UNIVERSITY OF TORONTO

Fall 2011 Final Exam -- Solutions

CSC180: Introduction to Computer Programming

Duration: 2.5 hours

December 13th, 2011

Last Name: _____

First Name: _____

Student Number: _____

Instructor (circle one): **F. Pitt** **S. Engels**

Instructions:

- **Write your name on the back of this exam paper.**
- **Do not open this exam until you hear the signal to start.**
- Have your student ID on your desk.
- No aids permitted other than writing tools. Keep all bags and notes far from your desk during the exam.
- There are 6 questions on 14 pages (including an API sheet). When you hear the signal to start, ensure that your exam is complete before you begin.
- Read over the entire exam before starting.
- Write comments where it would clarify your code.
- If you use any space for rough work, clearly indicate the section(s) that you want marked.

Mark Breakdown

Question 1:	**/12**
Question 2:	**/18**
Question 3:	**/10**
Question 4:	**/15**
Question 5:	**/20**
Question 6:	**/10**
Bonus:	**/1**
Total:	**/ 80**

Question 1: Short Answer (12 marks)

Answer the following questions in the space provided. When providing a written answer, write **as clearly and legibly as possible**. Marks will not be awarded to unreadable answers.

1. Which of the following has a `__str__` method? Circle all that apply.

a) dictionary **b)** string

c) integer **d)** list

2. True or False? Exceptions are a type of class.

True **False**

3. What is a possible structure for storing the QR code grid in Project 3? Circle all that apply.

a) a list of lists **b)** a dictionary

c) a list of tuples **d)** a string

4. True or False? Any class you create will inherit from some other class.

True **False**

5. True or False? The effect of the `import` statement is to copy and paste the contents of the imported file into the top of your code.

True **False**

6. Fill in the blank line with a single line of code that prints out the name entered by the user.

```
class Info:
    def __init__(self, value):
        self.value = value

x = Info(raw_input('Enter your name: '))
print x.value
```

7. Which of the following is not a valid variable name? Circle all that apply.

a) `class` **b)** `second-last`

c) `list` **d)** `for`

8. True or False? Mergesort always takes `n*log n` running time, whether it's sorted or not.

True **False**

9. If the input list is already sorted, how many swaps will insertion sort have to do?

Zero

10. True or False? Making class `A` inherit from class `B` causes `B`'s values to be copied into `A`.

True **False**

11. Circle all statements that could cause the following error:

```
NameError: name 'x' is not defined
```

a) `x += 1` **b)** `x = name`

c) `print x` **d)** `x = 5 / 0`

12. Which of the following could cause the following error:

```
TypeError: 'NoneType' object is unsubscriptable
```

a) assigning a value to None **b)** using None like a list

c) opening None like a file **d)** using None like an

Question 2: Exceptions (18 marks)

Consider the code below for converting decimals to binary numbers:

```
def convert_to_binary(value, length):
    """Convert the input integer value into a binary
    string of the given length, and return the result.
    """

    result = ""
    for index in range(0, length):
        digit = value % 2
        result = str(digit) + result
        value /= 2
    return result
```

The code works, but we want to enhance it to handle error conditions in the value provided by the user. The function should raise an exception in any of the following cases:

- The value is not an integer
- The value is negative
- The value is too big to be stored in a binary number of the given length

a) (6 marks) In the space below, write the definition for an `InvalidInputException` class, which is raised whenever one of the above cases arise. This class must inherit from the `Exception` class, and override the following methods:

- an `__init__` method that takes in a message string and stores it in the object.
- a `__str__` method that returns: `"Invalid input -- "`, followed by the stored message string.

```
class InvalidInputException(Exception):
    '''The class for storing invalid input exceptions. '''

    def __init__(self, message):
        self.message = message

    def __str__(self):
        return 'Invalid input -- ' + message
```

