Introduction to:
Computers & Programming:
Modules, Graphics & Random Numbers

Adam Meyers
New York University
Outline

• More about Functions
  – Optional Parameters, Variable Scope, Passing variables
• What is a Module?
• The Turtle Graphics Module
• The Random Number Module
• Problems using Turtle Graphics and Random Numbers
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
  – chicken = 'chicken'
  – egg = 'egg'
  – one = 1
  – two = 2
Parameters of Functions

• Input parameters to functions are local variables
  
  def print_character_square(length, character='*', spaces=0):
      for num in range(length):
          print(length*((' '*spaces)+character))
          for space in range(spaces-1):
              print()

• Normal Parameters are required

• Keyword Parameters include defaults

• Examples
  
  – print_character_square(5)
    • prints 5 lines with 5 asterisks on each line
  
  – print_character_square(5, character='X')
    • prints 5 lines with 5 Xs on each line

  – print_character_square(5, character='X', spaces=2)
    • Adds 2 spaces between characters and 2 blank lines between lines with Xs
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)`
Variables with Global Scope

- Global variables are valid throughout a program, both inside and outside of functions.
- Global variables defined outside of functions
  ```python
  value_of_exchange = 10
  def convert_dollars_to_dimes(dollars):
      return(value_of_exchange * dollars)
  ```
- Global variables defined within functions
  ```python
  def convert_dollars_to_dimes(dollars):
      global value_of_exchange
      return(value_of_exchange * dollars)
  ```
- In both cases, `value_of_exchange` is valid everywhere in the file where it is defined.
- It is good practice to always declare any global variables used in a function
Variables “local” to functions

• All the variables in the following function are “local” to that function
  – `def add_two_numbers(num1,num2):
    output = num1 + num2
    return(output)
  – The parameters `num1` and `num2` are local
  – The variable `output` is also local

• We can test these in idle
  – These variables are undefined in the shell
  – These variables are undefined in other functions
Bad Idea for global and local variables to have the same name

• bad idea
  – output = 100
  – def add_2_nums1(num1,num2):
    output = num1 + num2
    return(output)

• Declare global variable
  – def add_2_nums2(num1,num2):
    global output
    output = num1 + num2
    return(output)

• Or name them differently
  – final_output = 100
  – def add_2_nums3(num1,num2):
    sum = num1+num2
    return(sum)
Argument Passing: Avoids using any Global variables

• def add_nums(num1,num2):
  sum = num1+num2
  return(sum)

• def average(num1,num2):
  sum = add_nums(num1,num2)
  output = sum/2
  return(output)
Modules

• Module = package of variables and functions with a single purpose
  – help(module_name) → inventory of variables/functions in module
• Modules are called packages or libraries in other languages
• Modules We Used Before Today
  – math – various math functions and variables
    • math.pi
    • math.floor(number)
    • math.ceil(number)
  – time – various time related functions
    • time.sleep(seconds) – used in clock program
    • time.time() – number of seconds since January 1, 1970
• Modules we will cover in more detail today
  – turtle – graphics functions introduced in the modules
  – random – functions relating to pseudo-random numbers
Importing Modules

• import math
  – globally (beginning of file) or locally (beginning of function)
  – module components are accessed with dot syntax
    • Example: `math.ceil(5.9)`
  – math module clearly separated from components of your program

• from module import ceil
  – component is part of your program, used without dot syntax
    • Example: `ceil(5.9)`
  – OK, as long as you don't define `ceil` in your code as something else

