Last Name: _____ First: _____ Netid: _____ Section _____

CS 1110 Prelim 2 November 12th, 2015

This 90-minute exam has 6 questions worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

It is a violation of the Academic Integrity Code to look at any exam other than your own, look at any reference material, or otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
    if something:
        do something
        do more things
    do something last
```

You should not use while-loops on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, for-loops, recursion and so on) unless otherwise prohibited.

Question	Points	Score
1	2	
2	22	
3	16	
4	18	
5	18	
6	24	
Total:	100	

The Important First Question:

1. [2 points] Write your last name, first name, netid, and *lab section* at the top of each page. If you cannot remember the section number, please write down your lab's time and place.

Reference Page

Throughout this exam you will make use of strings, lists, and dictionaries. You are expected to understand how slicing works. In addition, the following functions and methods may be useful:

String Expressions and Methods

Expression or Method	Description
`len(s)`	**Returns**: number of characters in `s`; it can be 0.
`a in s`	**Returns**: True if the substring `a` is in `s`; False otherwise.
`s.find(s1)`	**Returns**: index of the first character of the FIRST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).
`s.find(s1,n)`	**Returns**: index of the first character of the first occurrence of `s1` in `s` STARTING at position n. (-1 if `s1` does not occur in s from this position).
`s.rfind(s1)`	**Returns**: index of the first character of the LAST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).
`s.count(s1)`	**Returns**: number of (non-overlapping) occurrences of substring `s1` in `s`.
`s.replace(a,b)`	**Returns**: A *copy* of `s` where all instances of substring `a` are replaced with the substring `b`.

List Expressions and Methods

Expression or Method	Description
`len(x)`	**Returns**: number of elements in list `x`; it can be 0.
`y in x`	**Returns**: True if `y` is in list `x`; False otherwise.
`x.count(y)`	**Returns**: number of times `y` appears in the list `x`.
`x.index(y)`	**Returns**: index of the FIRST occurrence of `y` in `x` (an error occurs if `y` does not occur in `x`).
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in list `x`, shifting later elements to the right.
`x.remove(y)`	Removes the first item from the list whose value is `y`. (an error occurs if `y` does not occur in `x`).

Dictionary Expressions and Methods

Expression or Method	Description
`len(d)`	**Returns**: number of keys in dictionary `d`; it can be 0.
`y in d`	**Returns**: True if `y` is a key `d`; False otherwise.
`d[k]`	**Returns**: The value in `d` for key `k` (Raises an error if `k` is not a key in `d`)
`d.keys()`	**Returns**: a list containing all the keys in `d`.
`d.values()`	**Returns**: a list containing all the values in `d`. It may have duplicates.

2. [22 points] **Classes and Subclasses**

Another programmer has written the class `BankAccount` for you. The only things that you know about `BankAccount` are the documentation and methods show below. In particular, you do not know what the other programmer named the attributes (they are hidden), and you only know the names of the getters and setters. **Do not implement this class**.

You want create a subclass `SavingsAccount` that adds functionality to to `BankAccount`. For example, savings accounts earn interest. Using what little you know about `BankAccount`, complete this class on the next page. In particular,

1. Fill in the missing information in the class.
2. Add getters and setters as approporiate for the instance attributes
3. Fill in the parameters of each method (beyond the getters and setters).
4. Implement each method according to its specification.
5. Enforce any preconditions in these methods using asserts

If there is not enough space to implement a method, write on the back of the page and indicate this in your method.

```
class BankAccount(object):
    """Instances represent a bank account.

    ATTRIBUTES (These may not be the actual names)
        account [int > 0]: The identifying account number
        balance [float >= 0]: The amount of money in account"""

    def getAccount(self):
        """Returns: The identifying account number."""

    def getBalance(self):
        """Returns: The current balance for this account."""

    def deposit(self,value):
        """Deposits value into account, adding it to the balance.

        Precondition: value is an int or float >= 0"""

    def withdraw(self,value):
        """Withdraws value from account, subtracting it from the balance.

        Precondition: value is an int or float <= current balance and >= 0"""

    def __init__(self,b):
        """Initializer: Creates an account with starting balance b.

        The account number is assigned automatically, and hence not provided
        as a parameter.

        Parameter b: The starting balance
        Precondition: b is a number (int or float) >= 0"""

    def __str__(self):
        """Returns: A string representation of this account.

        The account balance should be formatted to two decimal places.