Question 2 (continued)

b) (6 marks) In the space below, rewrite the function `convert_to_binary` from the previous page, with code in appropriate places that will raise `InvalidInputException`s. Make sure that the message stored in each exception reflects the case that caused the exception.

```
def convert_to_binary(value, length):
    """Convert the input integer value into a binary
    string of the given length, and return the result.
    """

    if not isinstance(value, int):
        raise InvalidInputException('Value is not an integer')
    if value < 0:
        raise InvalidInputException('Value is negative')
    result = ""
    for index in range(0, length):
        digit = value % 2
        result = str(digit) + result
        value /= 2
    if value != 0:
        raise InvalidInputException('Insufficient length')
    return result
```

c) (6 marks) In the space below, write code that will prompt the user for a decimal number, and then print out that value as a 10-digit binary number. If the input value is invalid, print a message (like the following for `-1`):

```
'The value -1 could not be converted.'
```

```
value = raw_input('Please enter a decimal value: ')
try:
    binary_value = convert_to_binary(int(value), 10)
    print binary_value
except:
    print 'The value %s could not be converted.', (value)
```

Question 3: Class Creation (10 marks)

Consider the Python console interactions on the right.

In the space below, write code to make the lines on the right execute without error and produce the results shown.

Comments and docstrings are not required, though they may help us give you part marks if your code is not 100% correct.

Note: when a luggage order is asked for its destination, it usually returns the destination that it was created with. Note the two exceptions to this:

- if no destination has been assigned, return `'parts unknown'`
- if the order is made of multiple pieces of luggage, return `'multiple destinations'`

```
>>> a1 = Luggage('Steve', 23.5)
>>> print a1
Steve's luggage: 23.5lbs.
>>> a2 = Luggage('Francois', 12, `Paris')
>>> str(a2)
"Francois's luggage: 12lbs."
>>> a2.destination()
'Paris'
>>> a1.destination()
'parts unknown'
>>> a = a1 + a2
>>> print a
Steve & Francois's luggage: 35.5lbs.
>>> a.destination()
'multiple destinations'
>>> print (a1 + 1.5)
Steve's luggage: 25lbs.
>>> a3 = Luggage()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: __init__() takes at least 3
arguments (1 given)
```

```
class Luggage:
    """ Luggage has an owner, weight, and optional destination.
    """

    def __init__(self, owner, weight, destination='parts unknown'):
        self.owner = owner
        self.weight = weight
        self.dest = destination

    def __str__(self):
        return "%s's luggage, weighting %glbs." % (self.owner, self.weight)

    def __add__(self, other):
        if isinstance(other, Luggage):
            return Luggage(self.owner + " and " + other.owner,
                           self.weight + other.weight,
                           'various places...')
        elif (isinstance(other, float) or
              isinstance(other, long) or
              isinstance(other, int)):
            return Luggage(self.owner, self.weight + other, self.dest)

    def destination(self):
        return self.dest
```

Question 4: Sorting (15 marks)

Consider the lines of code on the right.

When arranged in the correct order, this code defines a function that takes in a list of items arranges the contents of the list in sorted ord

In the space below, arrange these lines in the correct order so that they perform the require sorting operation. Make sure to provide the proper indenting, and include comments and docstrings that describe what the code is doi

Note: You must use all of the lines that we pr Other than comments, you may not add new and you may not modify these lines in any wa except their order and indentation.

```
else:
pivot = items[0]
left.append(item)
right.append(item)
items[len(left)+1:] = right
if len(items) > 1:
left, right = [], []
quicksort(left)
if item < pivot:
items[len(left)] = pivot
quicksort(right)
def quicksort(items):
for item in items[1:]:
items[:len(left)] = left
```

```
def quicksort(items):
    '''Sort the list of items from smallest to largest.
    '''
    if len(items) > 1:
        pivot = items[0]
        left, right = [], []
        for item in items[1:]:
            if item < pivot:
                left.append(item)
            else:
                right.append(item)
        quicksort(left)
        quicksort(right)
        items[:len(left)] = left
        items[len(left)] = pivot
        items[len(left)+1:] = right
```

Question 5: Super Duper Question (15 marks)

For this question, you will write two Python classes:

- `Spectrum` – which stores a bunch of chemical elements
- `Element` – which represents an individual element.

a) (5 marks) In the space below, write the `Element` class, which represents a single chemical element. This class will store the following two values:

- the element's atomic number (as a positive integer)
- the name of the element (as a string)

Note: we're simplifying things greatly here and ignoring most of the other information usually associated with chemical elements!

