Lab 3: Functions and Loops

CSE/IT 107

> Programs must be written for people to read, and only incidentally for machines to execute.
>
> — H. Abelson and G. Sussman

> The sooner you start to code, the longer the program will take.
>
> — R. Carlson

---

1 Introduction

The purpose of this lab is to build on last weeks lab and introduce lists and some fundamental algorithms of computer science.

Reading

Chapters 7 in Think Python: How to Think Like a Computer Scientist (Think)

Coding Conventions

Follow PEP-8 recommendations for your code.

Problems

Make sure your source code files are appropriately named. Make sure your code has a main function; use `boilerplate.py` you created in Lab 1.

Think Python is available from `http://www.greenteapress.com/thinkpython/`. Page numbers and exercise numbers refer to those found in the PDF file available at the website.

1. Think Exercise 6.1, page 52.

   Name your source code `think_6-1.py`

2. Think Exercise 6.3, page 55

   Name your source code `think_6-3.py`

3. Think Exercise 7.1 page 65. Include the recursive function from section 5.8 in Think in your program. Rewrite the function in two different ways: with a for loop and with a while loop. Name your functions `print_n_recur()`, `print_n_for()`, and `print_n_while()`

   Test each function by printing the string "It's just a flesh wound" ten times.

   Name your source code `think_7-1.py`

4. Write a program that has the computer guess what number you are thinking between 1 and 100. To get you started code is provided below. You have to write the `guess()` function. The function should check that the user entered a 'y/Y or a 'n/N' and repeat the question if they didn't. Rather than testing for both upper and lower case versions of the strings, Python has a string method `lower()` that converts the string to lower case. Similarly the string method `upper` converts strings to upper case.

   The guessing game is an example of a divide and conquer strategy known as binary search. Binary search divides the problem space in half each time.

```
def guess(a, b, n):
    pass


def main():
    a = 1
    b = 100
    n = b / 2

    # \ is the line continuation character
    print "The guessing game.\nThink of a number between \
{0} and {1} and I will guess it!".format(a, b)

    while a != b:
        # guess returns 3 values.
        # separate return values with a comma
        a, b, n = guess(a, b, n)

    print '\nYour were thinking of the number {0}'.format(a)
```

   Sample Output

```
The guessing game.
Think of a number between 1 and 100 and I will guess it!
```

```
is the number greater than 50? (Y/N) y
is the number greater than 75? (Y/N) u
Sorry, I don't understand. Please enter a Y or a N.
is the number greater than 75? (Y/N) k
Sorry, I don't understand. Please enter a Y or a N.
is the number greater than 75? (Y/N) y
is the number greater than 88? (Y/N) n
is the number greater than 82? (Y/N) y
is the number greater than 85? (Y/N) n
is the number greater than 84? (Y/N) n
is the number greater than 83? (Y/N) y
Your were thinking of the number 84
```

5. Think Exercise 7.2 (page 67 at the end of section 7.5).

   Python lets you create optional function parameters by assigning a value to the parameter inside the parenthesis. For example, if you have a function:

   `def foo(a, b=10):`

   and you called the function with `foo(6)`, `a` would be assigned 6 and b would be assigned 10, the default value. If you called the function with `foo(6, 3)`, `a` would be assigned 6 and `b` would be assigned 3.

   You can use E notation for numbers raised to powers. The `E` represents ten raised to the power of ($\times 10^b$). The `E` can be lower or upper case. For example to represent 10^7 one would write `1e7`. To represent the speed of light in meters per second squared – 299,792,458 – you would write 2.99792458e8

   You can also use E notation with negative exponents. To represent 10^{-10}, you would write `1e-10`.

   For the square root function pass epsilon as a parameter. Set the default value for epsilon to `1e-12`.

   Add a square root function, named `binary_square_root`, that uses a binary search method to find the square root. This is similar to what you did in the guessing game question. Use a default epsilon on `1e-12`

   Test you functions by finding the square root of 2, 4, 10, and 100.

   Name your source code `think_7-2.py`

6. Think Exercise 7.3, page 69

   Modify the `test_square_root()` function so it prints two additional columns. The first column is a number, a; the second column is the square root of a computed with the function from Section 7.5; the third column is the square root calculated using `binary_square_root()`; the fourth column is the square root computed by math.sqrt; the fifth column is the absolute value of the difference between math.sqrt and the

square root of the second column; and the sixth column is the absolute value difference between math.sqrt and the square root of the third column.

Use the string `format()` method. To see how the format method works try the following in a python interpreter:

```
x = 3.0000007
print '{0:.4f}'.format(x) print '{0:.4f}'.format(x) # print x with 4 decimals
print '{0:.10e}'.format(x) # print x with 10 decimals and E notation
for i in range(20):
...  print '{0:>5}'.format(i)
```

The last format will print `i` right justified with a field width of 5.

You can combine justification and other formating. To print x with E notation, 10 decimal places and right justified in a field width of 20, you could do the following:

```
print '{0:>20.10e}'.format(x)
```

For this problem, column 1 has a field width of 5 and is right justified; columns 2, 3, and 4 are floating point types printed with 10 decimal places; columns 5 and 6 are printed with 10 decimal places but use E notation.

Ignore the fact, the author doesn't print out all decimal places in his example.

Name your source code file `think_7-3.py`

7. Write a function that finds the roots of a quadratic equation $ax^2 + bx + c = 0$ using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

You are going to pass the values of a, b, c via command line options:

```
$ run quadratic a 10 b 5 c 10
```

On Moodle is a file named `quadratic.py` which provides one way to process command line arguments. Use that file as a starting point and add the functions you need to determine the roots of a quadratic equation. If there are no real roots, just print an error message; otherwise print out the quadratic equation for the given a, b, and c and the roots of that quadratic equation.

