• The comments suggest ways to modify the program
• Turtles come in several different shapes
  – (turtle.getshapes() will list them)
  – 'turtle' is in fact one of the possible shapes
  – This is being called with a keyword argument shape='turtle'
    • Args identified by name, rather than order
• colormode(255) allows colors to be set in combinations of Green, Yellow and Blue on a scale from 0 (no color) to 255 (saturated)
• The package is very detailed. It has its own manual: http://docs.python.org/py3k/library/turtle.html
Is there a way to improve the code?

- Do you notice any redundancy in the `draw_turtle_square` function?
  - Is there any way that a loop could be used to simplify the code?

- Is there any way we could generalize this function so that we could use one function for drawing, not just squares, but other shapes as well?
Revisit For Loop Checkerboard Program

• Re-examine Code from loop-examples lecture
Summary

• Identifiers in Python include:
  – variables
    • local variable names apply inside of functions
    • global variable names apply whenever they are not clobbered by a local variable name
  – functions – encapsulate sets of commands
  – program files
    • Scripts – special purpose
    • Modules (aka library files) – reusable code

• Intro to Graphics:
  – typically use X,Y coordinates for points on a plane
  – use some sort of RGB encoding for color
Summary

• Identifiers in Python include:
  – variables
    • local variable names apply inside of functions
    • global variable names apply whenever they are not clobbered by a local variable name
  – functions – encapsulate sets of commands
  – program files
    • Scripts – special purpose
    • Modules (aka library files) – reusable code
• Intro to Graphics:
  – typically use X,Y coordinates for points on a plane
  – use some sort of RGB encoding for color
Homework Part 1

- Read Chapter 7
  -(Note: We will Read Chapter 6 later in the term)
- Do Module 7
  - http://cs.nyu.edu/elearning/CSCI_UA_0002/module07.php
- Do Quiz 7 in NYUClasses
Homework 2: Part 2
Write a program

• Use the turtle module to draw a picture.
  – Several turtle programs are on the class website
  – See: http://docs.python.org/release/3.4.3/library/turtle.html

• Include at least a stick figure in your picture.
  – If you want the figure to have a round head use
    \texttt{my_turtle.circle(NUMBER)}
    where NUMBER is the radius of the circle (e.g., 50)

• Include functions that encapsulate code for reuse
  – Examples: draw_left_arm, draw_left_leg, draw_stick_figure
  – This makes it easy to draw multiple stick figures and different types of stick figures. It also makes your code easier to understand
Grading Criterion

• There must be at least one recognizable stick figure
• Higher score for encapsulating code as functions: draw_hand, draw_finger, etc.
• Higher score for calling the same function more than once with different parameters
• Higher scores for creative, interesting or surprising pictures