**CIS 61**

**Practice 03 - Higher Order Functions**

## Part 1 - What Would Python Display?

### Type Function if you believe the answer is <function...>, Error if it errors, and Nothing if nothing is displayed.

### As a reminder, the following two lines of code will not display anything in the Python interpreter when executed:

### >>> x = None

### >>> x

### Practice 1:

| >>> **lambda** x: x # A lambda expression with one parameter x  \_\_\_\_\_\_  >>> a = **lambda** x: x # Assigning the lambda function to the name a  >>> a(5)  \_\_\_\_\_\_  >>> (**lambda**: 3)() # Using a lambda expression as an operator in a call exp.  \_\_\_\_\_\_  >>> b = **lambda** x: **lambda**: x # Lambdas can return other lambdas!  >>> c = b(88)  >>> c  >>> z = 3  >>> e = lambda x: lambda y: lambda: x + y + z  >>> e(0)(1)()  \_\_\_\_\_\_  >>> f = lambda z: x + z  >>> f(3)  \_\_\_\_\_\_ |
| --- |

| >>> higher\_order\_lambda = lambda f: lambda x: f(x)  >>> g = lambda x: x \* x  >>> higher\_order\_lambda(2)(g) # Which argument belongs to which function call?  \_\_\_\_\_\_  >>> higher\_order\_lambda(g)(2)  \_\_\_\_\_\_  >>> call\_thrice = lambda f: lambda x: f(f(f(x)))  >>> call\_thrice(lambda y: y + 1)(0)  \_\_\_\_\_\_  >>> print\_lambda = lambda z: print(z) # When is the return expression of a lambda expression executed?  >>> print\_lambda  \_\_\_\_\_\_  >>> one\_thousand = print\_lambda(1000)  \_\_\_\_\_\_  >>> one\_thousand  \_\_\_\_\_\_ |
| --- |

### Practice 2: Type Function if you believe the answer is <function...>, Error if it errors, and Nothing if nothing is displayed.

| >>> **def** **even**(f):  ... **def** **odd**(x):  ... **if** x < 0:  ... **return** f(-x)  ... **return** f(x)  ... **return** odd  >>> steven = **lambda** x: x  >>> stewart = even(steven)  >>> stewart  \_\_\_\_\_\_  >>> stewart(61)  \_\_\_\_\_\_  >>> stewart(-4)  \_\_\_\_\_\_ | >>> **def** **cake**():  ... print('beets')  ... **def** **pie**():  ... print('sweets')  ... **return** 'cake'  ... **return** pie  >>> chocolate = cake()  \_\_\_\_\_\_  >>> chocolate  \_\_\_\_\_\_  >>> chocolate()  \_\_\_\_\_\_  >>> more\_chocolate, more\_cake = chocolate(), cake  \_\_\_\_\_\_  >>> more\_chocolate  \_\_\_\_\_\_ |
| --- | --- |

| >>> **def** **snake**(x, y):  ... **if** cake == more\_cake:  ... **return** **lambda**: x + y  ... **else**:  ... **return** x + y  >>> snake(10, 20)  \_\_\_\_\_\_  >>> snake(10, 20)()  \_\_\_\_\_\_  >>> cake = 'cake'  >>> snake(10, 20)  \_\_\_\_\_\_ |
| --- |

Part 2 - Environment Diagrams

I encourage you to do these problems on paper to develop familiarity with Environment Diagrams, which **will appear on the exam.**

You can check your work with the [Online Python Tutor](https://web.archive.org/web/20181003091417/http://tutor.cs61a.org/), but try drawing it yourself first

**Practice** 3 - Draw the environment diagram for the following code:

| n = 9  def make\_adder(n):  return lambda k: k + n  add\_ten = make\_adder(n+1)  result = add\_ten(n) |
| --- |

**Practice** 5 - Draw the environment diagram that results from executing the code below.

| def curry2(h):  def f(x):  def g(y):  return h(x, y)  return g  return f  make\_adder = curry2(lambda x, y: x + y)  add\_three = make\_adder(3)  add\_four = make\_adder(4)  five = add\_three(2) |
| --- |

**Practice 6** - Draw the environment diagram that results from executing the code below.

| n = 7  def f(x):  n = 8  return x + 1  def g(x):  n = 9  def h():  return x + 1  return h  def f(f, x):  return f(x + n)  f = f(g, n)  g = (lambda y: y())(f) |
| --- |