Error check for a division by zero case. If a equals zero, print an error message and exit the program. You can exit the program at anytime by calling `sys.exit([arg])`, where arg is the exit value, an integer. Exiting with 0 indicates success; any other integer indicates a failure.

8. Bill James in Slate Magazine `http://www.slate.com/articles/sports/sports_nut/2008/03/the_lead_is_safe.single.html` gives an algorithm to determine if a lead is safe in college hoops. Implement his algorithm. Pass in the lead in points, if the team has the ball, and the number of seconds remaining as command line arguments. You program should output to the user the string "Lead is safe" or "Lead is not safe".

   Write a `is_lead_safe` function. The function returns `True` or `False`. For functions that return booleans, function names often begin with `is`.

   Check that that the user entered 'y' or a 'n' for if it has the ball option. If they entered something else, print an error message and exit the program, giving a usage statement.

   When can you light the cigar, if your team has the ball and a 10 point lead? When can you light it, when the other team has the ball but you are up by 10? Place your answers as a comment at the beginning of your program.

   Name your source code file `safe_lead.py`

9. On Moodle is a program `sums.py`. At the moment it doesn't do much. You are going to modify it, so it performs a variety of functions.

   (a) Add a function `ones()` that returns a list of length `n` that are all ones.

   (b) Write a function `sum()` that finds the sum of a list and returns that value. Test your function with a list of zeros, ones, `x = [i for i in range(1, 11)]`, and `x = random_list(10, 0, 1000)`. How do you know if the last test worked?

   (c) Write a function `sum_even()` that sums only the even elements of the list. Test like you did for the sum function.

   (d) Write a function `sum_odd()` that sums only the odd elements of the list. Test like you did for the sum function.

   (e) From Project Euler, if you list all the natural numbers below 10 that are multiples of 3 or 5, you get 3, 5, 6 and 9. The sum of these multiples is 23. Write a function (`sum_multiples()` that finds the sum of all the multiples of 3 or 5 below 1000.

   (f) Legend has it that a King was so pleased with the invention of chess that he offered the inventor a great reward of gold. The inventor offered an alternative: he would get one grain of wheat on the first square of chess board, 2 grains on the second square of the chess board, four on the third, eight on the fourth, etc., doubling the number of grains on each square. Thinking this was a better deal, the ruler accepted. There are 64 squares on a chessboard. How many total grains of wheat did the ruler pay the inventor of chess? Name your function `idiot_king()`. **You have to use a for loop to solve this**. To check your answer, compare your answer with `2 ** 64 - 1`. They should be the same.

   (g) Assume a wheat grain weighs 50 mg and 1 mg = 0.0000022046 lbs. Assume also there are 60 lbs in a bushel. How many bushels did the King have to pay out? Write a function `bushels()` that determines this. In 2008, the US produced approximately 2.4 billion bushels of wheat!

10. Write a program that checks to see if a number is prime. A number greater than one is prime if it is only divisible by 1 or itself. Simplistically, to test if a number is prime you check if the number is divisible by 2 up to the square root of the number. Since you do not need to keep testing for primality once you have found it is divisible by some number between $[2, \sqrt{n}]$, use the break statement to exit the loop once you find the number is divisible by some number. Call the function `is_prime`. The function should return a boolean. Print all the primes from 2 to 1000. You can check your output against the list of primes found here `http://primes.utm.edu/lists/small/1000.txt`

    Name your source code `prime.py`

11. A perfect number is an integer whose sum of integer divisors, excluding the number itself, add up to the number. For instance 6 is a perfect number as $1 + 2 + 3 = 6$. Numbers who sum of divisors is less than the number are called deficient numbers. (10 is deficient; $1 + 2 + 5 < 10$) Numbers who sum of divisors is greater than the number are called abundant (12 is abundant; $1 + 2 + 3 + 4 + 6 > 12$). By definition, 1 is deficient.

    Write a program that finds all the abundant, deficient, and perfect numbers between 1 and 100. Print the results in a table like so:

```
1 [1] deficient
2 [1] deficient
3 [1] deficient
4 [1, 2] deficient
5 [1] deficient
6 [1, 2, 3] perfect
7 [1] deficient
8 [1, 2, 4] deficient
9 [1, 3] deficient
10 [1, 2, 5] deficient
```

    In the sample code `sums.py`, the `random_list()` function demonstrates the use of append to create a list. Use a list to store the divisors of the number.

    You can check your results against the table found here `http://en.wikipedia.org/wiki/Table_of_divisors#1_to_100`

    Name your program `perfect.py`

12. From Project Euler, Counting Sundays:

    You are given the following information, but you may prefer to do some research for yourself.

    1 Jan 1900 was a Monday.
    Thirty days has September,
    April, June and November.
    All the rest have thirty-one,
    Saving February alone,

Which has twenty-eight, rain or shine.
And on leap years, twenty-nine.
A leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400.

How many Sundays fell on the first of the month during the twentieth century (1 Jan 1901 to 31 Dec 2000)?

Write a program to find the answer.

Name your source code `sundays.py`

Submission

Create a tarball of your *.py files.

```
tar czvf cse107_firstname_lastname_lab3.tar.gz *.py
```

To check the contents of your tarball, run the following command:

```
tar tf cse107_firstname_lastname_lab3.tar.gz *.py
```

You should see a list of your Python source code files.

Upload your tarball in Moodle before the start of you next lab.