Department of Computer Science
CSC 326H1F – Fall 2008

Test 1

Aids allowed: one 2-sided sheet

Time: 50 minutes

Total marks: 48

Name: _____

Student number: _____

1	/4
2	/15
3	/25
4	/4
Total	**/48**

1. [4 marks]

Recall that *name mangling* in Python involves using method names and instance variable names that start with two underscores (e.g. `__m()`). One thing this accomplishes is preventing naming conflicts. What is the more general purpose of name mangling? In what way is name mangling of limited usefulness (compared to what is available in other languages)?

Name mangling is used to indicate that a particular class member (variable/method) should be treated as if it's private. [2 marks]

Name mangling is of limited usefulness because it does not actually enforce privacy (unlike the `private` keyword in C++/Java). [2 marks]

2. [15 marks]

For each of the following code fragments, give (all of) the output that is produced. If the code fragment produces an error, give all the output until the point at which the error occurs, and then clearly explain the error. When explaining an error, you are not required to provide the actual error message.

(a) [2 marks]

```
s = "computer science"
print s[0:4]
print s[-3:99]
```

`comp` **[1 mark]**
`nce` **[1 mark]**

(b) [2 marks]

```
s = "computer science"
s[0:8] = "engineering"
print s
```

The line `s[0:8] = "engineering"` produces an error. [1 mark]
This line attempts to modify a string, but in Python, string objects are immutable (cannot be modified). [1 mark]

(c) [2 marks]

```
L = [5, 10, 15, 20]
L[2:3] = ['lions', 'tigers', 'bears']
print L[2:3]
```

`['lions']` [2 marks]

(d) [3 marks]

```
L = [2, 4, [8, 16]]
N = L[2]
L[2] = [3, 6]
P = L[2][0]
P = 5
print N
print L.pop()
```

`[8, 16]` [1 mark]
`[3, 6]` [2 marks]

(e) [3 marks]

```
def magic(x, y):
   z = 10
   x = [0]
   y[1] = z+2

z = 20
L = [7, 14, 21]
M = [9, 18, 27]
magic(L, M)
print z
print L
print M
```

20 **[1 mark]**
[7, 14, 21] **[1 mark]**
[9, 12, 27] **[1 mark]**

(f) [3 marks]

```
def greeting(name='student'):
   name = 'Hi ' + name + '!'
   print name

greeting()
greeting('Thing 1')
greeting()
```

Hi student! **[1 mark]**
Hi Thing 1! **[1 mark]**
Hi student! **[1 mark]**

3. [25 marks]

This question involves creating a class representing a bank. A bank has clients, each of whom has a particular amount in their account at the bank.

(a) [5 marks]

Write a class called **`Bank`**. The constructor should take a list of the bank's clients and a list storing the amount of money in each client's account as parameters. These two lists should be stored as instance variables named **`clients`** and **`amounts`**. Do not create any other methods (yet). The following code fragment illustrates how your class will be instantiated:

```
b = Bank(['Elmer', 'Alice', 'Sue'], [2000, 4000, 3500])
```

```
class Bank(object):                    [1 mark]

   def __init__(self, C, A):           [2 marks]
      self.clients = C                 [1 mark]
      self.amounts = A                 [1 mark]
```

(b) [5 marks]

Banks regularly deduct fees from each account. Write an instance method called **`deductFee`** for your **`Bank`** class. This method should take an amount as a parameter, and then reduce the amount of money in each account by this amount. For example, if `b` is an instance of **`Bank`**, then the call `b.deductFee(20)` should subtract 20 from each account in `b`. You may assume that each account has enough money to pay the fee.

```
   def deductFee(self, M):                    [1 mark]

      for i in range(len(self.amounts)):      [3 marks]
         self.amounts[i] -= M                 [1 mark]
```

(c) [3 marks]

Define an exception called **`NoAccounts`**. Your exception should not take any arguments.

```
class NoAccounts(Exception):        [2 marks]
   pass                             [1 mark]
```

(d) [8 marks]

Banks like to know who their wealthiest client is. Write an instance method called **`richest`** for your **`Bank`** class. This method should return the name of the client whose account has the most money. To simplify things, you may assume no two clients have the same amount of money in their accounts; this means that your method always returns a single name. If the bank has no clients, raise a **`NoAccounts`** exception. The following code fragment illustrates how your method will be used:

```
b = Bank(['Elmer', 'Alice', 'Sue'], [2000, 4000, 3500])
b.richest()   # 'Alice'
c = Bank([], [])
c.richest()   # NoAccounts exception is raised
```

```
def richest(self):

   if len(self.amounts) == 0:
      raise NoAccounts

   name = self.clients[0]
   money = self.amounts[0]

   for i in range(1, len(self.amounts)):
      if self.amounts[i] > money:
         money = self.amounts[i]
         name = self.clients[i]

   return name
```

(e) [4 marks]

Write an appropriate instance method to overload the indexing operator for your **`Bank`** class, so that if `b` is an instance of **`Bank`** then `b[i]` returns a **tuple** consisting of the name of the `i`-th customer *and* the amount of money in that customer's account. For example:

```
b = Bank(['Elmer', 'Alice', 'Sue'], [2000, 4000, 3500])
b[1]   # ('Alice', 4000)
b[0]   # ('Elmer', 2000)
```

Your method may assume that it is always given a valid index; you are not required to raise any exceptions.

```
def __getitem__(self, i):                          [2 marks]
    return (self.clients[i], self.amounts[i])      [2 marks]
```

4. [4 marks]

Below is a table of regular expressions `RE` (in the first column) and strings `S` (in the second column). In the third column, write `'Y'` if the regular expression in column 1 matches the string in column 2, or `'N'` if not. If you wrote `Y` in the third column, then in the fourth column write the part of `S` that is matched by `RE`.

The first line of the table is completed for you. As that example shows, `RE` is **not** required to match the beginning of `S`. We are behaving like Python's `re.search()` function and looking for the leftmost match.

Helpful reminders: `'^'` matches the beginning of `S`. `'$'` matches the end of `S`.
`'.'` matches any character. `'*'` means to match 0 or more of the preceding pattern.
`'+'` means to match 1 or more or the preceding pattern.
`'[ ]'` matches any character listed between the brackets.

RE	**S**	**Succeeds? (Y/N)**	**match**
abc	dabc	Y	abc
b.*b	bcbcabcbca	**Y**	**bcbcabcb** **[1 mark]**
a[dc]+	accadcdcba	**Y**	**acc** **[1 mark]**
^.*lo$	Hello!	**N**	**[1 mark]**
0[0123]*1$	100010101	**Y**	**00010101** **[1 mark]**