GENERAL INSTRUCTIONS

This assignment consists of programming problems for which you must develop functions. We will be using Polya's approach to finding the algorithms that solve each problem. Rather than directly coding some solution to a problem – you must show the steps to your work. For each problem tagged with the words **"full development"** you must submit the following:

- A textual description of how you conceived your "plan" for solution. What steps did you take to understand the problem and come up with an algorithm. This section must be typed.
- Either pseudocode or a flowchart to illustrate your plan. If you are creating a flowchart, you may draw it out. However, it must be legible and neat. Illegible material will not be marked.

For all of the functions you must submit:

- A python function which is named and performs exactly as specified in the question and which will be submitted electronically in the file called a2.py as well as printed out and submitted with the paper copy of your assignment.
- An example of how you would test your coded solution (e.g., by running sample values through it) which will be submitted both electronically in the file a2Test.py and with the paper copy of your assignment. Testing is further discussed below.
- Be sure that you add **sufficient comments** to your functions and your tests to explain their purpose, the algorithm, the inputs and the outputs.
- Be sure to create your functions using the exact names specified in this assignment, and to pass the values in precisely the order they are shown. Failure to do this will cause the automarker (a program which will test your functions) to give you a zero for your function.

TESTING:

For this assignment we will test all of our functions in a separate python program `a2Test.py`. A sample function file `a2.py` and a sample function testing program file `a2Test.py` are available from the web. They clearly illustrate how to call a function from another file, and what a good test may be.

When you test, try to think up examples that will stress the limits of your algorithm. For example, try using quantities of zero (if permitted), one or many. If negative values are permitted, what happens when they are entered?

There is a more formal testing tool which we will use later in this course. For now, it is important to check that the output of a function is the value you expect. This can be accomplished simply by executing each function, and printing out the returned value.

SUBMITTING YOUR ASSIGNMENT:

Electronic submission:

Submit both a2.py and a2Test.py electronically using the submit command:

```
submit -c csc104h -a a2.py a2Test.py
```

Submit printout:

You must submit a paper copy of your assignment containing the full development steps of your algorithms (where required), the code for all of your functions, and the code for your test cases. Staple the pages together with a completed copy of the cover sheet (provided on the website) attached to the front. Then put your paper copy in the drop box for this course (csc104) in the Bahen building room BA2220.

CONCEPTS AND HINTS to help you through this assignment**:**

1. **Incremental development:**
   Question: How do you eat an elephant sandwich?
   Answer: One bite at a time
   As you develop each function, test it immediately. It is easier to attack problems related to a single function than many problems related to many functions. It also helps you to identify mistakes you are making more quickly, before you can repeat them in subsequent functions.
2. **Converting between integers and strings:**
   To convert a number to a text string: `str(number)`, e.g., `str(534)` would create the text string `'534'`.
   To convert a text string to an integer: `int(string)`, e.g., `a = '3542'`, `int(a)` would create the number `3542`.
3. **The modulus operator (%)**:
   Given any two numeric expressions `x` and `y` and the modulus operator `%`, the modulus operator, (also known as the remainder operator,) divides `x` by `y` and returns only the remainder.
   e.g. `5 % 2` is `1`
   `(2 + 5) % 4` is `3`
   `15 % (5 + 5)` is `5`
   `3 % 3` is `0`

4. **Using lists:**
   To define some list d: `d = ['sun', 'moon', 'stars', 'planets', 9]`
   To determine the number of items in the list d use the len() function: `count = len(d)`
   To access the items in the list, start at 0 and go to (`len(d) – 1`), so 'sun' is `d[0]` and 9 is `d[4]`
   To create a blank list: `blankList = list()`
   To append 'csc104' to blankList: `blanklist.append('csc104')`
   To switch the first and third items in d you must first create a temporary holding space for one of the items:

```
temp = d[0]
d[0] = d[2]
d[2] = temp
```

5. **Parallel lists:**
   Parallel lists are a pair of lists that are semantically related to each other. More specifically, given lists `a` and `b`, `a[0]` is related to `b[0]`, `a[1]` is related to `b[1]`, `a[2]` is related to `b[2]`, … `a[n-1]` is related to `b[n-1]`.

   In this exercise, we will use parallel lists to link monetary denominations to how many of each denomination is required. Suppose we have a list called denominations, where `denominations = [200, 100, 50, 25, 10, 5, 1]` which represents denominations from a Canadian toony (200) to a penny (1). We could create a parallel list and call it, say, coinCount, which would represent a quantity of each denomination. If we want to represent a toony, two loonies, and four nickels, (and zero of all the others), `coinCount` would equal `[1, 2, 0, 0, 0, 4, 0].` Where coinCount[0] is the number of denomination[0] coins, coinCount[1] is the quantity of denomination[1] coins, etc. When using parallel

lists, they must always be the same length to ensure associated values are present, even if they have a value of zero.

Though we relate our parallel lists to show values in one list and associated quantities in another, they represent a neat and simple trick with many useful applications. (e.g., to represent board moves, in the x and y direction, in many game programs.)

THE PROBLEMS TO BE SOLVED:

MONETARY SYSTEMS

You are about to go on a trip around the world. You are concerned that you will overpay or underpay for items that you pick up on your trip. You really don't want to have to figure out the monetary system of each country you go to – so you write a few python functions to help you out.

Function	**Specifications**
`getChange(cost,amtPayed)`	Accept a `cost` amount, the amount of money payed (`amtPayed`) and returnsa single value which represents the amount of change owed.

`pay(value,denom)`	**Full Development.**
	Determine how many of each denomination is required to pay for an item of a specified value. Accept an integer cost `value`, and a list of the monetary denominations in descending order of a country (`denom`). It returns a (parallel) list containing the quantities of each denomination required to pay for an item which costs `value` amount. The fewest number of coins and bills must be used.
	
	Note: The correct answer to this question must contain the fewest number of coins or dollars. For example, if we had to pay for an item which cost $20.45, we would pay with a twenty dollar bill, a quarter, and two dimes. Anything else would increase the quantity of coins and dollars given.
	
	Example:
	`d = [200, 100, 50, 25, 10, 5, 1]`
	`val = 998`
	`amounts = pay(val, d)`
	
	pay would calculate that you would need the following quantities of each denomination:
	4 of 200
	1 of 100
	1 of 50
	1 of 25
	2 of 10
	0 of 5
	3 of 1
	
	the list `amounts` would contain [4, 1, 1, 1, 2, 0, 3]

Function	Specifications
`calcValue(denom,quantities)`	**Full Development.** Accepts parallel lists containing the denominations of a monetary system (`denom`), and the quantity of each denomination that is present (`quantities`). Returns the (single) monetary value represented by the quantities of the denominations.
`getChangeDenominations(cost, amtPayed, denom)`	Accepts a `cost` amount, the amount of money tendered (`amtPayed`), the list of denominations (`denom`) for the monetary system (in descending order), calculates the amount to be returned for each denomination, and returns it as a parallel list. **Note**: the correct answer returns the fewest number of coins and bills.
`amtOwed(costList)`	Accepts a list containing the cost of several items (`costList`). Returns a value which is the total cost for all the items.