**CONTROLS**

>>> x = 4

>>> x > 2 and x < 6

\_\_\_\_\_\_

>>> True and not True # a.k.a. a contradiciton

\_\_\_\_\_\_

>>> True and True # a.k.a. a tautology

\_\_\_\_\_\_

>>> False and True or True

\_\_\_\_\_\_

>>> False and (True or True)

\_\_\_\_\_\_

>>> False or True or 1 / 0

\_\_\_\_\_\_

>>> False and 1 / 0

\_\_\_\_\_\_

>>> 3 and 4

\_\_\_\_\_\_

>>> 3 or 4

\_\_\_\_\_\_

>>> if True:

... print('True!')

... else:

... print('False!')

\_\_\_\_\_\_

>>> if 4:

... print('True!')

... else:

... print('False!')

\_\_\_\_\_\_

>>> if 0:

... print('True!')

... else:

... print('False!')

\_\_\_\_\_\_

>>> x = 42

>>> if x < 0:

... print('negative')

... elif x == 42:

... print('The answer to everthing')

... else:

... print('Boring number')

\_\_\_\_\_\_

**Environment Diagrams**

def square(x):

return x \* x

def sum\_of\_squares(x, y):

return square(x) + square(y)

result = sum\_of\_squares(3, 4)

# How many times do we call mul?

# How many frames do we draw for mul?

**-------------------------------------------------------------------------------------------------------**

from operator import add

first = add(3, 4)

def add(a, b):

return a + b

second = add(3, 4)

# What changes between the first time we call add and the

# second time? How does this affect our diagram?

score, opp\_score = 0, 0

def assign(arg0, arg1):

score = arg0

opp\_score = arg1

return True

success = assign(3, 9001)

def branch(x):

if x > 10:

x -= 5

elif x > 7:

x -= 2

if x % 2 == 0:

return 'even'

else:

return 'odd'

a = branch(12)

b = branch(8)

**-------------------------------------------------------------------------------------------------------**

def uhoh(x):

if x:

y = 5

return y

a = uhoh(True)

b = uhoh(False)

**---------------------------------------------------------------------------------------------------------**

def is\_even(x):

return x % 2 == 0

i = 0

while i < 2:

if is\_even(i):

print(i)

i += 1

**True AND True**

**True AND False**

**False AND True**

**False AND False**

**1/0 AND True**

**False AND 1/0**

**True AND 1/0**

**True OR True**

**True OR False**

**False OR True**

**False OR False**

**1/0 OR True**

**False OR 1/0**

**True OR 1/0**

**WHILE/FOR LOOPS**

>>> x = 0

>>> while x < 5:

... x += 1

... print(x)

\_\_\_\_\_\_

>>> while False:

... print('hi!')

\_\_\_\_\_\_

>>> while True:

... print('hi!')

... # press Control C to get out of this

\_\_\_\_\_\_

>>> def foo(n):

... while n > 0:

... if n \* n == 8 \* n - 16:

... return True

... n -= 1

... return False

>>> foo(3)

\_\_\_\_\_\_

>>> foo(9)

\_\_\_\_\_\_

>>> n = 0

>>> for x in range(5):

… n = n+x

>>> print (n)

\_\_\_\_\_

>>> for k in range(10):

… if k%2 == 0:

… print (k+1)

**Lambda**

>>> lambda x: x \* x

\_\_\_\_\_\_

>>> g = lambda x: x\*\*2

>>> g(4)

\_\_\_\_\_\_

>>> (lambda x, y: x \* y)(4, 5)

\_\_\_\_\_\_

Translate the following into lambda expressions:

# 1

def square(x):

return x \* x

# 2

def compose(f, g):

def h(x):

return f(g(x))

return h

Translate the following into def statements

# 1

pow = lambda x, y: x\*\*y

# 2

foo = lambda x: lambda y: lambda z: x + y \* z

foo(2)

Fill in the blank so result == 42

x = lambda x, y: lambda: x - y

result = (lambda \_\_\_\_, question: one(\_\_\_\_\_\_\_\_\_\_)(x, 4)