• from math import *
  – All components of math can now be used without dot syntax
  – Be careful to that no 2 functions/variables have the same name
    • Example, don't use `pi` as variable name for `price index`
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 of Turtle Graphics
  – Originally implemented as part of the *LOGO* language
  – Implementations used for teaching young children about programming (e.g., Microworlds)
  – Turtle module = python implementation
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 positions are represented as: X = -1, X = 0, X = 1, etc.
    • Horizontal positions 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
• The turtle module is object oriented (discussed more at end of semester)
  – Objects are types (like integer, boolean, etc.) that are defined in the module
  – Objects are defined along with special functions called methods
    • Example: my_turtle.fd(100) → moves the object my_turtle forward 100 pixels
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)
  - $left(DEC)$ and $right(DEC)$ – 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)
```
Turtle Functions in Modules_and_scripts.py

- **setup_turtle**
  - creates my_turtle and my_screen (like previous slide)
    - If does not already exist
  - resets screen if already exists
    - Searches for variable name in list
    - globals() – list of global variables
    - locals() – list of local variables

- **draw_turtle_square** (and **run_draw_turtle_square**)
  - same as previous slide

- **draw_turtle_square2** – simplification of draw_turtle_square

- **regular_polygon** – generalizes square to any number of sides
How much should a turtle turn?

- Angle in run_regular_polygon
  - 360/sides
  - Triangle = 120
  - Square = 90
  - Pentagon = 72
  - Hexagon = 60
  - Etc.

- Why doesn't it turn 60 degrees for a triangle?

- Imagine that you are the turtle
  - How much do you have to turn to draw each side?
  - To complete the triangle, how many degrees should you turn total?

- The turtle is turning around the outside, not the inside of the triangle
One Way to Describe a Checkerboard

• A Checkerboard is an 8 X 8 square with alternating colors, e.g., red and black.
• A Checkerboard can be broken down into 4 bars, each a 2 X 8 bar of alternating colors.
• A 2 X 8 bar of alternating colors can be broken down into 4 composite squares, each consisting of 2 X 2 small squares.
A Pictoral Description

X 4 =

X 4 =

X 4 =

X 4 =
One Way A Turtle Can Draw a Checkerboard?

- The turtle can draw one square and it could fill in with a color of our choice.
- The turtle can make four such squares next to each other, forming a composite square.
- It can make four composite squares next to each other, to form a bar.
- It can make four such bars, one under the other to form a checkerboard.
Intro to: Computers & Programming
Variables, Functions, Modules & Scripts
CSCI-UA.0002
for_loop_checkerboard.py 1

• Basic setup
  – import turtle
  – my_screen = turtle.Screen()
  – my_screen.setup(0.5,0.75,0,0)
    • width, height, startx, starty
  – my_turtle = turtle.Turtle(shape='turtle')

• draw_colored_turtle_square
• draw_4_black_and_red
• draw_4_black_and_red_4_times
• make_checkerboard
• Setup and then do something 4 times
  – Building block
  – Move in between blocks

• draw_colored_turtle_square
  – Setup: set colors and begin to fill
  – Repeated Steps:
    • put the pen down, move forward, turn left
  – Pick pen up and fill in color

• draw_4_black_and_red
  – Setup: initialize fill color and pen color
  – Repeated Steps: change fill color,
    draw_colored_turtle_square, turn right, move forward
for_loop_checkerboard.py 3

- **draw_4_black_and_red_4_times**
  - repeated steps:
    - **draw_4_black_and_red**
    - **move forward**

- **make_checkerboard**
  - Setup: set turtle speed
  - Repeated Steps:
    - **draw_4_black_and_red_4_times**
    - Turn 180 degrees, move forward, turn 270 degrees, move forward turn 270 degrees
The Random Package

- Psuedo Random Number: algorithm generates number
  - algorithm approximates properties of random numbers
  - probably unrelated to usage, e.g., next digit of Pi * the number of seconds since January 31, 1970, etc.

- Random.random() – generates number between 0 and 1
- Random.randint(start,end) – generates integer between start and end (inclusive)
