Last Name: ______________ First: ______________ Netid: ______________

CS 1110 Prelim 2 November 9th, 2017

This 90-minute exam has 5 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).

Question	Points	Score
1	2	
2	22	
3	22	
4	26	
5	28	
Total:	100	

Last Name: ____________________ First: ____________________ Netid: ____________________

References

String Functions and Methods

Function/Method	Description
`len(s)`	**Returns**: Number of characters in `s`; it can be 0.
`s.find(s1)`	**Returns**: Index of FIRST occurrence of `s1` in `s` (-1 if `s1` is not in s).
`s.count(s1)`	**Returns**: Number of (non-overlapping) occurrences of `s1` in `s`.
`s.replace(a,b)`	**Returns**: A *copy* of `s` where all instances of `a` are replaced with `b`.
`s.upper()`	**Returns**: A copy of `s`, all letters converted to upper case.
`s.lower()`	**Returns**: A copy of `s`, all letters converted to lower case.
`s.isalpha()`	**Returns**: True if `s` is *not empty* and its elements are all letters; it returns False otherwise.
`s.isdigit()`	**Returns**: True if `s` is *not empty* and its elements are all numbers; it returns False otherwise.

List Functions and Methods

Function/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.index(y)`	**Returns**: Index of FIRST occurrence of `y` in `x` (error if `y` is not in `x`).
`x.count(y)`	**Returns**: the number of times `y` appears in list `x`.
`x.pop()`	**Returns**: The first element of `x`, removing it from the list.
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in `x`. Elements after `i` are shifted to the right.
`x.remove(y)`	Removes first item from the list equal to `y`. (error if `y` is not in `x`).
`x.sort()`	Rearranges the elements of `x` to be in ascending order.

Other Functions

Function	Description
`range(a)`	**Returns**: An iterable for processing elements (0, 1, . . ., `a`-1).
`range(a,b)`	**Returns**: An iterable for processing elements (`a`, `a`+1, . . ., `b`-1).
`range(a,b,n)`	**Returns**: An iterable for elements (`a`, `a`+`n`, . . ., `a`+`n*((b-a-1)//n)`).
`isinstance(o,c)`	**Returns**: True if `o` is an instance of the class `c`; False otherwise.

The Important First Question:

1. [2 points] Write your last name, first name, and netid at the top of each page.

Last Name: ______________________ First: ______________________ Netid: ______________________

2. [22 points] **Iteration**.

   The following functions take lists as their input. Use for-loops to implement them according to their specification. You **do not** need to enforce the preconditions.

```
def  pairswap(nlist):
    """Swaps adjacent pairs in nlist.

    This is a procedure that modifies the list.  It swaps the element at each even
    position (starting with 0) with its later element.  If the list has an odd number
    of positions, the last element is not affected.

    Example: pairswap([1, 3, 5, 7]) alters the list so that it is [3, 1, 7, 5]
    Example: pairswap([1, 3, 5, 7, 9]) alters the list so that it is [3, 1, 7, 5, 9]

    Parameter nlist: The list to swap
    Precondition: nlist is a (possibly empty) list of numbers"""
```

```
def colavg(table):
    """Returns: A list of the averages of each column.

    Example: colavg([[1.0,2.0], [4.0,6.0]]) is [2.5,4.0]
    Example: colavg([[1.0,2.0], [3.0,4.0], [5.0,9.0]]) is [3.0,5.0]

    Parameter table: The table to average
    Precondition: table is a (non-empty) rectangular 2d list of floats."""
```

Last Name: ______________ First: ______________ Netid: ______________

3. [22 points] **Recursion**.

   Use recursion to implement the following functions according to their specification. Solutions using for-loops will receive no credit. You **do not** need to enforce the preconditions.

```
def filter(nlist):
    """Returns: A copy of nlist with all negative numbers removed, preserving order

    Example: filter([1, -1, 2, -3, -4, 0]) is [1, 2, 0]

    Precondition: nlist is a (possibly empty) list of numbers"""
```

```
def segregate(nlist):
    """Returns: A tuple segregating nlist into negative and non-negative.

    This function returns a tuple (pos,rlist). The value rlist is a reordered copy
    of nlist where negatives come before the non-negatives. However, ordering inside
    each part (negative, non-negatives) should remain EXACTLY as it is in nlist.
    The value pos indicates the first position of a non-negative number in rlist.
    If there are no non-negative numbers, pos is -1.

    Examples: segregate([1, -1, 2, -5, -3, 0]) returns (3, [-1, -5, -3, 1, 2, 0])
              segregate([-1, -5, -3]) returns (-1, [-1, -5, -3])

    Precondition: nlist is a (possibly empty) list of numbers"""
```

4. [26 points total] **Folders and Name Resolution**

Consider the three (undocumented) classes below, together with their line numbers.

```
 1  class A(object):
 2      x = 5
 3
 4      def __init__(self,x):
 5          self.x = x+3
 6
 7      def foo(self,x):
 8          self.y = self.x
 9          self.z = x
10
11  class B(A):
12      x = 10
13
14      def __init__(self,x):
15          self.foo(x-1)
16
17      def bar(self,y):
18          self.y = 4*x
19          self.z = self.x
```

```
20  class C(B):
21      y = 15
22
23      def __init__(self,x):
24          super().__init__(x+1)
25
26      def foo(self,y):
27          self.y = 2*self.x
28          self.z = 3*y
```

(a) [6 points] Draw the class folders in heap space for these three classes.

