Final Programming Exam Question Bank

The purpose of the final programming exam is to test your ability to create code on your own using minimal help documentation. This may be different than how you have approached your homework problems and lab problems, so it is important that you practice before the exam.

Please practice for the final programming exam by using only IDLE and both pages of the two-page reference help sheet. (Previous programming problems, the Internet, and AI are not allowed to help to ensure you understand what you are doing.)

Some programming problems below will be easier to do than others. I will use similar-style problems on the final. Aim to spend 15 minutes on each problem on average.

Lists:

1. Write a function called **median(myList)** which takes in a list of numbers, sorts them, and returns the value which is the arithmetic median of that list (the "middle" value when the list is sorted into order). You can assume the list will be of an odd length. For example:

```
myList = [7, 4, 9, 2, 1]
median(myList)
4
```

Another example:

```
myList = [‘the’, ‘big’, ‘dog’, ‘ran’, ‘fast’]
median(myList)
‘fast’
```

2. Write a function called **removeItem(myList,item)** which takes in a list and an item and returns a new list with each occurrence of the item removed from the original list. For example:

```
removeItem([1, 2, "the", "test", 3, "is", "the", "best" ], "the")
[1, 2, "test", 3, "is", "best"]
```

3. Write a function called **addList(myList, addend)** which takes in a list of numbers and a single addend as parameters. The function returns a newly created list which is each number in the original list plus the addend. For example:

```
 addList([1,2,3.5,4.2] ,3)
[4, 5, 6.5, 7.2]
```

Files:

1. Write a function called **countLines(filename)** that takes in the name of a file to open. It then returns the number of lines in the file. For example:

Contents of file called file.txt
Hello, there are just
two lines in this file.

```
countLines('file.txt')
2
```

2. Write a function called **sensor(inFile, outFile, badWord)** that takes in the name of a input file, the name of a new output file to create, and a word. It should read the input file and create an identical output file except with all occurrences of badWord replaced by ####. (Note, you do not have to "clean" up words/punctuation.)

Contents of file called file.txt
```
mary
sold
seashells
by
the
sea
 from
her
sea
store
```

```
sensor('file.txt', 'out.txt', 'sea')
```

Contents of new created file out.txt
```
mary
sold
seashells
by
the
####
from
her
####
store
```

Sets:

1. Write a function called **commonLetters(name1,name2)** that takes in the names of two people. It then returns a list of the letters common to both names. Uppercase letters should get lowered. The returned list does not need to be sorted and can be in any order. For example:

```
commonLetters('Anna', 'Andy')
['a', 'n']
```

2. Write a function called **boxOfCrayons(colors1,colors2)** that takes in two lists of colors and returns the set of all the colors from the two lists. For example:

```
boxOfCrayons(['black','gold'], ['purple','gold'])
{'black', 'gold', 'purple'}
```

Dictionaries:

1. Write a function called **letterCounts(mystring)** which takes in a string and returns a dictionary of letter counts in that string. Uppercase letters should be lowercased. For example:

```
letterCounts('Bananas Rock!')
{'b':1, 'a':3, 'n':2, 's':1, 'r':1, 'o':1, 'c':1, 'k':1}
```

2. Write a function called **mostCommonLetter(myString)** which takes in a string and returns the most common letter in that string. You can assume there will be a most common letter without a tie. Uppercase letters should be lowercased. For example:

```
mostCommonLetter('Hello Mommy!')
m
```

Combinations (combining one or more of the above):

1. Write a function called mostCommonWord(myString) which takes in a string, lowercases the string, splits the string into words, and prints the most common word and how many times it occurs. For example:

```
mostCommonWord('The test is the best')
the 2
```

2. Write a function called uniqueWords(sentence) which takes in a string of all lowercase words with no punctuation separated by a single space. It should then return the number of unique words in the sentence.

```
uniqueWords('coffee coffee coffee coffee coffee in the morning')
4
```