- Random.choice(sequence) – randomly picks item in sequence
- Random.seed() – uses the current time as a seed for pseudo random number generation
  - Random.seed(a=integer) – uses a as a seed, so psuedo-random numbers generated are predictable – useful for testing a system
In Class Program

• Use turtle and random
• Program should draw squares
• The squares should be random sizes
• The squares should be at random positions on the screen
• Useful functions
  – random.randint – to generate random integers
  – my_turtle.setpos(X,Y) – move turtle to position X,Y
  – my_turtle.pos() → returns current X and Y values for my_turtle
• Assume that both X and Y must be between -350 and +350
Random Numbers → Random Events

• Die rolls: random.randint(1,6)
  – Die rolls are independent of other die rolls

• Choosing independent events in a list, e.g.,
  – door_states = ['opened', 'locked', 'unlocked', 'destroyed']
  – next_door = door_states.choice()

• Choosing Events that don't repeat
  – Randomly removing a card from a deck of cards
  – We will use this when we start handling lists
Probability

- Probability(Event_X)
  - Fraction representing how likely Event_X is
  - If there are N equally likely events including Event_X
  - Then Probability(Event_X) = portion of some given set of events is likely to be Event_X

- Example: given a “fair” die with 6 sides,
  - A 6 is expected to be rolled 1/6 of the time
Total of 2 Dice is between 2 and 12

<table>
<thead>
<tr>
<th>Die 1</th>
<th>Die 2</th>
<th>Total</th>
<th>Die 1</th>
<th>Die 2</th>
<th>Total</th>
<th>Die 1</th>
<th>Die 2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>
# Probability of Totals of 2 Dice

- Fraction of combinations totaling 2, 3, \ldots, 12 are expected probabilities of these rolls

<table>
<thead>
<tr>
<th>Roll</th>
<th>Combos</th>
<th>Decimal</th>
<th>Roll</th>
<th>Combos</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1/36</td>
<td>.028</td>
<td>8</td>
<td>5/36</td>
<td>.139</td>
</tr>
<tr>
<td>3</td>
<td>2/36</td>
<td>.056</td>
<td>9</td>
<td>4/36</td>
<td>.111</td>
</tr>
<tr>
<td>4</td>
<td>3/36</td>
<td>.083</td>
<td>10</td>
<td>3/36</td>
<td>.083</td>
</tr>
<tr>
<td>5</td>
<td>4/36</td>
<td>.111</td>
<td>11</td>
<td>2/36</td>
<td>.056</td>
</tr>
<tr>
<td>6</td>
<td>5/36</td>
<td>.139</td>
<td>12</td>
<td>1/36</td>
<td>.028</td>
</tr>
<tr>
<td>7</td>
<td>6/36</td>
<td>.167</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
random_trials_with_2_dice

- Program Simulates any number of dice rolls
- Compares frequencies with expected frequencies
- As number_of_trials increases, the results are closer and closer to expected probability
- Try with 100 rolls, 1000 rolls, 10000 rolls, …
- Law of Large Numbers:
  - More trials → Average is Closer to Expected Value
Applications in Society: Randomness and Probability

- Choosing winners and losers without objective criteria
  - Lotteries: jobs, housing, draft, …
  - Drawing straws, rock/paper/scissors, odds & evens, flipping coins, …
  - Games of chance: cards, dice, etc.

- Fortune Telling, e.g., Tarot Carts
  - Illusion of Divination is reinforced when randomly chosen selection makes correct prediction

- Probability in Science and Technology
  - Predicting future events based on statistics from past events
  - Statistical Techniques in Artificial Intelligence
Pointillism Application for Graphics

• Pointillism:

• draw_blurry funcs in Modules_and_scripts.py
  – Randomly make dots near target locations to give the effect of fuzzy lines.

• Other Unimplemented Details
  – Shading
    • Dots closer together appear darker
    • Dots further apart appear lighter
  – Changing colors of dots
    • Random changes within small range of colors, e.g., choose RGB values with max difference of +/-50
Summary

• Variables
  – Global vs Local
  – Obligatory Parameters vs Optional Keyword Parameters
  – Passing Variables

• Intro to Graphics with the turtle module
  – typically use X,Y coordinates for points on a plane
  – Uses object oriented programming
    • Each turtle and screen object is associated with various methods (functions)

• Intro to random module
  – Simulate games of chance
  – Make pseudo-random decisions in art
Homework

• https://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/hw5.html