(b) [20 points] On the next two pages, diagram the call

```
>>> c = C(3)
```

You will need **nine diagrams**. 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 from part (a).

When diagramming a constructor, you should follow the rules from Assignment 5. Remember that `__init__` is a helper to a constructor but it is not the same as the constructor. In particular, there is an important first step before you create the call frame.

Last Name: ______________ First: ______________ Netid: ______________

Call Frames	Global Space	Heap Space
①		
②		
③		
④		
⑤		

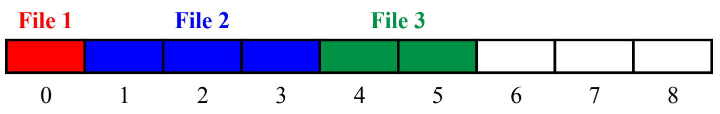
Last Name: ____________ First: ____________ Netid: ____________

Call Frames	Global Space	Heap Space
⑥		
⑦		
⑧		
⑨		

Last Name: ______________ First: ______________ Netid: ______________

5. [28 points] **Classes and Subclasses**

One of the main roles of the operating system is to manage files on your computer. The file system has a list of memory to store files. The unit of measurement used by the filesystem is a *block*, which is often 4 kilobytes. When the user saves a file, the filesystem finds the next available block in the list (usually at the end) and assigns that many blocks to the file. For example, the illustration below shows a file system where the first three files are one block, three blocks, and two blocks in size, respectively.



On the next page, you are going to model this process with the class `File`. Read the specification carefully. You will notice that this class has a class attribute. This attribute keeps track of the next available block in the filesystem. You will use this in the initializer to assign a starting position to the file.

One the page after the class `File`, you will see the class `Directory`. Recall that a directory is a folder that contains files. However, it has a lot of features in common with files, such as a name and location. Hence it makes sense for `Directory` to be a subclass of `File`.

On the next three pages, you are to do the following:

1. Fill in the missing information in each class header.
2. Add getters and setters as appropriate 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

To do the last part (enforce preconditions), you **may find it useful to add helper methods, but this is not required**. We have provided space for you to add these helpers, but you are free to leave this space blank. However, we do ask that all helper methods be *hidden* if you include them.

We have not added headers for any of the getters and setters. Like the helper methods, you are expected to write these from scratch. However, **you are not expected to write specifications for any of these methods: getters, setter, or helpers**.

Important: `Directory` is not allowed to access any hidden attributes or attributes of `File`. We are also adding the additional restriction that `Directory` may not access any getters and setters in `File`.

Last Name: ______________________ First: ______________________ Netid: ______________________

```
class File(                              ):            # Fill in missing part
    """A class representing a file on a computer

    CLASS ATTRIBUTES (NO GETTERS/SETTERS):
        BLOCK:  The next available file block [int >= 0]
    The initial block position is 0.

    MUTABLE ATTRIBUTES:
        _name: The name of the file [nonempty string not containing '/' or space]

    IMMUTABLE ATTRIBUTES:
        _start: The starting block of the file    [int >= 0]
        _size:  The number of blocks in the file   [int > 0]"""
    # CLASS ATTRIBUTES.




    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    # Class File (CONTINUED).
    def __init__(                                   ):      # Fill in parameters
       """Initializer: Creates a file the given name and size.

       The start of the file is the next available file block. The block position is
       incremented by the file size (which is an optional parameter)

       Parameter name: The file name
       Precondition: name is a nonempty string not containing '/' or space
       Parameter size: The number of blocks in the file (Optional; default 1)
       Precondition: size is an int > 0"""









    def __str__(                                   ):      # Fill in parameters
       """Returns: A string representation of this file.

       Format is 'filename (Blocks start to end)', or 'filename (Block start)'
       for files of size 1 block.
       Example: 'test.txt (Block 6)' or 'image.png (Blocks 2 to 5)' """










 class Directory(                              ):            # Fill in missing part
    """A class representing a folder containing other files

    IMMUTABLE ATTIBUTE (In addition to those from File):
        _contents : The files in this directory [list of File, may be empty]"""
    # DEFINE GETTERS/SETTERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
```

Last Name: ______________ First: ______________ Netid: ______________

```
# Class Directory (CONTINUED).
# DEFINE HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.








def __init__(                                  ):      # Fill in parameters
    """Initializer: Creates an directory with the given name and file contents

    Directories all have a size of 1 block.

    Parameter name: The directory name
    Precondition: name is a nonempty string not containing '/' or space
    Parameter contents: The files in this directory
    Precondition: contents is a (possibly empty) list of File"""






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

    The format is 'Directory name (Block start, n file/files)'
    Examples: 'Directory empty (Block 12, 0 files)'
              'Directory stuff (Block 8, 2 files)'
              'Directory loner (Block 10, 1 file)'

    IMPORTANT: You may not access any attributes or getters from class File."""
```