Name: _______Sample Solution_________________

Email address (UW NetID): _____________________________________

CSE 160 Spring 2018: Midterm Exam

(closed book, closed notes, no calculators)

Instructions: This exam is closed book, closed notes. You have 50 minutes to complete it. It contains 10 questions and 9 pages (including this one), totaling 90 points. Before you start, please check your copy to make sure it is complete. Turn in all 9 pages of the exam, together, when you are finished. When time has been called you must put down your pencil and stop writing. A syntax sheet will be provided separately. **Points will be deducted from your score if you are writing after time has been called.** You should only use parts of Python that have been covered in the class so far.

Good Luck! Total: 90 points. Time: 50 minutes.

Problem	**Points Possible**
1	**6**
2	**4**
3	**6**
4	**6**
5	**12**
6	**8**
7	**8**
8	**12**
9	**14**
10	**14**
Total	**90**

1) [6 pts] For each of the if statements below, write the output when x = 20, x = 40, and x = 100 in the table below. If there is no output then write “NO OUTPUT”.

```
a)
if x < 30:
    print "line 1"
if x <= 40:
    print "line 2"
elif x < 100:
    print "line 3"
```

```
b)
if x > 20:
    print "line 1"
else:
    if x < 50:
        print "line 2"
    print "line 3"
```

	x = 20	x = 40	x = 100
Code a)	**line 1** **line 2**	**line 2**	**NO OUTPUT**
Code b)	**line 2** **line 3**	**line 1**	**line 1**

2) [4 pts] Write the output of the code below in the box here:

```
sum = 0
for x in range(6, 0, -2):
    for y in range(x):
        sum = sum + y
print 'sum:', sum
```

MY ANSWER:

```
sum: 22
```

3) [6 pts] What output is produced after running the following piece of code?

```
A = [1, 3, 7]
B = A
C = A[:]

A.append(C[-1])
B[2] = 5
C[1:2] = [9, 10, 11]

print A
print B
print C
```

MY ANSWER:

```
[1, 3, 5, 7]

[1, 3, 5, 7]

[1, 9, 10, 11, 7]
```

4) [6 pts] What output is produced after running the following piece of code?

```
from operator import itemgetter

data = [ ("soda", 4, 8), ("tea", 3, 2), ("juice", 5, 7),
         ("water", 5, 0), ("coffee", 7, 2) ]

def some_key(x):
    return x[1] + x[2]

print sorted(data, key=some_key)
print sorted(data, key=itemgetter(2, 0))
```

MY ANSWER:

```
[('tea', 3, 2), ('water', 5, 0), ('coffee', 7, 2), ('soda', 4, 8), ('juice', 5, 7)]


[('water', 5, 0), ('coffee', 7, 2), ('tea', 3, 2), ('juice', 5, 7), ('soda', 4, 8)]
```

5) [12 pts] For each of the following statements, show what is printed. If nothing is printed then write “NO OUTPUT”.

```
x = -200
def fig(x):
    if x < 100:
        return "small"
    else:
        return "large"

def pear(x):
    print "pear:", x
    return x + 7

def apple(y):
    y = pear(y)
    print "apple:", y
```

```
  a) print pear(10)
```

```
     pear: 10
     17
```

```
  b) print apple(2)
```

```
     pear: 2
     apple: 9
     None
```

```
  c) print fig(120)
```

```
     large
```

```
  d) print fig(pear(2))
```

```
     pear: 2
     small
```

6) [8 pts] Given the following dictionary, write what each expression evaluates to. If an error is thrown, write "**Error**".

```
my_dict = {5:"red", 2:"orange", 0:"green", 1:"purple"}
```

a) `my_dict[2]`

`"orange"`

b) `my_dict[3]`

`KeyError: 3`

c) `my_dict[1][1]`

`"u"`

d) `my_dict["red"]`

`KeyError: 'red'`

It was o.k. to just say 'Error' for this question for part b) and d).

