

Name: \_\_\_\_\_

Email address (UW NetID): \_\_\_\_\_

## **CSE 160 Winter 2017: Final Exam**

(closed book, closed notes, no calculators)

**Instructions:** This exam is closed book, closed notes. You have 50 minutes to complete it. It contains 7 questions and 8 pages (including this one), totaling 70 points. Before you start, please check your copy to make sure it is complete. A syntax sheet will be provided separately. When time has been called you must put down your pencil and stop writing. **Points will be deducted from your score if you are writing after time has been called.**

Total: 70 points. Time: 50 minutes.

<b>Problem</b>	<b>Max Points</b>	<b>Score</b>
<b>1</b>	<b>4</b>	
<b>2</b>	<b>3</b>	
<b>3</b>	<b>4</b>	
<b>4</b>	<b>10</b>	
<b>5</b>	<b>23</b>	
<b>6</b>	<b>16</b>	
<b>7</b>	<b>10</b>	
Total	<b>70</b>	

1) [4 pts] Give three advantages of using functions. Your reasons should be as different as possible, and no longer than a sentence each (a phrase each is fine).

2) [3 pts] Write code that would produce an "TypeError" error

3) [4 pts] Write the output of the code below in the box here:

```
sum = 0
for x in range(1, 25, 2):
    temp = (x / 10) % 10
    sum = sum + temp
print 'sum:', sum
```

**MY ANSWER:**

4) [10 pts] Write a function called `union_sets` that takes two lists of sets of integers as arguments. Assume the two lists are of equal length. The function should return a list containing the unions of the two sets in the **corresponding positions** of each list. For example:

```
list_a = [{1}, {3, 4, 5}, {1, 2}]
list_b = [{3}, {3, 5, 6}, {2}]
print union_sets(list_a, list_b)
```

Would print something like this:

```
[set([1, 3]), set([3, 5, 4, 6]), set([1, 2])]
```

**MY ANSWER:**

```
def union_sets(lst_one, lst_two):
    # Assumes lst_one and lst_two contain at least one set each.

    # Your code here
```

5) [23 pts total] a) Write a function called `read_expenses(filename)` that takes the name of a file as a parameter. You can assume that the given file contains lines of the form :  
`store_name amount_spent` where `store_name` is a string (that contains no spaces or punctuation) and `amount_spent` is a float. The two values are separated by a single space. Here are the contents of a sample input file:

```
varsity_theater 10.56  
shultzys 15.34  
solstice_cafe 5.01  
shultzys 4.50  
solstice_cafe 6.0
```

Your function should read in the given file and return a dictionary mapping each `store_name` to the **total** amount of money you have spent there. You may assume the file name provided is valid and that the file is formatted as described and includes at least one `store_name amount_spent` pair. Calling `read_expenses` on the file above returns this dictionary:  
{'varsity\_theater': 10.56, 'shultzys': 19.84, 'solstice\_cafe': 11.01}

```
def read_expenses(filename):  
    # Your code here
```

b) Describe 3 things about your approach to testing this function.

5) (cont.) c) Write code in the main function that will:

- call the `read_expenses` function written in part a) to read a file called `"jan.txt"`.
- print out the expenses sorted from the store where you spent the most to the store where you spent the least (if you spent the same amount at more than one store, sort alphabetically from a to z by store name). You should print the results in **EXACTLY** the following format. For the sample input file shown in part a) the output would be:

```
Spent 19.84 at shultzys
Spent 11.01 at solstice_cafe
Spent 10.56 at varsity_theater
```

```
from operator import itemgetter
def main():
    # Your code here
```

6) [16 pts] You are given the following class definition:

```
class MovieRatings:
    def __init__(self, user_name):
        '''user_name: a string representing the name of the person
        these movie ratings belong to
        '''
        self.name = user_name
        self.scores = {}

    def rate(self, movie_name, rating):
        '''movie_name: a string representing a movie
        rating: a float representing this user's rating of the movie
        '''
        self.scores[movie_name] = rating
```

a) Write the code for the method below that is also a part of the class **MovieRatings**:

```
def get_highest_rating(self):
    '''Returns a float representing the highest rating of all
    the movies this user has rated. Returns 0 if the user
    has not yet rated any movies.
    '''
    # Your code here
```

6) (continued)

Write code in the `main` function, using methods from the `MovieRatings` class, to:

- add a rating of 9.5 for “Moonlight” to `marys_ratings`.
- Print Mary Jones’ highest rating

This code is outside of the class `MovieRatings`.

```
def main():  
    marys_ratings = MovieRatings("Mary Jones")  
    # Your code here:
```

b) Describe a change to the `MovieRatings` class that would NOT cause a **client** of the `MovieRatings` class to have to modify its code.

c) Describe a change to the `MovieRatings` class that might cause a **client** of the `MovieRatings` class to have to modify its code.

7) [10 pts] a) **Draw** the entire environment, including all active environment frames and all user-defined variables, 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:

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:

---

```
def raven(y):  
    x = 5  
    return dog(emu(y) + x)
```

```
def emu(x):  
    return x + 2
```

```
def dog(x):  
    y = emu(emu(x))  
    return emu(x) - y
```

```
y = 100  
x = 4  
print raven(emu(x))
```