Final Study Problems - CSC 242

Notes

- Note that these study problems are intended to be *representative* of the problems on the final, but not *comprehensive*, ie. they do not cover all the material for which you are responsible.
- The number and difficulty of problems on the actual final may differ from these.
- I recommend that you attempt these problems with a time limit (2:15 hh:mm). After you are done, go back and fix any problems you did not solve during the given time.

Problems

`QuoteCollector`

Implement a class `QuoteCollector` that collects the quotes stored in an html document and stores them in a **dictionary** to be retrieved later with the `getQuotes` method. Details:

- `QuoteCollector` must inherit from `HTMLParser`, this gives you the `feed` method automatically
- you will need to implement some additional methods, which?
- `getQuotes` - this method **returns** a dictionary with following structure (also see example below)
  - each key is an author
  - the corresponding value is a list containing the quotes written by that author
- the quotes and their author are stored using the html `<blockquote>` tag. For example, the first quote in the file quotes.html is marked up as follows:

```
<blockquote>A lie can travel half way around the world while the truth is
putting on its shoes. -Mark Twain</blockquote>
```

  - There will always be a `'-'` between the quote and its author, and never anywhere else.
  - remove leading and trailing white space characters (spaces, newlines and tabs, at least) from quotes and author names using the string `strip` method
- **quotes.html** should be in your working directory

Sample Usage:

```
>>> qc = QuoteCollector()
>>> qc.feed( open('quotes.html').read() )
>>> qc.getQuotes()
{'Mark Twain': ['A lie can travel half way around the world while the truth is
putting on its shoes.', "If you tell the truth, you don't have to remember
anything."], 'Virginia Woolf': ['If you do not tell the truth about yourself you
cannot tell it about other people.'], 'Ada Lovelace': ['The science of
operations, as derived from mathematics more especially, is a science of itself,
and has its own abstract truth and value.', 'The Analytical Engine has no
pretensions whatever to originate anything. It can do whatever we know how to
order it to perform.'], 'Ida B. Wells': ['The way to right wrongs is to turn the
light of truth upon them.'], 'Benjamin Disraeli': ['There are three types of
lies: lies, damn lies, and statistics.'], 'Ambrose Bierce': ['The hardest tumble
a man can make is to fall over his own bluff.'], 'Linus Torvalds': ['Talk is
cheap. Show me the code.'], 'Martin Fowler': ['Any fool can write code that a
computer can understand. Good programmers write code that humans can
understand.']}
>>> qc.getQuotes()['Mark Twain']
['A lie can travel half way around the world while the truth is putting on its
shoes.', "If you tell the truth, you don't have to remember anything."]
>>> qc.getQuotes()['Linus Torvalds']
['Talk is cheap. Show me the code.']
```

`CounterfeitError`

Implement an Exception class `CounterfeitError`. Sample usage:

```
>>> raise CounterfeitError() #doctest: +IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
...
CounterfeitError
>>> raise CounterfeitError('your message here!') #doctest:
+IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
...
CounterfeitError: your message here!
>>> isinstance(CounterfeitError(), Exception )
True
```

`Wallet`

Implement a class `Wallet` that stores a collection of bills. Details:

- The legal denominations are: 1, 5, 10, 20, 100. Any attempt to add a different denomination will raise a `CounterfeitError`
- The bills are always stored in descending order, i.e. 100s first, then 20s, ...
- composition *or* inheritance, your choice
- do *not* worry about efficiency

`__init__`

- can constructs empty Wallet
- optionally can supply a sequence of bills.