        Example: '[Account 89972:  $12.43]'  """
```

Last Name: _____ First: _____ Netid: _____ Section _____

```
class SavingsAccount(        BankAccount        ):        # Fill in missing part
    """Instances represent an interest-earning bank account

    MUTABLE ATTRIBUTE
        interest [float >= 0]: The percentage interest earned each month

    IMMUTABLE ATTRIBUTE
        type [nonempty string]: Description of the account type"""

    # PUT GETTERS/SETTERS HERE AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.

    def getInterest(self):
        return self._interest



    def getType(self):
        return self._type



    def getInterest(self,value):
        assert type(value) == float and value >= 0
        self._interest = value



    # Not a true setter, but does modify an attribute

    def addInterest(             self             ):          # Fill in parameters
        """Applies the interest to the balance.
        Multiplies balance by the interest rate, and then deposits the amount into
        the account.
        HINT: Interest is a PERCENTAGE."""
        v = self.getBalance()*self.getInterest()/100.0
        self.deposit(v)
```

```
# Initializer and built-in methods

def __init__(    self,   b,   i,   t = 'Standard'    ):  # Fill in parameters

    """Initializer: Creates a new savings account with the given attributes

    Param b: The tarting balance.
    Precond: b is an int or float >= 0

    Param i: Percentage interest earned each month.
    Precond: i is a float >= 0

    Param t: Account type (OPTIONAL: default is 'Standard').
    Precond: t is a nonempty string"""

    BankAccount.__init__(self,b)
    assert type(t) == str and len(t) > 0
    self._type = t
    self.setInterest(i)   # Enforces precondition
```

```
def __str__(              self              ):          # Fill in parameters
    """Returns: A string representation of this account

    The account balance should be formatted to two decimal places.  There are no
    restrictions on the number of decimal places for the interest (so no special
    formatting is required).

    Example: '[Account 89972:  $12.43 at 4.357%, Standard]'

    HINT: Formatting the amount of money is hard.  The parent BankAccount string
    already has properly formatted the balance.  Why not use that?"""

    s = BankAccount.__str__(self)[:-1]
    s = s + ' at '+str(self._interest)+'%'
    return s + ', '+self._type+']'
```

3. [16 points total] **Short Answer**.
Answer the following questions. Each answer will require multiple sentences, but should not require more than a paragraph.

(a) [4 points] What is the difference between `is` and `==`? Give an example of when you would want to use each of these operators.

The operator `is` compares objects by "folder name."

The operator `==` invokes the `__eq__` method. This generally, but not always, compares objects by contents.

In general, you want to use `==` if you want to compare objects by contents (e.g. check if two distinct Vector objects have different x and y attributes). You always need to use `is` to compare an object to `None`, as `None` has no contents.

(b) [4 points] What is the *bottom-up* rule? How does it relate to *overriding*?

The bottom-up rule says that, when we call a method or access attribute, Python starts from the object folder. If it cannot find it there, it then checks the class folder. Finally, it searches the parent classes in order, all the way to the folder for the object class.

Overridding is the act of the replacing a method that is defined in a parent class with a new definition. The bottom-up rule guarantees the new definition will always be used.

(c) [4 points] Initialize a dictionary `d` that makes the following expression True.

```
>>> d['A']+d['B'] == 10
True
```

There are *many* acceptable solutions to this problem. Here is a simple one.

```
>>> d = { 'A':5, 'B':5 }
```

(d) [4 points] Consider the function with the following specification.

```
def foo(x):
    """Returns:  The number x+1

    Preconditon: x is a number (int or float)"""
```

Write code to enforce the precondition for this function. However, instead of creating an AssertionError, the code should create a TypeError when the precondition is violated (e.g. do not use an assert statement). Error messages are optional.

We need to raise a TypeError, when the precondition is violated. We do this as follows:

```
if type(x) != int and type(x) != float:
    raise TypeError(`x`+' is not a number')
```

4. [18 points total] **Iteration**.

   The functions below all take a list as one their arguments. You should use for-loops to implement these functions. **You are not allowed to use the method** `index`. In addition, the second function (`remove_dups`) may not use the first function (`find`) as a helper.

   You **do not need to enforce** the function preconditions.

   (a) [9 points]

```
def find(seq,v):
    """Returns: Position of FIRST occurence of v in seq, or -1 if not there.
    Precondition:  seq is a list, v is any value"""
    # Accumulator is a int, defaulting at -1
    pos = -1

    # Must loop over positions
    for k in range(len(seq)):
        # Need first part if no return in loop
        if pos == -1 and seq[k] == v:
            pos = k                    # Putting a return here is okay

    return pos
```

   (b) [9 points]

```
def remove_dups(seq):
    """Returns: A copy of seq with all duplicates are removed.
    For each value with duplicates, only the first occurrence remains.
    Example: remove_dups([1,1,2,1,3,2]) returns [1,2,3]
    Precondition: seq is a list"""
    # Accumulator for the copy
    copy = []