7) [8 pts] What is the output of the following code? If the code has an error write "**Error**".

```
a = {1, 4, 5, 6, 9}
b = {4, 6, 8, 9}
```

a) `print b & a`

`set([9, 4, 6])`

b) `print a | b`

`set([1, 4, 5, 6, 8, 9])`

c) `print b - a`

`set([8])`

d) `print a.remove(8)`

`KeyError: 8`

Note: This notation was also fine: `{ 9, 4, 6 }` and since sets are not ordered, any ordering of elements within the set was fine.

It was also o.k. to just say 'Error' for this question for part d).

8) [12 pts] a) **Draw** the entire environment, including all active environment frames and all user-defined values, **at the moment that the MINUS OPERATION IS performed**. Feel free to draw out the entire environment, but be sure to CLEARLY indicate what will exist at the moment the **MINUS** operation is performed (e.g. cross out frames that no longer exist).

b) When finished executing, **what is printed out by this code**?

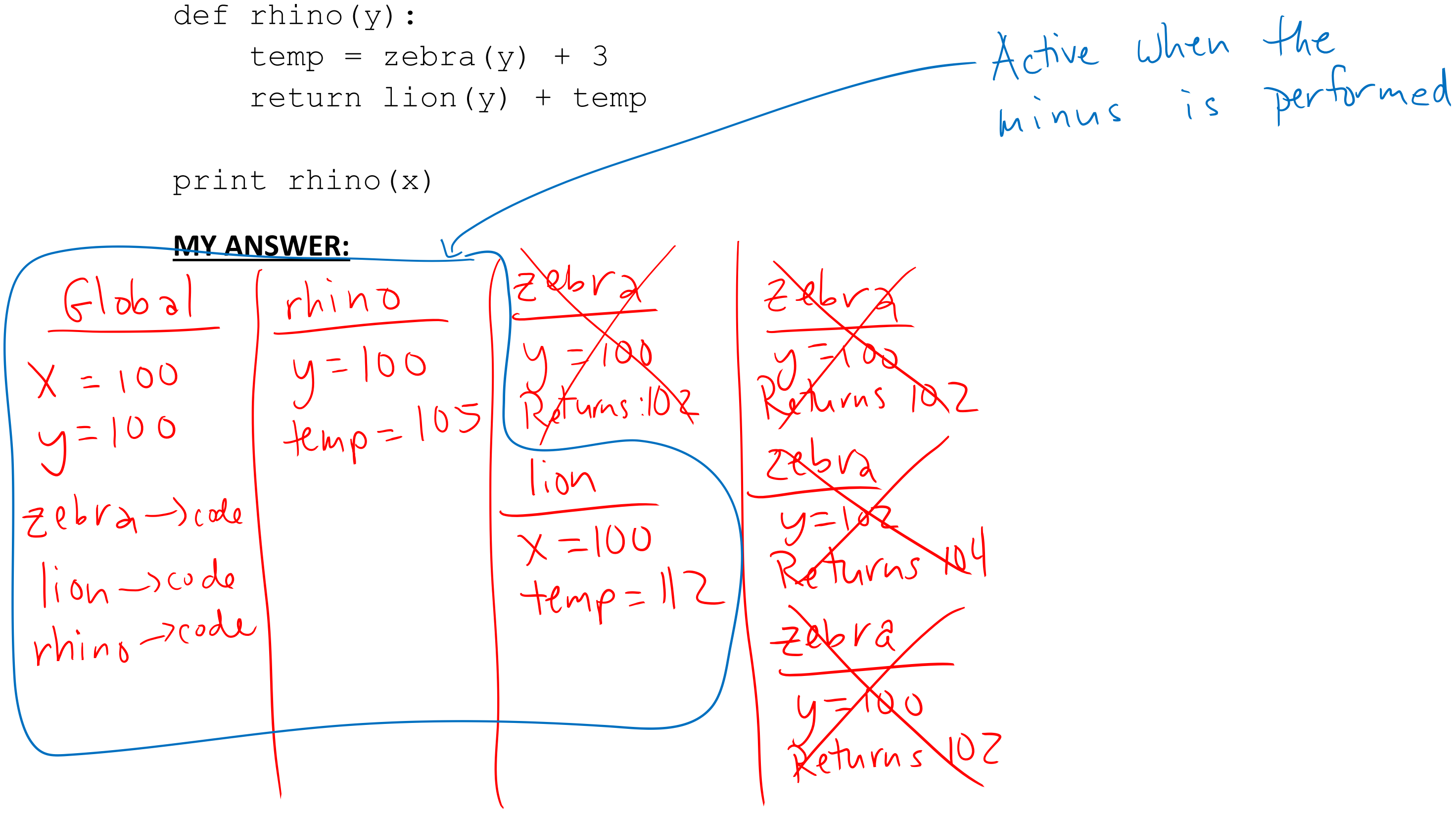
MY ANSWER: 95

c) **How many different stack frames** (environment frames) are active when the call stack is DEEPEST/LARGEST? (Hint: The global frame counts as one frame.)

MY ANSWER: 4

```
x = 100
y = 200

def zebra(y):
    return y + 2

def lion(x):
    temp = zebra(zebra(x)) + 8
    return zebra(x) - temp

def rhino(y):
    temp = zebra(y) + 3
    return lion(y) + temp

print rhino(x)
```

MY ANSWER:



9) [14 pts] Write a function `get_youngest_person` that takes a list of dictionaries as arguments and returns the name of the youngest person in the list. The list of dictionaries will have the following format:

```
people= [
{"name": "Alice", "age": 20},
{"name": "Bob", "age": 9},
{"name": "Dan", "age": 56}
]
```

For example, `get_youngest_person(people)` should return “Bob”. If there is more than one person with the smallest age, return the name of the person who occurs first in the list. You may assume the list contains at least one person and that no one is less than 1 year old.

```
def get_youngest_person(people):
  # Write your code here
```

```
# Two possible solutions:
def get_youngest_person(people):
   yp_index = 0
   yp_age = people[0]["age"]
   for i in range(len(people)):
       if people[i]["age"] < yp_age:
           yp_age = people[i]["age"]
