Name: ______Sample Solution__________________

Email address: ______________________________________

Quiz Section: __________

# CSE 140 Winter 2014: 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 16 questions and 13 pages (including this one), totaling 90 points. Before you start, please check your copy to make sure it is complete. Turn in all pages, together, when you are finished. Please write neatly; we cannot give credit for what we cannot read.

**Good Luck!**

Total: 90 points. Time: 50 minutes.

| **Page** | **Max Points** | **Score** |
|---|---|---|
| **2** | **6** | |
| **3** | **6** | |
| **4** | **7** | |
| **5** | **7** | |
| **6** | **11** | |
| **7** | **8** | |
| **8** | **8** | |
| **9** | **8** | |
| **10** | **14** | |
| **12** | **15** | |
| Total | **90** | |

1) [3 pts] Which of the following expressions will generate an error? (Circle one answer per item)

```
1. "314" + '159'     ERROR            [✓ NO ERROR]

2. 314 + 1.59        ERROR            [✓ NO ERROR]

3. 314 + "159"       [✓ ERROR]        NO ERROR
```

2) [3 pts] Given the code below :

```
if x > 200:
  print "line 1"
elif x > 180:
  print "line 2"
else:
  print "line 3"
if x < 200:
  print "line 4"
else:
  print "line 5"
```

Determine which of the following statements will be true: (Circle all that are true)

A.) Always exactly Zero lines of output are printed

B.) Always exactly One line of output is printed

[✓ **C.) Always exactly Two lines of output are printed**]

D.) Always exactly Three lines of output are printed

E.) The number of lines of output depends on the value of x

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

```
x = 10
def new_value():
     x = 15
     x = x/2
     return x
print x
```

**MY ANSWER:**

**10**

4) [3 pts] What output is produced after running the following piece of code? (Hint: You may find it useful to draw out the environment frames.)

```
i = 15

def z(i):
    q(i + 1)
    print "In z", i

def q(i):
    i = i + 1
    k(i)
    print "In q", i

def k(i):
    print "In k", i

p = 12
z(p)
```

**MY ANSWER:**

```
In k 14
In q 14
In z 12
```

5) [3 pts] What output is produced after running the following piece of code?

```
a = 1
lst = [1, 2, 3]

def foo(a, lst):
    a = 2
    lst.append(4)

b = 3
olst = [5, 6, 7]

foo(b, olst)
print b
print olst
```

**MY ANSWER:**

**3**

**[5, 6, 7, 4]**

6) [4 pts] For each of the following statements, state what is printed.

```
list_1 = [1,2,3]
list_2 = ['a','b','c']
list_3 = [list_1, ["do", "re", "me"], list_2]
```

a) `print list_3[0][1]`

**2**

b) `print list_2[2]`

**c**

c) `print list_3[2][0]`

**a**

d) `print list_3[1]`

**['do', 're', 'me']**

There is no problem #7.

8) [4 pts] What output is produced after running the following piece of code?

```
     def a(n):
         return n.split()[1]

     def b(n):
         return len(n)

    names = ["Isaac Newton", "Neils Bohr", "Albert Einstein"]

    print sorted(names, key = a)
    print sorted(names, key = b, reverse = True)
```

**MY ANSWER:**

```
['Neils Bohr', 'Albert Einstein', 'Isaac Newton']

['Albert Einstein', 'Isaac Newton', 'Neils Bohr']
```

9) [3 pts] You are given the following code:

```
        # Remove the element "Harry" from weasleys
line1    weasleys = { "Harry", "Fred", "George", "Ron", "Ginny" }
line2    not_weasleys = set("Harry")
line3    my_set = weasleys - not_weasleys
```

You print `my_set` and discover that it still contains five elements: "Harry", "Fred", "George", "Ron", and "Ginny". You are allowed to add one print statement to help diagnose this bug.

**WHAT would you print, and WHERE would you print it? (indicate using line numbers shown above)

**MY ANSWER:**

**Print `not_weasleys` after line 2, before line 3.**

10) [4 pts] a) What are **two ways** to include documentation of a function?

1.____**Docstrings (with ‘’’ or “””)** _______________________________________________

2.____**Comments inside of the function body (with #)** _____________________________

b) What is the difference between the purpose of the two ways?

