6.189 Day 3

Today there are no written exercises. Turn in your code tomorrow, stapled together, with your name and the file name in comments at the top as detailed in the Day 1 exercises.

Readings

*How To Think Like A Computer Scientist*, chapter 3; chapter 6, sections 6.5 - 6.9.

*6.01 Fall 2009 Course Notes* sections 2.1, 2.2, and 2.3 (up to the heading ‘List Comprehensions’). Note that the 6.01 course notes are very fast paced- if you see concepts you are confused about, or we haven’t covered, don’t fret- we will cover them, and you are free to come to office hours to ask questions.

Exercise 3.0 – Print vs Return

**Note:** this exercise is in section 2.2 of the 6.01 Course Notes, if you have already gone through it you may skip it.

This isn’t really an exercise, just an important bit of reading. Make a new file called Day3.py and define these two different functions:

```python
def f1(x):
    print x + 1

def f2(x):
    return x + 1
```

Run your code in the shell. What happens when we call these functions?

```bash
>>> f1(3)
4
>>> f2(3)
4
```

It looks like they behave in exactly the same way. But they don’t, really. Try this:

```bash
>>> print(f1(3))
4
None
>>> print(f2(3))
4
```

In the case of *f1*, the function, when evaluated, prints 4; then it returns the value `None`, which is printed by the Python shell. In the case of *f2*, it doesn’t print anything, but it returns 4, which is printed by the Python shell. Finally, we can see the difference here:
>>> f1(3) + 1
4
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
>>> f2(3) + 1
5

In the first case, the function doesn’t return a value, so there’s nothing to add to 1, and an error is generated. In the second case, the function returns the value 4, which is added to 1, and the result, 5, is printed by the Python read-eval-print loop. The book *How To Think Like A Computer Scientist* was translated from a version for Java, and it has a lot of print statements in it, to illustrate programming concepts. But for just about everything we do, it will be returned values that matter, and printing will be used only for debugging, or to give information to the user.

Print is very useful for debugging. It’s important to know that you can print out as many items as you want in one line:

```python
>>> x = 100
>>> print 'x', x, 'x squared', x*x, 'sqrt(x)', x**0.5
x 100 x squared 10000 sqrt(x) 10.0
```

**Exercise 3.1 – Defining A Function**

Recall how we define a function using `def`, and how we pass in parameters. In Day3.py, transform your code from exercise 2.4 (the rock, paper, scissors game) into a function that takes parameters *(instead of asking the user for input)*. Make sure to return your answer, rather than printing it.

**Exercise 3.2 – Math Module**

In this exercise, we will play with some of the functions provided in the math module. A module is a Python file with a collection of related functions. To use the module, you need to add the following line at the beginning of your program:

```python
import math
```

Now if you want to find out what is $\sin(90^\circ)$, we first need to convert from degrees to radians and then use the sin function in the math module:

```python
radians = (90.0 / 360.0) * 2 * math.pi
print math.sin(radians)
```

Many computations can be expressed concisely using the “multadd” operation, which takes three operands and computes $a\times b+c$. One of the purposes of this exercise is to practice pattern-matching: the ability to recognize a specific problem as an instance of a general category of problems.

For the mathematical functions, you can generally call `math.func`, where `func` is whatever function you want to call. For example, if you want the sine of an angle $a$ (where $a$ is in radians), you can call `math.sin(a)`. For logarithms, the function `math.log(n)` calculates the natural logarithm of $n$. You can calculate a log of any base using `math.log(n, b)`, where $b$ is the base of the logarithm. The math module even includes constants such as $e$ (math.e) and $\pi$ (math.pi). Documentation for the math module is available at [http://www.python.org/doc/2.5.1/lib/module-math.html](http://www.python.org/doc/2.5.1/lib/module-math.html).
In the last part, you get a chance to write a method that invokes a method you wrote. Whenever you do that, it is a good idea to test the first method carefully before you start working on the second. Otherwise, you might find yourself debugging two methods at the same time, which can be very difficult.

1. Write a function \texttt{multadd} that takes three parameters, \( a \), \( b \) and \( c \). Test your function well before moving on.

2. Underneath your function definition, compute the following values using \texttt{multadd} and print out the result:

   \[
   \sin \left( \frac{\pi}{4} \right) + \frac{\cos \left( \frac{\pi}{2} \right)}{2}
   \]

   \[
   \left\lfloor \frac{276}{19} \right\rfloor + 2 \log_7(12)
   \]

   \textbf{Hint:} If you are unfamiliar with the notation \( \left\lfloor \cdot \right\rfloor \), this represents the ceiling of a number. The ceiling of some float \( x \) means that we always “round up” \( x \). For example, \( \left\lfloor 2.1 \right\rfloor = \left\lfloor 2.9 \right\rfloor = 3.0 \). Look at the math module documentation for a way to do this!