           yp_index = i
   return people[yp_index]["name"]


------------------------

def get_youngest_person(people):
   name_dict = people[0]
   yp_name = name_dict["name"]
   yp_age = name_dict["age"]
   for name_dict in people:
       if name_dict["age"] < yp_age:
           yp_age = name_dict["age"]
           yp_name = name_dict["name"]
   return yp_name
```

10) [14 points] Write a function called `transpose` that takes a `pixel_grid` as described in Homework 3 as an argument and returns the transpose of that `pixel_grid`. This is identical to how we would transpose a matrix: swap the rows and columns. For example, if we had:

```
grid1 = [ [1, 2, 3],
          [4, 5, 6] ]       # a grid with 2 rows and 3 columns


grid2 = [ [1, 2, 3, 4] ]    # a grid with 1 row  and 4 columns


grid3 = [ [1] ]             # a grid with 1 row  and 1 column
```

The call: `transpose(grid1)` would return:

```
[ [1, 4],
  [2, 5],
  [3, 6] ]
```

A grid with 3 rows and 2 columns

The call: `transpose(grid2)` would return:

```
[ [1],
  [2],
  [3],
  [4] ]
```

A grid with 4 rows and 1 column

The call: `transpose(grid3)` would return: `[ [1] ]`

A grid with 1 row and 1 column

You may assume that the provided `pixel_grid` contains at least one row and one column.

Write your code on the next page:

10) (continued)

```
def transpose(pixel_grid):
    # Write your code here

# Two common solutions:

# append and append
def transpose(pixel_grid):
    transposed_grid = []
    num_orig_row = len(pixel_grid)
    num_orig_col = len(pixel_grid[0])
    for i in range(num_orig_col):
        transposed_row = []
        for j in range(num_orig_row):
            transposed_row.append(pixel_grid[j][i])
        transposed_grid.append(transposed_row)
    return transposed_grid

    ------------------------


# create and fill
def transpose(pixel_grid):
    transposed_grid = []
    num_orig_row = len(pixel_grid)
    num_orig_col = len(pixel_grid[0])
    for i in range(num_orig_col):
        transposed_grid.append([])
        for j in range(num_orig_row):
            transposed_grid[i].append(0)
    for i in range(num_orig_row):
        for j in range(num_orig_col):
            transposed_grid[j][i] = pixel_grid[i][j]
    return transposed_grid
```