Supplementary study guide for COMP130

John MacCormick, Dickinson College

This study guide contains brief explanations of content that is not fully explained in the assigned reading from the textbook. Anything that is explained in the assigned textbook reading will not be repeated here. When studying, please use the textbook as the primary reference. Use this study guide only for supplementary information that is not in the textbook.

# input()

The built-in input() function is covered fully in section 5.11 of the textbook, but we won’t cover that until week seven of the semester, so this supplementary study guide gives a brief explanation here.

The input() function is used to receive input from the user of the computer program while it is running, as in the following example

answer = input('What is your favorite day of the week? ')

The parameter is a string known as the *prompt*. In the above example, the prompt is 'What is your favorite day of the week? '.

The prompt will be printed. Then, the program waits to receive input from the user. The user is then expected to type their response using the keyboard, terminating the response using the Enter key. The string typed by the user is returned to the program. In the above example, the variable answer will store whatever text was typed by the user.

Here is a complete program demonstrating the use of input():

name = input('Please enter your name. ')

color = input('What is your favorite color? ')

print('Very interesting,', name, '...')

print('I wonder why', color, 'is your favorite color.')

# random.randint()

The use of random numbers is explained in section 13.2 of the textbook, but we need only a small subset of the information provided there. Facilities for using random numbers in Python are made available by importing the random module:

import random

In the first part of the semester, the only function we need is random.randint(a, b). This function returns a random integer between a and b inclusive. For example, the following program simulates rolling two 6-sided dice.

import random

roll1 = random.randint(1, 6)

roll2 = random.randint(1, 6)

total = roll1 + roll2

print('You rolled a', roll1, 'and a', roll2)

print('That gives a total of', total)

# for loops

Section 4.1 and 4.2 provide a very brief introduction to for loops. Here we provide a little more detail. The variable immediately after the for keyword is called the *loop counter*. By default, the loop counter starts counting from zero and increases by 1 each time. For example, the loop counter my\_num counts from 0 up to 5 in the following code fragment:

for my\_num in range(6):

print('This is the start of the loop body.')

print('The loop counter is currently', my\_num)

print('This is the end of the loop body.')

Because the loop counter starts at zero, it counts up to but not including the range parameter. In the above example, the range parameter is 6. The loop is executed six times. The value of the loop counter ranges from 0 to 5 inclusive, which does include exactly six numbers: 0, 1, 2, 3, 4, 5.

It is often useful to store and update information in variables inside the loop body. For example, the following code calculates the value of :

sum\_of\_squares = 0

for n in range(10):

n\_squared = n\*n

sum\_of\_squares = sum\_of\_squares + n\_squared

print('Sum is', sum\_of\_squares)

# Nested for loops

It is possible to include a for loop inside another for loop. We call this a *nested* for loop. Within a nested loop, the first for loop is called the *outer* loop and the second for loop is called the *inner* loop. The outer and inner loops should use different loop counters, as in the following example using the outer loop counter i and the inner loop counter j:

for i in range(3):

print('Outer loop counter i is', i)

for j in range(2):

print('Inner loop counter j is', j)

print('i + j is', i+j)

print('Outer loop iteration is finishing, and i is still', i)

Note the extra indentation of the inner loop body. Also note that the inner loop body in the above example is executed a total of six times. There are two iterations in the inner loop, but these two iterations are themselves executed three times by the outer loop, for a total of .

One common application of nested for loops is drawing two-dimensional grids in graphics applications. For example, the following code will draw circles of radius 4 in a rectangular grid spacing of 20 pixels, assuming we have already created a Turtle object t:

for i in range(3):

x = 20 \* i

for j in range(5):

y = 20 \* j

t.penup()

t.goto(x, y)

t.pendown()

t.circle(4)

# Constructors

We already know about datatypes. In computer programming, the word *class* can mean the same thing as *datatype*. Although there are certain technical distinctions between a class and a datatype, we will treat them as the same thing in this course. In computer programming, an *object* is an *instance* of a class. We create an object using a special kind of function called a *constructor*. The name of the constructor is always the same as the name of the class. For example, the following line of code calls the Turtle constructor inside the turtle module, creating an instance of the Turtle class:

my\_turtle = turtle.Turtle()

There are several important things to notice about this line of code:

* Python identifiers are case-sensitive, so turtle and Turtle are completely separate, different entities. In fact, lowercase turtle is the name of a module that can be imported. Uppercase Turtle is the name of a class (or datatype) that is defined in the turtle module.
* When we add parentheses after the word Turtle, it becomes the constructor Turtle() -- a special kind of function that is used to create a new object (or instance) in the Turtle class. Constructors can have parameters, but this one has a zero parameters.
* The variable my\_turtle refers to the new object that was created by the constructor. The datatype of this variable is Turtle.

# The graphics module and more constructors

Other examples of constructors are provided in the graphics module created by John M. Zelle. This module is not built in to Python. You must download the graphics.py file and save it in your current working folder before you can import this module using import graphics. A link is provided on the course webpages.

The following code creates a Point object and a Circle object.

p = graphics.Point(50, 100)

c = graphics.Circle(p, 25)

The variable p refers to an object which is an instance of the Point class, located at . The variable c refers to an object which is an instance of the Circle class. The center of this circle is located at p, and it has radius 25 units. The online documentation of the graphics module explains these constructor parameters in more detail. It also explains the constructors of several other classes such as Line, Rectangle, and Text.

# Methods and dot notation

A *method* is a special kind of function which performs an action on an object. For example, given the my\_turtle object defined earlier, we can use the forward method of the Turtle class to move this object forward 100 pixels:

my\_turtle.forward(100)

Methods are always invoked using *dot notation*, which has the format **object.method(parameters)**.

In Python, dot notation is ambiguous. It can also be used to invoke a *function* inside a module, using the format **module.function(parameters)**. For example random.randint() invokes the randint function from the random module. Another example would be turtle.clear(), which invokes the clear function from the turtle module. It does not perform an action on any specific Turtle object, but clears the entire turtle module graphics system. Contrast this with my\_turtle.forward(100), which invokes the forward method on the specific object my\_turtle.