UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2011 EXAMINATION

CSC108H1S

St. George Campus

Duration -- 3 hours

Examination Aids: No Exam Aids Allowed

Student #: ____________

Last Name:____________ **First Name:**____________

Do **not** turn this page until you have received the signal to start. In the meantime, please fill out the identification section above, and read the instructions below carefully.

This term test consists of 10 questions on 15 pages (including this one), printed on one side of the paper. When you receive the signal to start, please make sure that your copy of the examination is complete.

Answer each question directly on the examination paper, in the space provided, and use the reverse side of the pages for rough work. If you need more space for one of your solutions, use the reverse side of the page and indicate clearly the part of your work that should marked.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

Please note that you are **required** to write docstrings for every function unless otherwise specified.

#1:	______/10
#2:	______/10
#3:	______/10
#4:	______/10
#5:	______/10
#6:	______/10
#7:	______/10
#8:	______/10
#9:	______/10
#10:	______/10
***Total*:**	____/100

Question 1 [*10 marks total*]

For each of the following, circle the correct answer.

I. Which of the following is not true about dictionaries?
a) Keys must be immutable
b) All entries are key:value pairs
c) Specific entries are accessed by index position
d) They are defined with { }

II. `a = return print("CS IS FUN")`
What is the value of `a`?
a) there is no value because this line of code produces an error
b) `"CS IS FUN"`
c) `print("CS IS FUN")`
d) `None`

III. Which of the following are NOT true about conditional statements?
a) You can have unlimited `elif` statements for every `if`
b) You can only have one `else` per `if`
c) Each `if` must have an `elif` or `else`
d) `elif` statements must specify a Boolean expression

IV. To write a while loop that is not infinite you must:
a) specify a terminating condition
b) have a colon at the end of the while statement line
c) move towards the terminating condition in the loop
d) know how many iterations will be executed
e) a, b, and c
f) All of the above

V. For each of the following, specify whether the statement is (T)rue or (F)alse

T / F Tuples are mutable data types.

T / F Methods can only be called on objects that are instances of the class in which the method is written.

T / F The following 2 lines of code make two variables that have references which both point to the same object.

```
hwA = Homework()
hwB = Homework()
```

T / F Given the following code, the final value of `student1.name` is `"Bob"`

```
student1 = Student()
student2 = student1
student1.name = "Bob"
student2.name = "Fred"
```

Question 2 [*10 marks total*]

Your classmate wrote the simple module shown to the right and put it in a file ***grp.py***. When she used it, she got the following error. Explain why this happened. (Don't just paraphrase the error message, but explain why Python was confused.)

```
group = [ ]
size = 0
def add(new):
    group.append(new)
    size = size + 1
```

```
>>> import grp
>>> grp.add('alice')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "grp.py", line 6, in add
size = size + 1
UnboundLocalError: local variable 'size' referenced
before assignment
```

Answer:

Next, she modified the file as shown in the second box below, re-started Python, and entered the following four lines:

```
1.>>> import grp
2.>>> group = grp.group
3.>>> size = grp.size
4.>>> add = grp.add
```

```
group = [ ]
size = 0
def add(new):
    global size
    group.append(new)
    size = size + 1
```

She added lines 2-4 above to make her life easier by adding aliases for the variables and function in the ***grp*** module, so that she could refer, for example, to ***grp.group*** as just ***group***. It seemed to work at first, but when she checked on the group and its size, she was surprised and confused. Explain what's going on. Your classmate wrote:

```
>>> group
[ ]
>>> size
0
>>> add('alice')
>>> group
['alice']
>>> size
0
>>> grp.group
['alice']
>>> grp.size
1
```

Answer:

Question 3 [*10 marks total*]

Write a function named **`uniqueChars`** that takes in a string as a parameter. The purpose of the function is to create a list of all the unique characters in the string in the order in which they appear in the string parameter. For this question, an upper case character is the same as the lower case character (so if you had the string `"Abbbbba  Baaaa"`, the resulting list would be `['A','b',' ']`, not `['A', 'b', 'a', 'B', ' ']`); you must preserve the case of the first instance of the character in your list. Once you have finished retrieving all the unique characters from the string, you should return the list of unique characters.

Sample invocation:

```
>>> uniqueChars("Hello World")
['H', 'e', 'l', 'o', ' ', 'W', 'r', 'd']
```

Answer:

Question 4 [*10 marks total*]

Assume that a function **`bubble_sort(lst)`** is already implemented. If you invoke with the function with the list `[9,4,5,8,1,7,2,6,4]` you will get:

```
>>> bubble_sort([9,4,5,8,1,7,2,6,4])
[1,2,4,4,5,6,7,8,9]
```

Write down the list content after each full iteration on the elements until the list is sorted.

