Last Name: _____ First Name: _____ Cornell Netid: _____

CS 1110 Prelim 1 October 4th, 2012

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, to look at any other reference material, or to 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 may not use for-loops or recursion on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, and so on).

Question	Points	Score
1	2	
2	18	
3	22	
4	18	
5	20	
6	20	
Total:	100	

The Important First Question:

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

Last Name: _____ First Name: _____ Cornell Netid: _____

2. [18 points total] **Objects and Functions**.
For this question, you are given a class called `Time` provided by the module `time`. Objects of type `Time` are **mutable objects** with two attributes: `minutes`, and `hours`. Like the objects in Assignment 3, these attributes have invariants:

Attribute	Invariant
`minutes`	`int` value in range `0..59`
`hours`	`int` value ≥ 0

The constructor `Time(h,m)` takes two arguments: the hours and minutes, in that order.

Define the following functions using the type `Time`. Follow the specifications provided. You do not need to check preconditions with assert statements; they just tell you what to expect.

(a) [8 points]

```
def before(time1, time2):
    """Returns:  True if time1 is earlier than time2
    Example:  1 hr 59 min is before 2 hrs 0 min
    Precondition:  time1 and time2 are Time objects"""
    # A longer version using if-statements is also okay.
    return ( time1.hours < time2.hours or
            (time1.hours == time2.hours and
             time1.minutes < time2.minutes ) )
```

(b) [10 points]

```
def sum(time1, time2):
    """Returns:  The sum of time1 and time2 as a new Time object
    Example:  Sum of 1 hr 59 min and 1 hr 2 min is 3 hr 1 min
    DO NOT ALTER time1 or time2, even though they are mutable
    Precondition:  time1 and time2 are Time objects"""
    # Sum the attributes
    m = time1.minutes+time2.minutes
    h = time1.hours+time2.hours

    # Make sure invariant satisfied
    h = h + m/60
    m = m % 60

    # Create new time object
    return Time(h,m)
```

3. [22 points total] **Testing and Program Flow**.

(a) [8 points] Unlike many other languages, sequences in Python can contain a mixture of datatypes. For example, the following is legal in Python

```
s = [1, True, 'Hello', 3.0]
```

This list contains an `int`, a `bool`, a `string` and a `float`.

For reasons of program safety, we often prefer that the elements of our lists all have the same type. That is why we might have a function like the following:

```
def extract_int(seq):
    """Return:  a sequence containing only the ints in seq.
    Function takes the list seq and returns a new list that contains
    only those elements of seq that are ints, in the same order
    they occur in seq.
    Example: extract_int([1,True,2]) returns [1, 2]
    Precondition: seq is a list"""
```

Do not implement this function. Instead, write down a list of at least **four test cases** that you would use to test out this function. By a test case, we just mean an input and an expected output; you do not need to write an `assert_equals` statement. For each test case *explain why it is significantly different from the others.*

There are many, many answers to this question. Here are but just a few.

Input: `[]`, **Output**: `[]`
The empty list is always an important test case.

Input: `[1,'Hello',True]`, **Output**: `[1]`
A list with just one int (and a mixture of non-ints).

Input: `[1,'Hello',2,True,3]`, **Output**: `[1,2,3]`
A list with more than one int (and a mixture of non-ints).

Input: `[1,2,3]`, **Output**: `[1,2,3]`
A list with all ints.

Input: `[True,False]`, **Output**: `[]`
A list with no ints.

(b) [8 points] Below are three function definitions using asserts and try-except. Write out the series of print statements displayed for each of the given function calls (on next page).

```
def first(n):
    print 'Start first'
    try:
        second(n)
        print 'In first try'
    except:
        print 'In first except'
    print 'Done first'
```

```
def second(n):
    print 'Start second'
    try:
        assert n <= 0, 'is not <= 0'
        print 'In second try'
    except:
        print 'In second except'
    assert n >= 0, 'not >= 0'
    print 'Done second'
```

Function Calls:

i. `first(-1)`

```
'Start first'
'Start second'
'In second try'
'In first except'
'Done first'
```

ii. `first(1)`

```
'Start first'
'Start second'
'In second except'
'Done second'
'In first try'
'Done first'
```

iii. `first(0)`

```
'Start first'
'Start second'
'In second try'
'Done second'
'In first try'
'Done first'
```

Several people answered with the error messages in the two asserts (e.g. `'not >= 0'`). This is wrong. The error message does not display if it is caught by an `except`.

