Introduction to:
Computers & Programming
Program Files in Python:
Modules and Scripts

Adam Meyers
New York University
Outline

• 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
What is a Script?

- A script is a file that contains a single program
  - Functions defined in other files (modules) can be used (if loaded using the keyword `import`)

- Some example scripts are found in the python folder Extras. These can be run in several ways:
  - Double-clicking if the `python launcher` is the default program for files of type .py
  - By using `open with` and choosing the `python launcher`
  - Typing `python filename.py` on a command line (in a shell)

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

- The script:
  - Apple – PYTHON-DIRECTORY/Extras/Demo/turtle/tdemo_colormixer
  - Windows – PYTHON-DIRECTORY/Demo/turtle/tdemo_colormixer
  - We can run it
  - Or we can see the variables and functions loaded in it
The Example tdemo_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 (we will not discuss this in detail here)
  - The real action is in the *main* function
The Example tdemo_colormixer 2

- By convention, many programmers name their principle function that calls all others *main*.
- The program first defines some variables and creates objects of two types:
  - *screen* and *colorturtle* (the programmer's modified version of a type called *turtle*)
  - And writes the message “Drag” to label the window
- The function *shift*, part of the definition of *colorturtle*, maps the y position of a turtle to a numerical value
- The function *setbgcolor* sets the red, green and blue components for the background color of the screen at the end of each call to the function *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 (and gives the path for the turtle.py file)
  – *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., Microworl ds)
  – 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()
    - 'turtle.' prefix for commands from the turtle module

- **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
  
  ```python
  import turtle
  my_screen = turtle.Screen()
  my_turtle = turtle.Turtle()
  ```

• Putting the pen down and drawing a square
  
  ```python
  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)
  ```
Drawing a 2\textsuperscript{nd} Square Under the 1\textsuperscript{st} One

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
    - You may overwrite function 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
Summary

• There are at least 2 kinds of program files
  – Scripts or Programs
  – Modules or Library Files

• Programs usually have a very specific purpose
  – They tend not to be very flexible

• Library files tend to be reusable code
  – To incorporate into any program that needs it

• Encapsulation: If you understand what a function does, you can usually forget how it works, even if you wrote the function.

• Graphics: (a) typically use X,Y coordinates for points on a plane; (b) use some sort of RGB encoding for color
Homework Part 1

• Read Chapter 5
• Yes, I am asking you to read the chapters out of order.
• We will read Chapter 4 after Chapter 5
Homework 2: Write a program

• Use the turtle library to draw a picture.
  – I put both 4 squares programs on the website to use as models, as well as a pointer to the turtle module help menu.

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

• Include functions that encapsulate code for reuse
  – This could make it easy to (for example) draw multiple stick figures