Answer:

Question 5 [*10 marks total*]

Write a function `dropComments(inputfilename,outputfilename)` that reads in a python .py source file and copies the file source code but drops all docstrings and comments from the file.

Sample invocation:

```
>>> dropComments("infile.py", "outfile.py")
```

infile.py

```
def get_value():
    '''Docstring for get_value'''
    X = input("Enter value")
    Y = X * X # assign to Y
    # printing message to user
    print "X squared is", Y
```

outfile.py

```
def get_value():
    X = input("Enter value")
    Y = X * X
    print "X squared is", Y
```

Answer:

Question 6 [*10 marks total*]

Write a python function `build_index(filename)` that counts the frequencies of each word in a text file, and prints each word with its count and line numbers where it appears. We define a word as a contiguous sequence of non-white-space characters. Different capitalizations of the same character sequence should be considered the same word, e.g. Python and python should be treated as the same word. Only alphabet characters and white spaces will be present in the input file. The output is formatted as follows: each line begins with a number indicating the frequency of the word, a white space, then the word itself, and a list of line numbers containing this word. Hint: use a dictionary to store words and the string's `split()` method to split a line.

Sample invocation:

```
>>> build_index("file1.txt")
2 This 1 2
2 is 1 2
1 CSC108 1
2 exam 1 2
1 file 2
1 for 2
1 Question 1
1 six 1
1 of 1
```

file1.txt

```
This is CSC108 exam
This file is for Question six of exam
```

Answer:

Question 7 [*10 marks total*]

The following definitions have been entered into a Python shell:

```
class Account():
    chargeRate = 0.01

    def __init__(self, start):
        self.value = start

    def debit(self, amount):
        debitAmt = min(amount, self.value)
        self.value = self.value - debitAmt
        return debitAmt

    def deposit(self, amount):
        self.value += amount

    def fee(self, baseAmt):
        self.debit(baseAmt * self.chargeRate)

    def withdraw(self, amount):
        if self.value >= 10000.0:
            self.fee(amount/2.0)
        else:
            self.fee(amount)
        return self.debit(amount)


class Chequing(Account):
    chargeRate = 0.05
    def deposit(self, amount):
        if self.value <= 1:
            Account.deposit(self,
                     (1-self.chargeRate) * amount)
        else:
            Account.deposit(self, amount)
```

Assume that the following expressions have been evaluated:

```
>>> Eric  = Chequing(4000.0)
>>> Ellen = Account(4000.0)
```

(Continues on the next page)

Write the values of the following expressions. Write **None** when there is no value; write **Error** when the expression results in an error and explain briefly why it's an error. Assume that these expressions are evaluated one after another.

```
>>> Eric.withdraw(3000.0)


>>> Eric.value


>>> Eric.withdraw(1000.0)


>>> Eric.value


>>> Eric.deposit(5000.0)


>>> Eric.value


>>> Ellen.withdraw(3000.0)


>>> Ellen.value


>>> Ellen.withdraw(1000.0)


>>> Ellen.value


>>> Ellen.deposit(5000.0)


>>> Ellen.value
```

Question 8 [*10 marks total*]

A. Define a class **Counter** with an instance variable x. When instantiated, **Counter** takes no argument, and x is initialized to 0. When an instance of **Counter** appears where a string is expected, it is converted to the string representation of the value of x. **Counter** has an instance method **count()**, which takes no argument and increments x by one.

Sample invocation:

```
>>> import prob2
>>> c = prob2.Counter()
>>> print c
0
>>> c.count()
>>> c.count()
>>> print c
2
```

Answer:

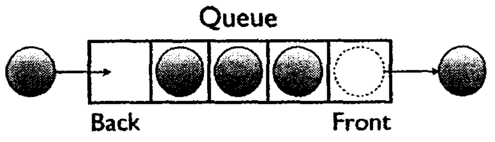
B. Now, define two classes that inherit from **Counter**. Class CounterAdd introduces the instance method add(), which, when passed a number, adds that number to x. Class CounterReset introduces the instance method reset(), which is invoked with no argument and sets x back to 0. The following session provides examples:

```
>>> import prob2
>>> ca = prob2.CounterAdd()
>>> ca.count()
>>> ca.add(10)
>>> print ca
11
>>> cr = prob2.CounterReset()
>>> cr.count()
>>> cr.count()
>>> print cr
2
>>> cr.reset()
0
```