(c) [6 points] Below you have a function definition, together with several function calls. For each call, there is a list of printed output. Add traces and watches (e.g. print statements) to the function definition (in the space provided), so that all of the function calls produce *exactly* the output shown. We will accept any correct solution.

```
def foo(x):
    """Do something"""
    print 'Starting'
    print 'First if'
    if x < 0:

        result = -x+1
        print 'result is '+`result`

    else:

        result = x+1
        print 'result is '+`result`

    # COULD ALSO PRINT RESULT HERE
    print 'Second if'
    if x % 2 != 0:

        result = result+1
        print 'result is '+`result`

    else:

        result = 2*result


    print 'Ending'
```

Function Call: `foo(1)`

Printed Output:

```
'Starting'
'First if'
'result is 2'
'Second if'
'result is 3'
'Ending'
```

Function Call: `foo(2)`

Printed Output:

```
'Starting'
'First if'
'result is 3'
'Second if'
'Ending'
```

Function Call: `foo(-3)`

Printed Output:

```
'Starting'
'First if'
'result is 4'
'Second if'
'result is 5'
'Ending'
```

Function Call: `foo(-4)`

Printed Output:

```
'Starting'
'First if'
'result is 5'
'Second if'
'Ending'
```

4. [18 points] **String Slicing**.
When announcing the release date for a new product, companies have to pay attention to the official date format In the US our date format is `mm/dd/yy` (e.g. month, day, year). However, Europe often uses the format `dd/mm/yy` (e.g. day, month, year). For example, a product released on 3/6/12 in the US might be released 9/3/12 in Europe, which is just 3 days later.

As shown in the example above, sometimes the month and day may only be a single digit; in practice we pad these out to two digits with a leading zero. With this in mind, complete the function below using what you know about strings. You might find the following functions and/or methods useful.

Function or Method	Description
len(s)	Returns: number of characters in `s`; it can be 0.
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.rfind(s1)	Returns: index of the first character of the LAST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).

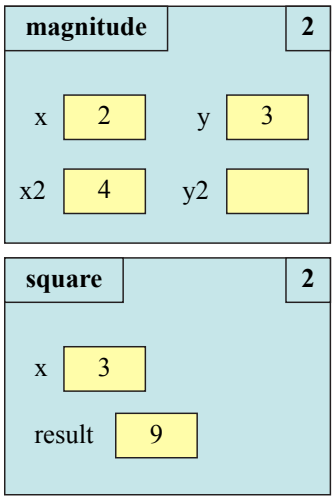
```
def europeanize(date):
    """Returns:  European version of this date (type is a string).
    Days and months are padded (if necessary) to become two digits each.
    Example: europeanize('3/6/12') is '06/03/12'
    Example: europeanize('01/29/11') is '29/01/11'
    Precondition: date a string representing a US date."""
    # Find dividers
    first = date.find('/')
    secnd = data.rfind('/')

    # Get components
    month = date[:first]
    day = date[first+1:secnd]
    year = date[secnd+1:]

    # Pad if necessary.
    if len(month) < 2:
        month = '0'+month


    if len(day) < 2:
        day = '0'+day


    # Put back together
    return day+'/'+month+'/'+year
```

5. Call Frames

(a) [8 points] Your are given two functions below. For the call `magnitude(2,3)`, draw the *entire call stack* when `magnitude` is in the middle of executing line 2 and `square` has just completed line 1 . You do not need to draw what the two call frames look like at any other time. Remember that we draw the most recently called function at the bottom.

```
def magnitude(x,y):
    """Returns: dist to origin"""
1   x2 = square(x)
2   y2 = square(y)
3   result = math.sqrt(x2+y2)
4   return result
```

```
def square(x):
    """Returns: square of x"""
1   result = x*x
2   return result
```

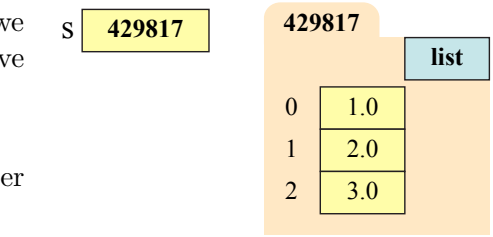


(b) [12 points]

If our value is a mutable container, like a list, then we represent the value using a folder. Suppose we have the following list in global space:

```
s = [1.0,2.0,3.0]
```

