CS 115 Exam 3, Fall 2011

Your name: _____

Rules

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only resource you may consult during this exam.
- Explain/show work if you want to receive partial credit for wrong answers.
- As long as your code is correct, you will get full credit. No points for style.
- When you write code, be sure that you clearly indicate the indentation level of each statement.

Grade (instructor use only)

	Your Score	**Max Score**
Problem 1		25
Problem 2		15
Problem 3		15
Problem 4		25
Problem 5		20
Total		100

Problem 1: 25 points.

Consider the following useless Python program:

```
class X:
    def __init__(self, a):
         self.b = a
         self.c = 0
    def f(self):
          return self.b + self.c
    def g(self, d)
          return self.f() + d

x1 = X(4) # Statement 1
x2 = X(0) # Statement 2
print(x1.f()) # Statement 3
print(x2.g(2)) # Statement 4
```

(a) List all of the *instance variables* created in this code.

(b) List all of the *methods* defined in this code.

(c) List at least one of the *objects* created in this code.

(d) List all of the *classes* defined in this code.

(e) Which methods are called during the execution of Statement 2?

(f) Which methods are called during the execution of Statement 4?

Assume that there is a Python dictionary called `bonuses` that stores the end-of-year bonuses of some employees. For example:

```
bonuses = {'John': 3000, 'Paul': 1500, 'George': 2000}
```

Note: do NOT assume that these are the actual keys and values in the dictionary. This is just an example.

Assume that the variable `employee` contains the name of an employee, and the variable `base` contains that employee's base salary.

Using the `bonuses` dictionary, print that employee's total salary. For a given employee:

Total salary = base salary + bonus (if any).

Problem 2: 15 points.

Consider the following unsorted list:

```
breakfast = ['coffee',
             'pancakes',
             'waffles',
             'juice',
             'banana',
             'yogurt',
             'cereal',
             'bacon',
             'toast',
             'donuts']
```

and the following selection sort (which is essentially identical to your lab code):

```
def selection_sort(list_to_sort):
   for i in range(len(list_to_sort) - 1):
     min_index = find_min_index(list_to_sort[i:]) + i
     list_to_sort[i], list_to_sort[min_index] =
           list_to_sort[min_index], list_to_sort[i]

def find_min_index(L):
   return L.index(min(L))
```

In the diagrams on the next page, show the contents of the 10-element array after each of the first 4 iterations of the *for*-loop in `selection_sort`. For your reference, the sorted version of the list appears in Question 3 on the following page.

If the list does not change from one iteration to the next, you can write "SAME" for the next iteration.

INITIAL ORDER	AFTER i=0 ITERATION	AFTER i=1	AFTER i=2	AFTER i=3
coffee				
pancakes				
waffles				
juice				
banana				
yogurt				
cereal				
bacon				
toast				
donuts				

Problem 3: 15 points.

Consider the following sorted list:

```
breakfast = ['bacon',
             'banana',
             'cereal',
             'coffee',
             'donuts',
             'juice',
             'pancakes',
             'toast',
             'waffles',
             'yogurt']
```

and the following binary search (which is essentially identical to your lab code):

```
# binary_search()
# Finds the position of an item in a list
# Parameters: the list; the item to search for
# Returns: the item's position (or None)
def binary_search(search_list, value_to_find):
     first = 0
     last = len(search_list) - 1

     while first <= last:
          middle = (first + last) // 2
          if value_to_find == search_list[middle]:
               return middle
          elif value_to_find < search_list[middle]:
               last = middle - 1
          else:
               first = middle + 1
     return None
```

You may want to label the elements of `breakfast` with their numeric index values before proceeding.

Answer the questions on the next page.

(a) Fill out the following table tracing a binary search for *waffles* in this list. You should fill out one row per iteration of the loop. If there are more rows than iterations, leave the extra rows blank.

In the *Compare To:* column, you should give the VALUE (the name of the breakfast item) of the list element that will be compared to *waffles*.

Old value of `first`	**Old value of `last`**	**Compare to (e.g. *pancakes)*:**
0	9	

(b) Fill out the following table tracing a binary search for *granola* in this list.

Old value of `first`	**Old value of `last`**	**Compare to (e.g. *pancakes)*:**
0	9	

Problem 4: 25 points.

In this problem, you will define a class to represent a coin purse. A coin purse can contain the following types of coins:

- Pennies: worth 1 cent ($0.01) each
- Dimes: worth 10 cents each
- Quarters: worth 25 cents each

No methods in this class should use input() or print()!

Your class should be named `CoinPurse`, and you should define the following methods:

`__init__`: This method initializes the coin purse.

- Parameters: 3 parameters for the initial contents of the purse -- the number of pennies, dimes, and quarters.
- Should initialize an instance variable for each type of coin using the values that are passed to it.

`get_total`: Returns the total amount of money in the purse, in dollars.

`add_money`:

- Parameters for the numbers of pennies, dimes, and quarters to add to the current amounts in the purse.
- Updates the amounts in the purse.
- Does not return anything.

`pay`:

- Parameters for the numbers of pennies, dimes, and quarters to remove from the purse.
- Updates the amounts in the purse. If the number of coins of a specific type (e.g. pennies) to remove is greater than the amount that was in the purse, then pay out all of the coins of that type and set the new amount to 0.
- Returns the dollar amount that was actually paid.

Problem 5: 20 points.

For this problem, you must write a **complete program**. That includes a docstring, a `def main()`, any necessary library imports, etc.

You can assume that the `CoinPurse` class from Problem 4 has already been correctly defined.

Read the instructions carefully before you start coding!

Your program should contain the following:

1. A function called `MakePurse` to do the following:
    - Ask the user to enter 3 integers (pennies, dimes, and quarters)
    - Print an error message and exit the program if any of the user's inputs could not be converted to a valid integer.
    - Otherwise, create a `CoinPurse` object with those values.
    - Return the newly created `CoinPurse`

2. A main function that does the following:
    - Call `MakePurse` repeatedly to create and populate a list of `CoinPurse` objects.
    - Continue creating `CoinPurse` objects until the user creates a `CoinPurse` with 0 coins in it.
    - After the list has been created, ask the user for 3 more values: the numbers of pennies, dimes, and quarters to steal from each purse. You can assume these are valid integers.
    - Using the methods of `CoinPurse`, attempt to steal those coins from each purse in the list.
    - Compute and print the total dollar amount that was actually stolen.