- raises a `CounterfeitError` if any bill is not a legal denomination

`__repr__` - see usage below

```
>>> w = Wallet()
>>> w
Wallet([])
>>> w = Wallet( [1,5,10,5,1,20,100,20,100] )
>>> w
Wallet([100, 100, 20, 20, 10, 5, 5, 1, 1])
>>> w = Wallet( [1,2,5] ) #doctest: +IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
...
CounterfeitError: 2 is a not a legal denomination!
```

`add`

- adds a single bill to the Wallet
- raises a `CounterfeitError` if any bill is not a legal denomination

```
>>> w = Wallet()
>>> w.add(5)
>>> w.add(10)
>>> w.add(1)
>>> w.add(10)
>>> w
Wallet([10, 10, 5, 1])
>>> w.add(4) #doctest: +IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
...
CounterfeitError: 4 is a not a legal denomination!
```

`addBills`

- adds a sequence of bills to the Wallet
- raises a `CounterfeitError` if any bill is not a legal denomination

```
>>> w = Wallet()
>>> w.addBills( [1,5,10,20,100,1])
>>> w
Wallet([100, 20, 10, 5, 1, 1])
>>> w.addBills( [1,20,2]) #doctest: +IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
...
CounterfeitError: 2 is a not a legal denomination!
```

`remove`

- removes a single bill of given denomination from the Wallet
- raises a `ValueError` if the bill is not there

```
>>> w = Wallet([10, 10, 5, 1])
>>> w.remove(5)
>>> w
Wallet([10, 10, 1])
>>> w.remove(5) #doctest: +IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
...
ValueError: list.remove(x): x not in list
```

`value`

- **returns** the total value of wall bills in the Wallet

```
>>> w = Wallet([100,100,20,10,10,5,1,1,1])
>>> w.value()
248
```

`removeAmount`

- attempts to remove the given amount from the Wallet
- removes bills to pay as much of given amount as possible
- **returns** any amount that was unable to be paid, or 0 if the whole amount was removed
- hint: the Wallet is sorted, if you remove the bills from highest to lowest, you will always remove the maximum amount
- if you can't quite get it, do your best and make sure you spend time on other problems

Example 1: the total amount can be made by bills removed from the Wallet, 0 is returned.

```
>>> w = Wallet([100,100,20,10,10,5,1,1,1])
>>> w.value()
248
>>> w.removeAmount(137)
0
>>> w
Wallet([100, 10, 1])
>>> w.value()
111
```

Example 2: there isn't enough money in the Wallet, all the bills are removed, and the the remaining amount 42 is returned.

```
>>> w = Wallet([100,20,10,10,10,5,1,1,1])
>>> w.value()
158
>>> w.removeAmount(200)
42
>>> w
Wallet([])
>>> w.value()
0
```

Example 3: there *is* enough money in the Wallet, but the bills can't make the exact amount, so the amount possible 18 is removed and the remaining 1 is returned.

```
>>> w = Wallet([100,10,5,5,1,1,1])
>>> w.value()
123
>>> w.removeAmount(19)
1
>>> w
Wallet([100, 5])
```

`fibExpr`

You should be familiar with the *Fibonacci sequence:* `1,1,2,3,5,8,13,21,34,...`. The first two numbers (index 0 and 1) in the sequence are 1 and 1 and every other number is the sum of the two previous numbers. Write a **recursive** function `fibExpr` that **returns** the **strings** indicated in the example below. In particular, `fibExpr(n)` is a string containing an expansion of the n^th^ Fibonacci number, each string can be constructed from the two previous strings. Hint: what is the pattern?

```
>>> fibExpr(0)
'1'
>>> fibExpr(1)
'1'
>>> fibExpr(2)
'(1+1)'
>>> fibExpr(3)
'((1+1)+1)'
>>> fibExpr(4)
'(((1+1)+1)+(1+1))'
>>> fibExpr(5)
'((((1+1)+1)+(1+1))+((1+1)+1))'
>>> fibExpr(6)
'(((((1+1)+1)+(1+1))+((1+1)+1))+(((1+1)+1)+(1+1)))'
>>> eval(fibExpr(6))
13
>>> [eval(fibExpr(n)) for n in range(6,11)]
[13, 21, 34, 55, 89]
```