**Docstrings tell the purpose or meaning or abstraction that the function represents and are meant for users/clients callers – tells *what* the function does**

**Comments inside the body of the function document the implementation, specific code choices and are meant for programmers reading/modifying the code – tell *how* the function does it**

11) [3 pts] List **three** significant **differences** between a Python **Set** and a **List**? Give reasons that are as different from one another as possible. Use no more than one sentence each

1.__ **Unlike a list, a set cannot contain duplicates.**

2.__ **Unlike a list, sets are unordered: iteration over a set happens in an arbitrary order.**

3.__**Unlike a list, you cannot index into a set.**

4.__**Unlike a list, you can only put immutable values in sets**

12) [4 pts]:

Give two examples of **mutable** types:

____**dictionary, set, list**

Give two examples of **immutable** types:

_____**string, tuple, float, int, bool**

13) [8 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.

b) 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, That is: `global, foo, double_minus42, double`**
**This occurs inside of the call to `double` that happens on line 5 (see line numbers in picture below).**

---

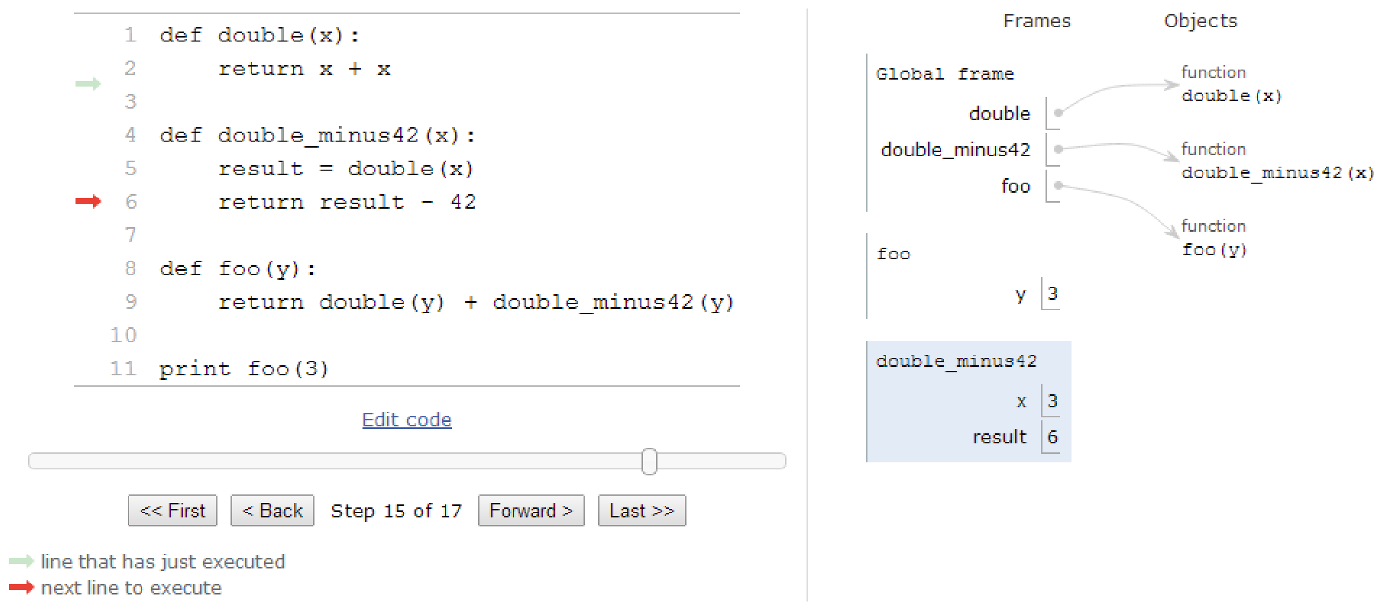
```
def double(x):
    return x + x

def double_minus42(x):
    result = double(x)
    return result - 42

def foo(y):
    return double(y) + double_minus42(y)

print foo(3)
```

**ANSWER:**



```
 1  def double(x):
 2      return x + x
 3
 4  def double_minus42(x):
 5      result = double(x)
 6      return result - 42
 7
 8  def foo(y):
 9      return double(y) + double_minus42(y)
10
11  print foo(3)
```

Step 15 of 17

Frames

```
Global frame
          double  •
  double_minus42  •
             foo  •

foo
               y  3

double_minus42
               x  3
          result  6
```

Objects

```
function
double(x)

function
double_minus42(x)

function
foo(y)
```

14) [8 pts] Given a dictionary that maps names of people (strings) to their ages (integers), write a function that will return a **list** of all the people in the dictionary who are OVER 20 years old. The dictionary passed to the function will be similar to the one below:

```
ages = {'Bob':18, 'Jessica':24, 'Ann':42}
```