    # Okay to loop over values
    for x in seq:
        # Only add if not a duplicate
        if not x in copy:
            copy.append(x)

    return copy
```

5. [18 points total] **Recursion**.

   The made-up language Baablee takes any word and doubles all of the vowels. For example, `'bite'` becomes `'biitee'`, `'coat'` becomes `'cooaat'`, and `'beef'` becomes `'beeeef'`. For the purpose of Baablee, the only vowels are `'a'`, `'e'`, `'i'`, `'o'`, and `'u'`. `'y'` is not a vowel.

   Implement the functions below which convert a string to its Baableed version and back. You cannot use for-loops; you must use recursion. Otherwise, you are allowed to use any other functions or methods that you wish. You do not need to enforce preconditions.

   (a) [9 points]

```
def encode(s):
    """Returns: A Baableed copy of s

    Example: encode('axe') is 'aaxee', encode('beef') is 'beeeef'

    Precondition: s is a string of only lower-case letters"""

    # Base case if string empty
    if len(s) == 0:
        return s

    # First letter is a vowel; duplicate it
    if s[0] in 'aeiou':
        return s[0]+s[0]+encode(s[1:])

    # First letter is consonant; no change
    return s[0]+encode(s[1:])
```

   (b) [9 points]

```
def decode(s):
    """Returns:  The a copy of s, restored to original form

    Example: decode('aaxee') is 'axe', decode('beeeef') is 'beef'

    Precondition:  s is a Baableed string of only lower case letters"""

    # Base case if string empty
    if len(s) == 0:
        return s

    # First letter is a vowel; delete the next vowel
    if s[0] in 'aeiou':
        return s[0]+decode(s[2:])    # Note slice starts at 2

    # First letter is consonant; no change
    return s[0]+decode(s[1:])
```

Last Name: _____ First: _____ Netid: _____ Section _____

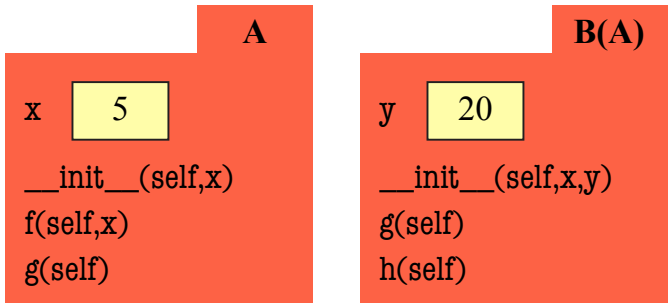
6. [24 points total] **Folders and Name Resolution**

Consider the two (undocumented) classes below.

```
class A(object):
    x = 5
    def __init__(self,x):
        self.y = x                  1

    def f(self,x):
        self.x = x                  1

    def g(self):
        return self.x+self.y        1
```

```
class B(A):
    y = 20
    def __init__(self,x,y):
        self.y = 42                 1
        A.__init__(self,x)          2
        self.z = y                  3

    def g(self):
        return self.y*self.x        1

    def h(self):
        self.f(self.z)              1
        return self.g()             2
```

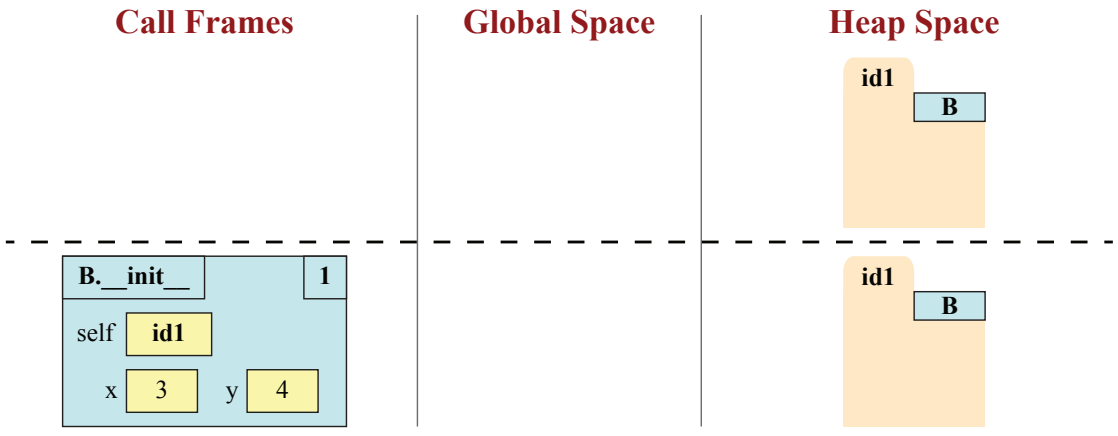
(a) [6 points] Draw the class folders for the classes A and B. Follow the conventions used for Assignment 5 (particularly with regards to methods).



(b) [18 points] On this page and the next, diagram the evolution of the assignment statement

```
>>> b = B(3,4)
```

You will need **eight diagrams**. In each diagram, draw the call stack, global space and heap space. If the contents of any space are unchanged between diagrams, you may write *unchanged*. You do **not** need to draw the class folders in heap space.



Last Name: _____ First: _____ Netid: _____ Section _____