University of Washington
CSE 140 Introduction to Data Programming
Winter 2013

Midterm exam

February 6, 2013

Name: ***Solutions***____________________________________

UW Net ID (username): ____________________________________

This exam is closed book, closed notes. You have **50 minutes** to complete it. It contains 18 questions and 14 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. **Write your initials on the top of *ALL* pages** (in case a page gets separated during test-taking or grading).

Please write neatly; we cannot give credit for what we cannot read.

Good luck!

Page	Max	Score
2	6	
3	6	
4	9	
5	6	
6	6	
7	10	
8	5	
9	5	
10	3	
11	12	
12	8	
13	6	
14	6	
Total	90	

1 Short answer

1. An immutable data structure is one that cannot change after being created. Give three reasons to use immutable data. Give reasons that are as different from one another as possible. Use no more than one sentence each.

   (a) ***Immutable objects can be put in a set or used as a key to a dictionary.***

   (b) ***Immutable objects protect your data structures from being corrupted by a client of your code or by your own code.***

   (c) ***Immutable objects are easier to reason about, since you don't have to keep track of what their current value is.***

   (d) Use of an immutable object is documentation: it is a signal to other programmers that the object will not change.

   Here are some common incorrect answers:

   - ***Use an immutable object to prevent change. (That is repeating the definition, not giving a reason.)***
   - ***Efficiency. (This could be an acceptable answer with more explanation, but we haven't dwelled on this in CSE 140.)***
   - ***Repeated use of the same value. (This is possible with a mutable structure: just don't change it.)***
   - ***More convenient to use. (In what way, or for what purpose?)***

2 Execute Python expressions

Execute each of the following expressions.

- If it executes without an error, then:

  value state the value that it evaluates to

- If it suffers an error during evaluation:

  error describe the error (in one phrase — a brief explanation in your own words)

  frame state the name of the current environment frame: "global" or a function name

  operator state the operator or operation that caused the error, such as + or function invocation

  arguments state the values to which the operator was being applied

Your answer will contain either part "value", ***or*** parts "error", "frame", "operator", and "arguments".

Hint: none of these is a syntax error.

2. `len(set(1, 2, 3))`

   value

   error ***wrong number of arguments***

   frame ***global***

   operator ***function invocation***

   arguments **set function** ***and*** **1** ***and*** **2** ***and*** **3**

3. `len(set([1, 2, 3]))`

   value **3**

   error

   frame

   operator

   arguments

4. `len(set([(1, 2), (3, 4)]))`

   value **2**

   error

   frame

   operator

   arguments

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

   value

   error ***placing mutable value in a set***

   frame ***set***

   operator ***set constructor***

   arguments $\overset{\textit{list}}{\boxed{1 \mid 2}}$ ***and*** $\overset{\textit{list}}{\boxed{3 \mid 4}}$

6. `len(set(((1, 2), (3, 4))))`

   value **2**

   error

   frame

   operator

   arguments

The following questions assume these definitions:

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

def at42(f):        # The name "at42" is short for "apply_to_42"
  return f(42)
```

7. `at42(double)`

   value **84**

   error

   frame

   operator

   arguments

8. `at42(double(42))`

   value

   error ***invoking a non-function*** ***or*** ***int is not callable.***

   frame ***at42***

   operator ***function invocation***

   arguments **84** ***and*** **42**

   It is* not *correct to say 84 does not exist (it certainly does) nor that* `at42` *expects a function (Python does no checking of argument types at function calls: the argument gets passed in, and the error happens within* `at42`*).

9. `double(at42(42))`

   value

   error ***invoking a non-function*** *or* ***int is not callable***

   frame ***at42***

   operator ***function invocation***

   arguments **42** *and* **42**

10. `double(double)(42)`

    value

    error ***cannot add two functions***

    frame ***double***

    operator ***+***

    arguments **function double** *and* **function double**

3 Draw the environment

11. Draw the entire environment, including all active environment frames and all user-defined variables, at the moment that the division operation is performed.

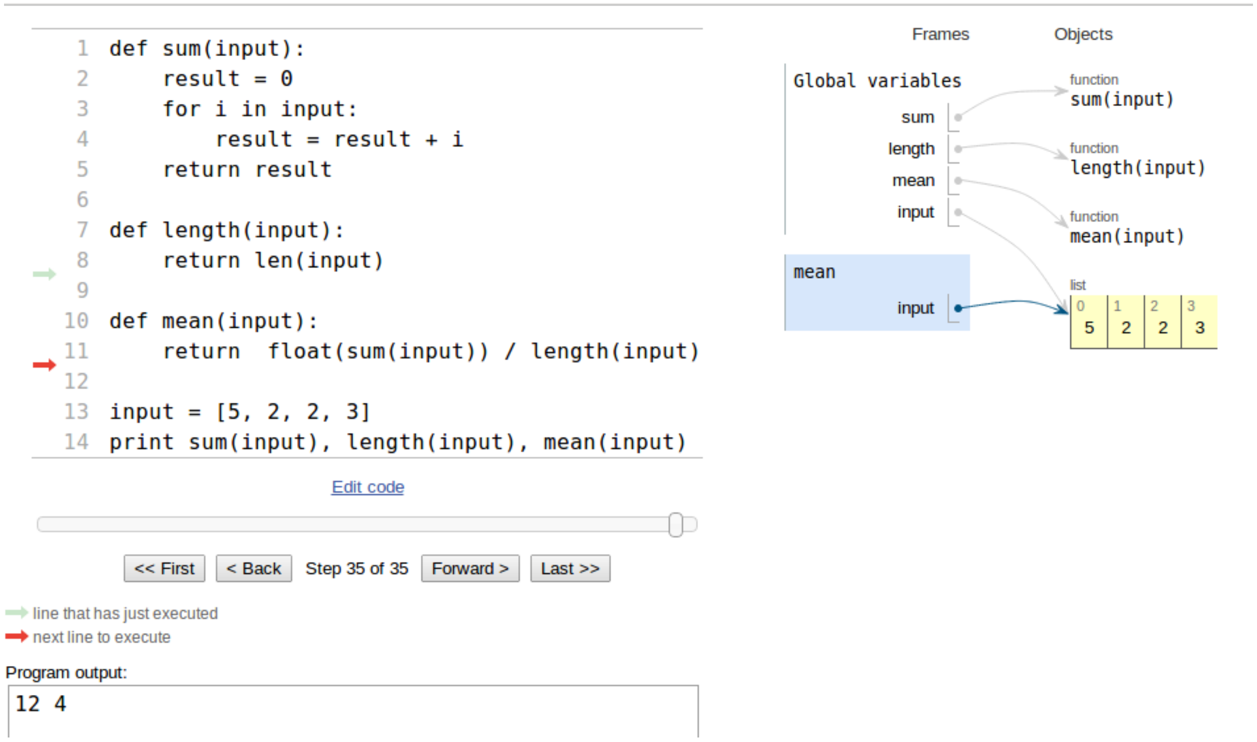
```
def sum(input):
    result = 0
    for i in input:
        result = result + i
    return result

def length(input):
    return len(input)

def mean(input):
    return  float(sum(input)) / length(input)

input = [5, 2, 2, 3]
print sum(input), length(input), mean(input)
```

```
Global variables:
  * sum ----> <function sum(input)>
  * length -> <function length(input)>
  * mean ---> <function mean(input)>
  * input --> [5, 2, 2, 3]
                ^
mean:           |
  * input ----*
```



```
def sum(input):
    result = 0
    for i in input:
        result = result + i
    return result

def length(input):
    return len(input)

def mean(input):
    return  float(sum(input)) / length(input)

input = [5, 2, 2, 3]
print sum(input), length(input), mean(input)
```

Program output:

```
12 4
```

4 Debugging

You have written the following program (in file `printdicts.py`), and you have successfully loaded it into the Python interpreter (say, by pressing F5 within IDLE while viewing `printdicts.py`, or alternately by typing `from printdicts import *`).

```
print_many_dicts(dicts):
    """Print each dictionary in the input, which is a list of dictionaries."""
    for dict in dicts:
        print_dict(dict)

def print_dict(dict):
    """Print the given dictionary."""
    print '{'
    for k in dict:
        print '  ' + str(k) + ': ' + str(dict[k]) + ','
    print '}'
```

Here is an example of using it in the interpreter:

```
>>> print_many_dicts( [ {1:2, 3:4}, {"abra":"cadabra", "hello":"world"} ] )
{
  1: 2,
  3: 4,
}
{
  hello: world,
  abra: cadabra,
}
```

12. You type an expression to the interpreter, and you get the following output:

```
>>> ... you typed your expression here ...
{
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "printdicts.py", line 4, in print_many_dicts
    print_dict(dict)
  File "printdicts.py", line 10, in print_dict
    print '  ' + str(k) + ': ' + str(dict[k]) + ','
TypeError: list indices must be integers, not str
```

Write the smallest expression that you can that produces this exact output.

`print_many_dicts([["a"]])`

13. After typing a different expression to the interpreter, you get the following output. The *only* difference from the previous problem is the last line of output.