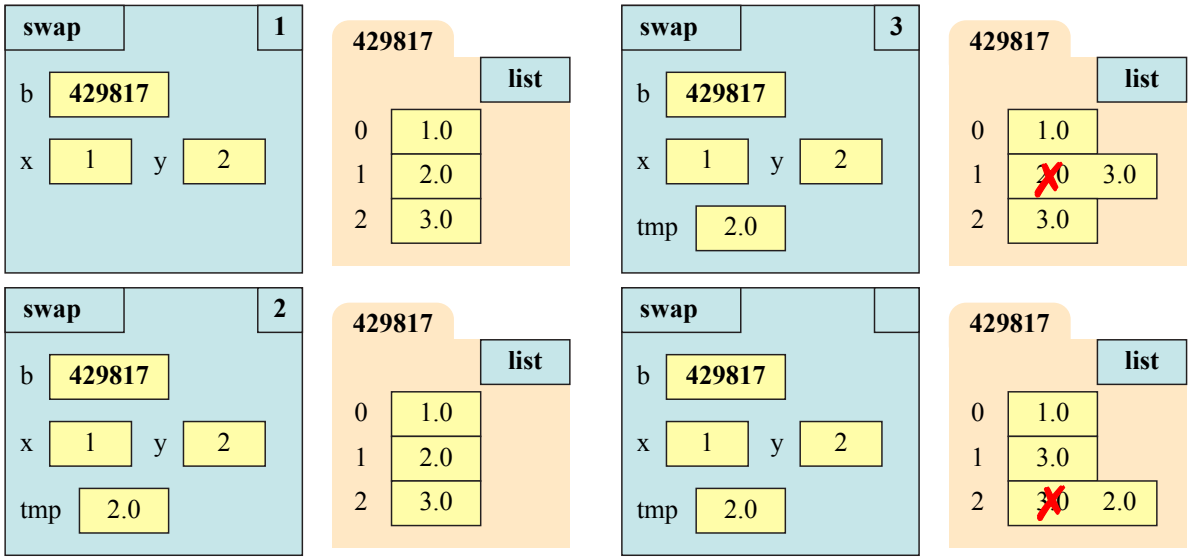
In this case, the variable `s` and its associated folder might look like the illustration to the right.



Consider the function definition below. Execute the function call `swap(s,1,2)`. By that we mean draw the call frame, and show how the call frames changes at each line in the function. To do this, you will draw what the frame looks like at four points in time: first when the function starts, and then once after it completes each of the line in the function body. Next to each time step, draw the current state of the folder for the list.

```
def swap(b,x,y):
1   tmp = b[x]
2   b[x] = b[y]
3   b[y] = tmp
```

Last Name: _____ First Name: _____ Cornell Netid: _____



6. [20 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 a *statement* and an *expression*?

   A *statement* is a command to do something. Each line in a function definition should be a statement.

   An *expression* represents a value. Python will evaluate an expression as part of a command, but it is not a command by itself.

   (b) [4 points] What types of things does Python store in *global space*? How does global space differ from a *call frame*?

   Global space stores all global variables and all function names in the active module.

   Unlike a call frame, the contents of global space only disappear when you quit Python or if you manually delete them.

(c) [4 points] Which of the following two expressions produces an error (none, one, or both)?

```
False and (5 / 0)          (5 / 0) or True
```

Explain why this is the case.

Only the second one produces an error. The first one does not produce an error because Python "short-circuits" the calculation; once it knows that the first value of an `and` is `False`, it does not need to evaluate anything else. But it must always evaluate the first expression, which is why the second one produces an error.

(d) [4 points] Explain the purpose of *preconditions* in a function specification. Why are they necessary?

Preconditions are a promise that our function will work properly if the arguments all satisfy the restrictions listed. They are necessary because we cannot possibly guarantee that our function will work on any arbitrary argument. For example, we cannot guarantee that a math function like `cos` will work on a string, or on a list. So we only guarantee the function on those arguments that satisfy the precondition.

(e) [4 points] What is an *attribute*, and what is an *attribute invariant*? How are they related?

An *attribute* is a named variable inside of an object. For example, `minutes` is an attribute in a `Time` object from the first question of the exam. An *attribute invariant* is a restriction on the types of values that may be assigned to the attribute. Again, `minutes` is an attribute in a `Time` object can only be assigned int values between 0 and 59.