Your class must implement the following methods, along with any other helper methods you consider useful:

- `__init__`: initialize this element with the number and name provided (both required arguments).
- `__str__`: return a string representation of this element, in the form: `"(number)name"`
- `__cmp__`: return a negative integer if this element is smaller than the other element, zero if this element is equal to the other element, and a positive integer if this element is greater than the other element. This method compares elements by their number alone, ignoring their name.

Comments and docstrings are not required, but may help give you part marks if your code is not 100% correct.

```
class Element:
    """ A single chemical element (simplified to atomic number and name).
    """

    def __init__(self, number, name):
        self.number = number
        self.name = name

    def __str__(self):
        return "(%d)%s" % (self.number, self.name)

    def __cmp__(self, other):
        return self.number - other.number
```

Question 5 (continued)

b) (10 marks) In the space below, write the `Spectrum` class, which represents a collection of chemical elements. Again, comments and docstrings are not required, but are helpful if your code is confusing.

Your class must implement the following methods, along with any other helper methods you consider useful:

- `__init__`: initialize this spectrum to be empty.
- `__len__`: return the number of elements in this spectrum.
- `__str__`: return a string representation of this spectrum, listing the atomic numbers and names of the spectrum's elements. The format of each element name and number is the same as specified in Part a). Elements are listed from lowest atomic number to the highest, in a single line separated by commas.
- `add_elem`: add an element to this spectrum; if the element is already in this spectrum do **not** add it a second time (keep the spectrum unchanged).
- `get_min`: remove and return the element from this spectrum with the smallest atomic number.

Note: You are not allowed to directly access an element's attributes anywhere within the `Spectrum` class. Your interaction with the `Element` class should only be through its methods.

```
class Spectrum:
    """ A "spectrum" consists of a collection of chemical elements.
    """

    def __init__(self):
        self.elems = []

    def __len__(self):
        return len(self.elems)

    def __str__(self):
        return "[" + ", ".join(map(str, self.elems)) + "]"

    def add_elem(self, e):
        i = 0
        while i < len(self.elems) and self.elems[i] < e:
            i += 1
        if self.elems[i] != e:
            self.elems.insert(i, e)

    def get_min(self):
        return self.elems.pop(0)
```

Question 5 (continued)

c) (1 mark) What is the name of the sorting technique you used when ordering the elements of the `Spectrum` class?

Various answers, depending on implementation.

d) (4 marks) What is the running time of your `add_elem` method in the `Spectrum` class? Explain your answer.

Running time: _____

What is the running time of your `__str__` method in the `Spectrum` class? Explain your answer.

Running time: _____

Question 6: Tracing (10 marks)

Consider the piece of code on the right.

In the space below, write the output that this program will print to the screen when it is run

```
class Foo():

    def __init__(self, x=1, y=2, z=3):
        self.nums = [x, y, z]

    def __str__(self):
        return str(self.nums)

    def set(self, x):
        self.nums = x

    def copy(self, x):
        nums = []
        for item in x:
            nums.append(item)

f1 = Foo()
f2 = Foo('One', 'Two')
f3 = Foo(f1.nums)

print 'Part One: '
print f1
print f2
print f3

print 'Part Two: '
f3 = f1
f1.set(['Four', 'Five', 'Six'])
f2.copy(['Seven', 'Eight'])
print f1
print f2
print f3

print 'Part Three: '
f1 = Foo('Nine', 'Ten')
f4 = f2.set(['Eleven', 'Twelve'])
print f1
print f2
print f3
print f4
```

Part One:

[1, 2, 3]

['One', 'Two', 3]

[[1, 2, 3], 2, 3]

Part Two:

['Four', 'Five', 'Six']

['One', 'Two', 3]

['Four', 'Five', 'Six']

Part Three:

['Nine', 'Ten', 3]

['Eleven', 'Twelve']

['Four', 'Five', 'Six']

None