```
>>> ... you typed your expression here ...
{
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "printdicts.py", line 4, in print_many_dicts
    print_dict(dict)
  File "printdicts.py", line 10, in print_dict
    print '  ' + str(k) + ': ' + str(dict[k]) + ','
IndexError: list index out of range
```

Write the smallest expression that you can that produces this exact output.

print_many_dicts([[3]])

14. In no more than one sentence, explain the following Python error and how it could arise:

    ```
    TypeError: unhashable type: 'list'
    ```

    A list is mutable, but you have tried to use it in a context where only immutable objects are permitted, such as in a set or as a dictionary key.

    It is wrong to state that you are trying to convert a list into a set (that's not a problem: for example, `set([1,2])`***).***

    It is wrong to state that you are trying to modify a list or anything else.

5 Write Python code

15. Write the body of the max_even function.

```
def max_even(lst):
   """Return the largest number in the input that is an even number.
      The input must be a list of integers.
      If there is no even number in the input, return None.
      Examples:
        max_even([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 9])  =>  6
        max_even([3, -1001])  =>  None
   """

  result = None
  for i in lst:
    i%2 == 0:
      if result == None or i > result:
        result = i
  return result
```

Another solution:

```
  evens = []
  for value in lst:
    if value % 2 == 0:
      evens.append(value)
  if len(evens) == 0:
    return None
  else:
    return max(evens)
```

Yet another solution:

```
  evens = []
  for value in lst:
    if value % 2 == 0:
      evens.append(value)
  max = None
  for value in evens:
    if max == None or value > max:
      max = value
  return max
```

16. Write the body of a function that prints each word in a given file, on its own line. For example, if you were to execute `print_words("gettysburg.txt")` where file `gettysburg.txt` is as follows:

```
   Four score    and   seven
     years

ago, our
```

then the output would be

```
Four
score
and
seven
years
ago,
our
```

We have given you the first line of the solution.

```
def print_words(filename):
  """Print each word of the given file, each on its own line."""
  file = open(filename, "r")

  for line in file:
    for word in line.split():
      print word
```

Another solution:

```
  for word in file.read().split():
    print word
```

In the next two problems, suppose that you are given the following data structure.

```
home_runs = [
    ("Babe Ruth",       47,     1926),
    ("David Ortiz",     54,     2006),
    ("Albert Pujols",   47,     2009),
    ("Alex Rodriguez",  54,     2007),
    ("Babe Ruth",       49,     1930),
    ("Alex Rodriguez",  47,     2003),
    ("Babe Ruth",       54,     1920),
    ("Sammy Sosa",      49,     2002)
    ]
```

17. Write code that will print this list, sorted alphabetically by first name and (when the name is the same) by date. In particular, your code should print this data structure (though the formatting, such as line breaks, might be different):

```
[('Albert Pujols', 47, 2009),
 ('Alex Rodriguez', 47, 2003),
 ('Alex Rodriguez', 54, 2007),
 ('Babe Ruth', 54, 1920),
 ('Babe Ruth', 47, 1926),
 ('Babe Ruth', 49, 1930),
 ('David Ortiz', 54, 2006),
 ('Sammy Sosa', 49, 2002)]
```

Don't define a function — just write statements that achieve the goal.
Don't do anything special for formatting — just use Python's `print` statement, applying it to a list of tuples.

```
import operator
home_runs.sort(key=operator.itemgetter(0, 2))
print home_runs
```

18. Write code that will print this list, sorted in decreasing order by number of home runs and (when the number of home runs is the same) by increasing date. In particular, your code should print this data structure (though the formatting, such as line breaks, might be different):

```
[('Babe Ruth', 54, 1920),
 ('David Ortiz', 54, 2006),
 ('Alex Rodriguez', 54, 2007),
 ('Babe Ruth', 49, 1930),
 ('Sammy Sosa', 49, 2002),
 ('Babe Ruth', 47, 1926),
 ('Alex Rodriguez', 47, 2003),
 ('Albert Pujols', 47, 2009)]
```

Don't define a function — just write statements that achieve the goal.
Don't do anything special for formatting — just use Python's `print` statement, applying it to a list of tuples.

```
import operator
home_runs.sort(key=operator.itemgetter(2))
home_runs.sort(key=operator.itemgetter(1), reverse=True)
print home_runs
```