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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
</tr>
</tbody>
</table>
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"

<table>
<thead>
<tr>
<th></th>
<th>x = 20</th>
<th>x = 40</th>
<th>x = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code a)</td>
<td>line 1</td>
<td>line 2</td>
<td>NO OUTPUT</td>
</tr>
<tr>
<td></td>
<td>line 2</td>
<td>line 2</td>
<td></td>
</tr>
<tr>
<td>Code b)</td>
<td>line 2</td>
<td>line 1</td>
<td>line 1</td>
</tr>
<tr>
<td></td>
<td>line 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```python
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?

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

A.append(C[-1])
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?

```python
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”.

\[
\begin{align*}
x &= -200 \\
def \text{fig}(x): \\
&\quad \text{if } x < 100: \\
&\quad \quad \text{return } "\text{small}" \\
&\quad \text{else:} \\
&\quad \quad \text{return } "\text{large}"
\end{align*}
\]

\[
\begin{align*}
def \text{pear}(x): \\
&\quad \text{print } "\text{pear:}\), x \\
&\quad \text{return } x + 7
\end{align*}
\]

\[
\begin{align*}
def \text{apple}(y): \\
&\quad y = \text{pear}(y) \\
&\quad \text{print } "\text{apple:}\), y
\end{align*}
\]

\[
\begin{align*}
a) \text{print } \text{pear}(10) \\
&\quad \text{pear: 10} \\
&\quad 17 \\
b) \text{print } \text{apple}(2) \\
&\quad \text{pear: 2} \\
&\quad \text{apple: 9} \\
&\quad \text{None} \\
c) \text{print } \text{fig}(120) \\
&\quad \text{large} \\
d) \text{print } \text{fig}(\text{pear}(2)) \\
&\quad \text{pear: 2} \\
&\quad \text{small}
\end{align*}
\]
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'

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

```python
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
```
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 |

---

```python
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)
```

---

**Active When the minus is performed**

**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:

```python
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.

```python
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:

```python
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