Higher Order Functions and Environment Diagrams

Link to this document: http://tinyurl.com/legf7p4

Instructions: Form a group of 4-5. Start on problem 0. Check off with a lab assistant when you think everyone in your group understands how to solve problem 0. Repeat for problem 1, 2, ...
You're not allowed to move on from a problem until you check off with a lab assistant.
You are allowed to use any and all resources at your disposal, including the interpreter, lecture notes and slides, discussion notes, labs, and the lab assistants. The purpose of this section is to have all the students working together to learn the material and get better at Computer Science together.

0a. What do lambda expressions do?

0b. Explain the difference between the following:

```
>>> def square(x):
...  return x * x


>>> square(4)


>>> square


>>> square = lambda x: x * x
```

0c. Determine if each of the following will error:

```
>>> 1/0


>>> def boom():
      return 1/0


>>> boom()
```

1a. Rewrite the following expressions using lambdas instead of their named counterparts:

i. `square(4)`

ii. `sum_of_squares(3, 4)`

iii.

```
def hello_world():
  return "hello world!"
hello_world()
```

1b. What will Python output?

i. `(lambda x: x(x))(lambda y: 4)`

Hint 1: What is the operator? What are the operands?
Hint 2: Explain what each lambda expression does in English.

ii. `(lambda x, y: y(x))(mul, lambda a: a(3, 5))`

2a. Draw an environment diagram for the following

```
>>> def make_adder(n):
...   return lambda x: x + n
...
>>> add_4 = make_adder(4)
>>> add_4(3)
```

Hint: http://www.ocf.berkeley.edu/~shidi/cs61a/guerrilla/env.txt

2b. Draw an environment diagram for the following

```
>>> def curry(f, x):
...   return lambda y: f(x, y)
...
>>> def square(x):
...  return x * x
…

>>> sum_of_squares = lambda x, y: square(x) + square(y)
>>> warped_square = curry(sum_of_squares, 3)
>>> warped_square(4)

# Tricky; Use a new environment diagram (otherwise it'd be cluttered)
>>> (lambda: curry(sum_of_squares, 3)(4))()

# OPTIONAL; Start with a clean sheet of paper for this
>>> curry(sum_of_squares, 3)((lambda x: 4)(lambda: 1/0))
```

3. Write a `make_skipper`, which takes in a number `n` and outputs a function. When this function takes in a number `x`, it prints out all the numbers between 0 and `x`, skipping every `n`th number.

```
>>>a = make_skipper(2)
>>>a(5)
1
3
5
def make_skipper(n):
```

4. Write `compose`, a function that takes in two functions as arguments and returns a function that takes in one argument and applies the second function and then the first function to that argument:

```
>>> a = compose(lambda x: x * x, lambda x: x + 4)
>>> a(2)  # (2 + 4) * (2 + 4)
36
def compose(f, g):
```

5. Write make_alternator.

```
>>> a = make_alternator(lambda x: x * x, lambda x: x + 4)
>>> a(5)
1
6
9
8
25
```

6. Here is a version of cons:

```
def cons(a, b):
  return lambda m: m(a, b)
```

Write car and cdr:

```
def car(f):




def cdr(f):
```

(Optional) draw an environment diagram for the following:

```
z = cons('hello', 'world')
cdr(z)
```

Challenge Problem

7. The logician Alonzo Church invented a system of representing non-negative integers entirely using functions. Here are the definitions of 0, and a function that returns 1 more than its argument:

```
def zero(f):
    return lambda x: x

def successor(n):
    return lambda f: lambda x: f(n(f)(x))
```

This representation is known as Church numerals. Define one and two directly, which are also functions. To get started, apply successor to zero. Then, give a direct definition of the add function (not in terms of repeated application of successor) over Church numerals. Finally, implement a function church_to_int that converts a church numeral argument to a regular Python int:

```
def one(f):
    """Church numeral 1."""


def two(f):
    """Church numeral 2."""


def church_to_int(n):
    """Convert the Church numeral n to a Python integer.

    >>> church_to_int(zero)
    0
    >>> church_to_int(one)
    1
    >>> church_to_int(two)
    2
    """
```

```
def add_church(m, n):
    """Return the Church numeral for m + n, for Church numerals m and n.

    >>> three = successor(two)
    >>> church_to_int(add_church(two, three))
    5
    """


def mul_church(m, n):
    """Return the Church numeral for m * n, for Church numerals m and n.

    >>> three = successor(two)
    >>> four = successor(three)
    >>> church_to_int(mul_church(two, three))
    6
    >>> church_to_int(mul_church(three, four))
    12
    """
```

7. Please fill out this survey on your experience at this Guerrilla Section:
http://tinyurl.com/ltu4yhc