CSE 160 practice final exam

Name:

There are 60 points total in the exam.

PART 1: READING CODE

For each of the following pieces of code, write the output. If there is an error, describe the error and the cause, and include the output up until the error.

1.
(2 points)

```
def removeall(somelist, items_to_remove):
  """Return a list with all items from somelist except those that appear in items_to_remove"""
  for item in somelist:
    if item in items_to_remove:
      somelist = somelist.remove(item)
  return somelist

print removeall([1,2,3,4], [2,4])
```

2.
(2 points)

```
def f(i, j=2):
  return j

print f(f(4,3))
```

3.
(5 points)

```
def histogram(words, stopwords=[]):
  """Return a dictionary mapping each word in words to its frequency in words.
  Exclude words that appear in stopwords."""
  d = {}
  for w in words:
    if not w in stopwords:
      c = d.setdefault(w,0)
      d[w] = c + 1
  return d

phrase = "I didn't ask for a dime"
d = histogram(phrase, ["for"])
print d["a"]
print d["dime"]
```

PART 2: WRITING CODE

4.
(10 points)

Fill in the body of the following function. Tests are provided below to help you better understand the function.

```
def similar_pairs(list1, list2, similar):
    """Given two lists, return a list of "similar" pairs of elements.
    Each pair contains one element from list1 and one element from list2.
    The output only contains pairs that are similar according to the given function."""
```

```
### Tests

states = ["Alabama", "Alaska", "Maine", "Texas", "Washington", "West Virginia", "Wisconsin",
"Wyoming"]
capitals = ["Montgomery", "Juneau", "Augusta", "Austin", "Olympia", "Charleston", "Madison",
"Cheyenne"]

def same_first_letter(string1, string2):
    return string1[0] == string2[0]

assert similar_pairs(states, capitals, same_first_letter) == [('Alabama', 'Augusta'),
('Alabama', 'Austin'), ('Alaska', 'Augusta'), ('Alaska', 'Austin'), ('Maine', 'Montgomery'),
('Maine', 'Madison')]

def same_length(string1, string2):
    return len(string1) == len(string2)

assert similar_pairs(states, capitals, same_length) == [('Alabama', 'Augusta'), ('Alabama',
'Olympia'), ('Alabama', 'Madison'), ('Alaska', 'Juneau'), ('Alaska', 'Austin'), ('Washington',
'Montgomery'), ('Washington', 'Charleston'), ('Wyoming', 'Augusta'), ('Wyoming', 'Olympia'),
('Wyoming', 'Madison')]

def same_string(string1, string2):
    return string1 == string2

assert similar_pairs(states, capitals, same_string) == []
```

5.
(8 points)

Using your similar_pairs function, write an expression that yields pairs in which the number of vowels is the same. You may assume that the following list is defined:

```
vowels = ["A", "a", "E", "e", "I", "i", "O", "o", "U", "u"]
```

The result of your expression would be:

```
[('Alabama', 'Juneau'), ('Alabama', 'Augusta'), ('Alaska', 'Montgomery'), ('Alaska', 'Austin'),
('Alaska', 'Olympia'), ('Alaska', 'Charleston'), ('Alaska', 'Madison'), ('Alaska', 'Cheyenne'),
('Maine', 'Montgomery'), ('Maine', 'Austin'), ('Maine', 'Olympia'), ('Maine', 'Charleston'),
('Maine', 'Madison'), ('Maine', 'Cheyenne'), ('Washington', 'Montgomery'), ('Washington',
'Austin'), ('Washington', 'Olympia'), ('Washington', 'Charleston'), ('Washington', 'Madison'),
('Washington', 'Cheyenne'), ('Wisconsin', 'Montgomery'), ('Wisconsin', 'Austin'), ('Wisconsin',
'Olympia'), ('Wisconsin', 'Charleston'), ('Wisconsin', 'Madison'), ('Wisconsin', 'Cheyenne')]
```

You are permitted to write and use a helper function.

6.
(10 points)

a) Write a list comprehension (in a single line of code) to create a list named 'evens' that contains only the even numbers from 0 to 100 inclusive.

b) Given a dictionary named int_to_string_dict create a new dictionary named 'only_B_values' that contains only the mappings from int_to_string_dict where a number maps to a string that starts with the letter "B".

For example, given the dictionary:
int_to_string_dict = { 99: 'Problems', 12: 'Monkeys', 7: 'Deadly Sins', 8: 'Ball', 12: 'Bakers', 9: 'Cloud' }

The result should contain only: { 8: 'Ball', 12: 'Bakers' }

c) Rewrite the following for nested loops as a single line of code. (hint: using list comprehensions)

```
backwards_tens = []
for i in range(10):
    backwards_tens.append([])
    for j in range(9, -1, -1):
        backwards_tens[i].append(i * 10 + j)
```

PART 3: DESIGN

7.
(10 points)

You are given a file of restaurants and bars around Seattle:

name	type	latitude	longitude
Serafina	restaurant	47.638097	-122.326074
Siren Tavern	bar	47.57352	-122.329555
Sitting Room	bar	47.625827	-122.358592
Six Arms	restaurant	47.614123	-122.327743
Targy's Tavern	bar	47.637914	-122.365266
Buca di Beppo	restaurant	47.625575	-122.33963
Chop Suey	restaurant	47.613595	-122.314444
Dubliner	bar	47.651175	-122.350034
Eastlake Zoo Tavern	bar	47.639826	-122.326088
Salty's on Alki	restaurant	47.586484	-122.376451

a) Specify a set of functions for the following requirements. Do not implement the functions. Provide the name, the arguments, and a doc string that describes all inputs and any return value.

Requirement 1: "I want to find all establishments within 10 blocks of my current location." (You may consider 10 blocks to be about 0.007 degrees latitude/longitude.)

Requirement 2: "I want to find a bar that is near a lot of other bars."

b) State one possible strength of your design

c) State one possible weakness of your design.

PART 4: Understanding code

8.
(5 points)

Consider the following definitions:

```
w = 3
x = (3,4,5)
y = [3,4,5]
z = set([3,4,5])
```

a) Assuming the above definitions of w, x, y, and z, for each of the below statements, circle "No error" or "Error", depending on whether execution of it causes an error. Answer assuming that you attempt to execute each statement, even if a previous one caused an error.

```
                 #   For each statement, circle either "No error" or "Error"

d = {}           #   "No error"     "Error"

d[w] = "test"    #   "No error"     "Error"

d[x] = "test"    #   "No error"     "Error"
```

```
d[y] = "test"    #   "No error"     "Error"

d[z] = "test"    #   "No error"     "Error"
```

b) Every case that succeeded has something in common. What is it?

9.
(8 points)

Consider the following code:

```
def gcd(a, b):
    """Compute the greatest common divisor, using Euclid's algorithm."""
    if b == 0:
        return a
    if a < b:
        return gcd(b, a)
    return gcd(a-b, b)
```

Given the following expression:

```
  gcd(15, 10)
```

show the stack frames just before the "return a" statement is first executed.