   If everything is working correctly, your output should look like:

   \[
   \sin(\pi/4) + \cos(\pi/4)/2 \text{ is: 1.06066017178}
   \]

   \[
   \text{ceiling}(276/19) + 2 \log_7(12) \text{ is: 17.5539788165}
   \]

3. Write a new function called \texttt{yikes} that has one argument and uses the \texttt{multadd} function to calculate the following:

   \[
   xe^{-x} + \sqrt{1 - e^{-x}}
   \]

   There are two different ways to raise \( e \) to a power- check out the math module documentation. Be sure to return the result! Try \( x=5 \) as a test; your answer should look like:

   \texttt{yikes(5) is 1.0303150673.}

**Exercise 3.3 — Writing Simple Methods**

In this problem you’ll be asked to write three simple methods (method is an interchangeable term for ‘function’). Be sure to test your functions well!

1. Write a method \texttt{is_divisible} that takes two integers, \( m \) and \( n \). The method returns \texttt{True} if \( m \) is divisible by \( n \), and returns \texttt{False} otherwise.

2. Write a method \texttt{rand_divis_3} that takes no parameters and returns \texttt{True} if a randomly generated number is divisible by 3, and \texttt{False} otherwise. For this method we’ll use a new module, the \texttt{random} module. At the top of your code, underneath \texttt{import math}, add the line \texttt{import random}. We’ll use this module to generate a random integer using the function \texttt{randint}, which works as follows:

   \texttt{random.randint(lo, hi)}

   where \texttt{lo} and \texttt{hi} are integers that tell the code the range in which to generate a random integer. 0 to 100 is probably a decent range.

3. Imagine that Python doesn’t have the \texttt{!=} operator built in. Write a method \texttt{not_equal} that takes two parameters and gives the same result as the \texttt{!=} operator. Obviously, you cannot use \texttt{!=} within your function! Test if your code works by thinking of examples and making sure the output is the same for your new method as \texttt{!=} gives you.
Exercise 3.4 – Quadratic Formula

Write a function `roots` that computes the roots of a quadratic equation. Check for complex roots and either print an error message saying that the roots are complex, or just compute and print out the complex roots if you wish.

**Hint 1:** Your function should take three parameters- what are they?

**Hint 2:** We know the roots are complex when what condition about the discriminant is met?

Exercise 3.5 – Café menu

Write a function that first displays a simple café menu (see example below), asks the user to enter the number of a choice, and either prints the appropriate action OR prints an error message that their choice was not valid.

Example output:

```python
>>> menu()
1. Soup and salad
2. Pasta with meat sauce
3. Chef’s special
Which number would you like to order? 2
One Pasta with meat sauce coming right up!
```

Another example output:

```python
>>> menu()
1. Soup and salad
2. Pasta with meat sauce
3. Chef’s special
Which number would you like to order? 5
Sorry, that is not a valid choice.
```

Exercise 3.6 – The game of Nims/Stones

In this game, two players sit in front of a pile of 100 stones. They take turns, each removing between 1 and 5 stones (assuming there are at least 5 stones left in the pile). The person who removes the last stone(s) wins.

Make a new file called nims.py. Write a function to play this game (should it take any parameters?). This may seem tricky, so break it down into parts. Like many programs, we have to use nested loops (one loop inside another). In the outermost loop, we want to keep playing until we are out of stones. Inside that, we want to keep alternating players. You have the option of either writing two blocks of code, or keeping a variable that tracks the current player. The second way is slightly trickier since we haven’t learned lists yet, but it’s definitely do-able! Feel free to ask the LAs for help if you want to look ahead and challenge yourself with this approach.

Finally, we might want to have an innermost loop that checks if the user’s input is valid. Is it a number? Is it a valid number (e.g. between 1 and 5)? Are there enough stones in the pile to take off this many? If any of these answers are no, we should tell the user and re-ask them the question.
If you choose to write two blocks of code, the basic outline of the program should be something like this:

```python
def play_nims(pile, max):
    '''
    An interactive two-person game; also known as Stones.
    @param pile: the number of stones in the pile to start
    @param max: the maximum number of stones you can take on one turn
    '''

    while [pile is not empty]:
        while [player 1’s answer is not valid]:
            [ask player 1]
            [execute player 1’s move]

        [same as above for player 2]
```

Note the lines of code underneath the definition of `play_nims`. This is called a docstring, and is handy to use to tell users of your program what parameters to pass in, and what your program does.

Be careful with the validity checks. Specifically, we want to keep asking player 1 for their choice as long as their answer is not valid, BUT we want to make sure we ask them at least ONCE. So, for example, we will want to keep a variable that tracks whether their answer is valid, and set it to False initially.

When you’re finished, test each other’s programs by playing them!