6.S189 Homework 3

`http://web.mit.edu/6.s189/www/materials.html`

What to turn in

Checkoffs 6 and 7 are due at **5 PM** on **Thursday, January 17th**. Checkoff 6 is over Exercises 6.1 - 6.3 and Checkoff 7 is over Exercises 7.1 - 7.2.

Exercise 6.1 – Finding Bugs

The following set of instructions were given to Ben Bitdiddle, and he produced the code below. Find at least 3 bugs he made, and say how to fix them. Feel free to write your answers in either the `homework_3.py` template or on this piece of paper.

Instructions: Write a `negate` function that takes a number and returns the negation of that number. Also write a `large_num` function that takes a number, and returns True if that number is bigger than 10000, and False otherwise. Additionally, write some code to test your functions.

```
def negate(num):
    return -num

def large_num(num):
    res = (num > 10000)

negate(b)
neg_b = num
print 'b:', b, 'neg_b:', neg_b

big = large_num(b)
print 'b is big:', big
```

Bugs:

1.

2.

3.

Exercise 6.2 – Mutability

We've learned about many Python data structures (strings, lists, tuples, dictionaries). For both "mutable" and "immutable", please give a short (5 words or fewer) definition, and then list what data structure(s) have that characteristic.

Feel free to write your answers in either the `homework_3.py` template or on this piece of paper.

Mutable:

Immutable:

Exercise 6.3 – Collision Detection of Balls

Many games have complex physics engines, and one major function of these engines is to figure out if two objects are colliding. Weirdly-shaped objects are often approximated as balls. In this problem, we will figure out if two balls are colliding. You'll need to remember how to *unpack tuples*; refer to Chapter 9.2 or ask an LA if this is confusing.

We will think in 2D to simplify things, though 3D isn't different conceptually. For calculating collision, we only care about a ball's position in space, as well as its size. We can store a ball's position with the (x, y) coordinates of its center point, and we can calculate its size if we know its radius. Thus, we represent a ball in 2D space as a tuple of `(x, y, r)`.

To figure out if two balls are colliding, we need to compute the distance between their centers. If this distance is less than or equal to the sum of their radii, they are colliding.

In `homework_3.py`, write a function `ball_collide` that takes two balls as parameters and computes if they are colliding; your function should return a Boolean representing whether or not the balls are colliding. **Optional:** For a little extra challenge, write your function to work with balls in 3D space. How should you represent the balls? You will also need to write your own test cases - be sure to figure out any edge cases you need to test.

Exercise 7.1 – An Introduction to Dictionaries

Quick Reference `D = {}` creates an empty dictionary
`D = {key1:value1, ...}` creates a non-empty dictionary
`D[key]` returns the value thats mapped to by key. (What if there's no such key?)
`D[key] = newvalue` maps `newvalue` to `key`. Overwrites any previous value. Remember - `newvalue` can be *any* valid Python data structure.
`del D[key]` deletes the mapping with that key from `D`.
`len(D)` returns the number of entries (mappings/key-value pairs) in `D`.
`x in D, x not in D` checks whether the key `x` is in the dictionary `D`.
`D.keys()` - returns a list of all the keys in the dictionary.
`D.values()` - returns a list of all the values in the dictionary.

In `homework_3.py`, write a dictionary that catalogs the classes you took last term - the keys should be the class number, and the values should be the title of the class.

Then, write a function `add_class` that takes 3 arguments - a class number, class description, and the dictionary - that adds new classes to your dictionary. Use this function to add the classes you're taking next term to the dictionary.

Finally, write a function `print_classes` that takes two arguments - a Course number (e.g. '6') and the dictionary you've built - and nicely prints out all the classes you took in that Course. Make sure your code works even if you didn't take any classes in that course!

Example output:

```
>> print_classes('6', myClassDict)
6.s189 - Introduction to Python
6.01 - Introduction to EECS
>> print_classes('9', myClassDict)
No Course 9 classes taken
```

For this exercise, we suggest using strings everywhere (rather than ints or floats). Be sure to test with Course numbers that you both did and did not take!