Answer:

Question 9 [*10 marks total*]

In this question, you are going to write a class called `Queue` that simulates the behavior of a queue. This type of data structure is very useful in simulating real-life queues (see figure below):



When items are added to a queue, they are appended at the back of the queue. When items are removed, they are removed one at a time from the front of the queue. This way the queue maintains fairness since the first element in is the first element out. Complete the implementation of the class `Queue` below. Docstrings are provided to guide you in the implementation. Think carefully about which of the Python data structures you've studied is most suitable for storing the elements of a queue.

```
class Queue():
    def __init__(self):
        '''Initialize an empty queue'''




    def enqueue(self,Item):
        '''Add Item to the back of the queue'''




    def dequeue(self):
        '''Remove and return the item at
        the front of the queue'''




    def getQueueSize(self):
        '''Return the number of elements
        in the queue'''
```

Question 10 [*10 marks total*]

Two words are called anagrams if one word can be created by rearranging the characters in the other (regardless of capitalization). For example APPLE and PLeaP are anagrams. Your friend is writing a program to see if two words are anagrams but it's not quite working.

```
def is_anagram(word1, word2):
    # check that lengths are the same
    if len(word1) != len(word2):
        return False
    # check that everything in word1 is in word2
    for c in word1:
        if c not in word2:
            return False
        #check that everything in word2
        #is in word1
    for c in word2:
        if c not in word1:
            return False
    return True
```

Using `assert` to generate test cases, answer the following to help your friend test the function:

(a) Provide a test case for a pair of anagrams for which the function would return `False` (a false negative result)

(b) Why does the code return that they are not anagrams when they are?

(c) Provide a test case for a pair of non-anagram strings for which the function would return `True` (a false positive result)

(d) Why does the code return that they are anagrams when they are not?

(e) Provide a test case for a pair of non-anagram strings for which the function would return `False` (a correct negative).

END OF EXAMINATION

Appendix: Short Python function/method descriptions:

len(x) -> integer
    Return the length of the list, tuple, dict, or string x.
max(L) -> value
    Return the largest value in L.
min(L) -> value
    Return the smallest value in L.
open(name[, mode]) -> file object
    Open a file. Legal modes are "r" (read), "w" (write), and "a" (append).
range([start], stop, [step]) -> list of integers
    Return a list containing the integers starting with start and ending with stop - 1 with step specifying the amount to increment (or decrement). If start is not specified, the list starts at 0. If step is not specified, the values are incremented by 1.
float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.
int(x) -> integer
    Convert a string or number to an integer, if possible. A floating point argument will be truncated towards zero.

dict:
D.get(k) -> value
    Return the value associated with the key k in D.
D.has_key(k) -> boolean
    Return True if k is a key in D and False otherwise.
D.keys() -> list of keys
    Return the keys of D.
D.values() -> list of values
    Return the values associated with the keys of D.
D.items() -> list of (key, value) pairs
    Return the (key, value) pairs of D, as 2-tuples.

file:
F.close()
    Close the file.
F.read([size]) -> read at most size bytes, returned as a string.
    If the size argument is negative or omitted, read until EOF (End of File) is reached.
F.readline([size]) -> next line from the file, as a string. Retain newline.
    A non-negative size argument limits the maximum number of bytes to return (an incomplete line may be returned then). Return an empty string at EOF.

list:

L.append(x)
    Append x to the end of the list L.
L.index(value) -> integer
    Returns the lowest index of value in L.
L.insert(index, x)
    Insert x at position index.
L.pop([index]) -> item
    Remove and return item at index (default last).
L.remove(value)
    Remove the first occurrence of value from L.
L.sort()
    Sorts the list in ascending order.

str:

str(x) -> string
    Convert an object into its string representation, if possible.
S.find(sub[,i]) -> integer
    Return the lowest index in S (starting at S[i], if i is given) where the string sub is found or -1 if sub does not occur in S.
S.index(sub) -> integer
    Like find but raises an exception if sub does not occur in S.
S.isdigit() -> boolean
    Return True if all characters in S are digits and False otherwise.
S.lower()
    Return a copy of the string S converted to lowercase.
S.replace(old, new) -> string
    Return a copy of string S with all occurrences of the string old replaced with the string new.
S.rstrip([chars]) -> string
    Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead.
S.split([sep]) -> list of strings
    Return a list of the words in S, using string sep as the separator and any whitespace string if sep is not specified.
S.strip() -> string
    Return a copy of S with leading and trailing whitespace removed.
S.upper()
    Return a copy of the string S converted to uppercase.

END OF APPENDIX