Homework 3: Higher-order functions

This homework will be done using the hw3.py file available on the class website. The file contains a series of unwritten function definitions. Below are descriptions of what each of the functions should do. You should complete the function definitions so they do the expected thing. Remember to change the `return` statement in the function.

You should test your functions! This might be more complicated in this assignment, since the returned value is often a function. You should think carefully about good ways to test your functions.

It is always ok to use functions defined in one exercise when writing the functions of later exercises. It is also ok to write your own functions to use in writing these functions. That includes copying over functions from previous homeworks.

When you are done defining these functions, upload the modified hw3.py file in a homework3 folder inside your submission folder. Also print the file and bring it to class.

Exercise 1: `max_of_two`

This function takes three inputs. The first two are functions and the third is a number. The `max_of_two` then calls each of the two functions on the number that was given. The output is then the maximum of those two outputs.

Exercise 2: `max_of_funcs`

This function takes as input two functions, f and g. The output should be a new function h, with the property that for any input x the output h(x) is equal to the maximum of f(x) and g(x). Note that this function is very different from the function in exercise 1, but the two are very closely related.

Exercise 3: `is_even`

This function is a test for whether its input is even. It takes an integer and returns `True` or `False`, depending on whether the input is even. The trick here is that you cannot use a `def` statement to create this function. You must do it with a `lambda` statement.

Exercise 4: `conditional_print`

The `print` function takes any input and always prints it. A "conditionally printing function" is a function that prints only inputs of a particular type (while doing nothing on other inputs). For example, you could create a function that prints its input if its input is an integer and a multiple of 3, and does nothing otherwise. What you will make here is a function that constructs such conditionally printing functions. It should take a single input, which we will call a "test". A test is a function like `is_even` that outputs `True` or `False`. It should return a conditionally printing function that only prints inputs for which the test returns `True`.

Exercise 5: `print_evens`

Use what you have done above to create a function called `print_evens` that will print even integers and ignore all other integer input. (It is ok if the function causes an error when given other input.) Do not build `print_evens` directly. You cannot use a `def` statement or a `lambda` statement to create it.

Exercise 6: `environment diagram`

This exercise is not a programming exercise. Below is a piece of pretty scary code. Your goal is to figure out what values `a` and `b` will have at the end of the execution of this piece of code. In order to do this, you should draw a very careful, detailed environment diagram. Make sure you show all frames that are created during the execution of the code, and all important pieces of information, including all assigned variables, as well as which frames are the parents of which other frames. The exact style you use to draw it (the way I do in class or the way the book does it, for example) does not matter, as long as all that information is clear. You are free to actually run this code and check that your answer is correct, but it is the environment diagram I really care about. Draw it large enough that it's easily readable, and staple it to the printout of hw3.py to turn in in class.

```
g = lambda x: x + 3
def foo(f):
    def bar(g):
        return f(g)
    return bar
f = foo(g)
a = f(2)
g = lambda x: x * x
b = f(3)
```