The call: `over_twenty(ages)` would return a list containing the names `Jessica` and `Ann`.

```
def over_twenty(age_map):
    ''' Return a list that contains all the people in age_map
    that are over 20 years old. '''
    # Your code starts here

    result = []
    for person in age_map:
        if age_map[person] > 20:   # We accepted >= 20 as well
            result.append(person)
    return result
```

15) [8 pts]: Implement the following function:

```
def reverse_dictionary(to_reverse):
   ''' Return a NEW dictionary that is the reverse of the
       dictionary to_reverse.
       to_reverse will have unique keys and values.
        For example, if to_reverse is:
           { 'a': 1, 'b': 2, 'c': 3 }
        Then this function returns
           { 1: 'a', 2: 'b', 3: 'c' }
    '''
    # Your code starts here

    new_dict = {}
    for key in to_reverse.keys(): # or for key in to_reverse:
        value = to_reverse[key]
        new_dict[value] = key
    return new_dict
```

16) [14 pts] Write a function `build_index_grid(rows, columns)` that, given a number of `rows` and `columns`, creates a list of lists of that shape that includes the `'row,column'` of that location.

For example, after the following code is executed:

```
    new_index_grid = build_index_grid(4,3)
```

`new_index_grid` would contain:

```
 [ ['0,0', '0,1', '0,2'],
   ['1,0', '1,1', '1,2'],
   ['2,0', '2,1', '2,2'],
   ['3,0', '3,1', '3,2'] ]
```

<u>Note that these are **strings**.</u>

After the following code is executed:

```
    small_index_grid = build_index_grid(1,1)
```

`small_index_grid` would contain:

```
  [ ['0,0'] ]
```

**Fill in your code on the next page:**

```
def build_index_matrix(rows, columns):
    """
    Given a number of rows and columns, return a new list
    of lists where each inner list contains a list of the
    indices for that row in the form 'row,column'.
    You may assume that rows and columns will be integers > 0.
    """
    # Your code starts here

    new_grid = []
    for i in range(rows):
        new_row = []
        for j in range(columns):
            index_string = str(i) + ',' + str(j)
            new_row.append(index_string)
        new_grid.append(new_row)
    return new_grid
```

17) [15 pts] Implement the function `friends_of_friends_less(graph, user)` that returns a set of friends of friends of the given `user` **whose ID is less than the `user's` ID**. The result should be a set, that does not include the given `user` nor any of that `user's` friends. You may assume that the parameter `user` and all other nodes in the graph are labeled with positive integer userIDs, as in the facebook data.

For example, if `friends_of_friends(seattle_people, 750)` returned a set containing these values:

```
  { 100, 2005, 34, 8000, 700 }
```

Then `friends_of_friends_less(seattle_people, 750)` should return a set containing these values:

```
    { 100, 34, 700 }
```

Note that this is very similar to the `friends_of_friends` function you implemented in HW4 except that only friends of friends whose userIDs are less than the parameter `user` should be part of the set that is returned. It is possible that the set returned could be empty.

**You CANNOT call the function `friends_of_friends` or any other functions from HW4 other than `friends()`, shown below, in your solution.**

You may call the following function from HW4:

```
def friends(graph, user):
    """Returns a set of the friends of the given user, in the
    given graph. """
    return set(graph.neighbors(user))
```

Hint: As with HW4, using set operations will make this question easiest. It is also fine if your solution uses two “passes”.

**Fill in your code on the next page.**

```
def friends_of_friends_less(graph, user):
    """Returns a set of friends of friends of the given user
    in the given graph whose ID is less than the user's ID.
    Input: user, a positive integer userID
    Output: a set of userIDs of the friends of friends of user
    where each member's ID is < user
    The result does not include the given user nor any of that
    user's friends.
    """
    # Your code starts here

    # Do friends_of_friends, except don't need to remove self
    # (we will do that in the next step)
    fof_set = set()
    user_friends = friends(graph, user)
    for person in user_friends:
        fof_set = fof_set | friends(graph, person)
    fof_set = fof_set - friends(graph, user)

    # Create set of friends less than user
    less_set = set()
    for person in fof_set:
        if person < user:
            less_set.add(person)
            # OR less_set = less_set | { person }
            # OR less_set = less_set | set([person])
    return less_set
```