Exercise 7.2 – Address Book

For this problem, we provide you with a function, `buildAddrBook(fileName)`, that takes one parameter (the name of a csv file containing names, addresses and contact information), and builds and returns an address book. The address book is a dictionary that has keys that are strings of the form `'LastName, FirstName'`, and values that are lists that contain a phone number and email address.

`buildAddrBook(filename)` can be used as follows:

```
>>> addrBook = buildAddrBook('rawAddresses.csv')
>>> print addrBook
{'Lemon, Liz': ['(202) 555-8130', 'lizlemon@nbc.com'],
 'Manchu, Foo': ['(480) 555-6134', 'foomanchu@gmail.com']}
```

Note that the rawAddresses.csv file we provide you will result in a different result than is shown above. We have shortened it here for demonstration purposes.

Your job is to fill in the definition for the function `changeEntry(addrBook, entry, field, newValue)`. This function will allow you to change a pre-existing entry (but not add a new one). A description of the parameters:

`addrBook` : A dictionary that is in the address book format as generated by the function `buildAddrBook`.

`entry` : The entry to be changed (in our example above, 'Lemon, Liz' and 'Manchu, Foo' are the only valid entries).

`field` : The field to change. The only valid fields are: `'name'` (changes an entry's name), `'phoneNumber'` (changes an entry's phone number), and `'emailAddress'` (**adds** a new email address to the specified entry).

`newValue` : The new value for the indicated field.

If a user enters in an entry that's not already in the address book, or enters in an invalid field, an appropriate error message should be printed.

To continue with the earlier example, here's an interaction in the IDLE shell that shows us modifying our address book with the `changeEntry` function:

```
>>> changeEntry(addrBook, 'Lemon, Liz', 'emailAddress', 'lizzing@starwars.net')
>>> print addrBook
{'Lemon, Liz': ['(202) 555-8130', 'lizlemon@nbc.com', 'lizzing@starwars.net'],
'Manchu, Foo': ['(480) 555-6134', 'foomanchu@gmail.com']}

>>> changeEntry(addrBook, 'Lemon, Liz', 'phoneNumber', '1-900-OKFACE')
>>> print addrBook
{'Lemon, Liz': ['1-900-OKFACE', 'lizlemon@nbc.com', 'lizzing@starwars.net'],
'Manchu, Foo': ['(480) 555-6134', 'foomanchu@gmail.com']}

>>> changeEntry(addrBook, 'Lemon, Liz', 'name', 'Lemon, Elizabeth')
>>> print addrBook
{'Lemon, Elizabeth': ['1-900-OKFACE', 'lizlemon@nbc.com', 'lizzing@starwars.net'],
'Manchu, Foo': ['(480) 555-6134', 'foomanchu@gmail.com']}
>>> print addrBook['Lemon, Liz']

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print addrBook['Lemon, Liz']
KeyError: 'Lemon, Liz'

>>> changeEntry(addrBook, 'Bitdiddle, Ben', 'emailAddress', 'bitdiddly@bu.edu')
Invalid entry: Bitdiddle, Ben

>>> changeEntry(addrBook, 'Manchu, Foo', 'homeAddress', 'Baltimore, MD')
Unexpected field: homeAddress
```

You can uses these interactions (or similar interactions) to test your code.

A few notes: Note that when we change 'Lemon, Liz' to 'Lemon, Elizabeth', there is no longer an entry for 'Lemon, Liz'. Also note that no exceptions are thrown (red, angry error on the terminal) when we pass in an entry that isn't in the address book (such as 'Bitdiddle, Ben') or an invalid field (such as 'homeAddress') - an error message is instead printed.

Once you've done this, you're done!

Optional Extension: In your `changeEntry` function, allow an additional field, `'newEntry'`, that allows you to add a new entry to the address book. Hint: in order to implement this, think about what the values of `entry` and `newValue` will need to be. Be sure that your error messages from before still work (unless `field = 'newEntry'` has been specified)!