CS 1110 Prelim 1 October 12th, 2017

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
```

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

Question	Points	Score
1	2	
2	14	
3	22	
4	20	
5	22	
6	20	
Total:	100	

The Important First Question:

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

Reference Sheet

Throughout this exam you will be asked questions about strings and lists. You are expected to understand how slicing works. In addition, the following functions and methods may be useful.

String Functions and Methods

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.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.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 or Method	Description
len(x)	**Returns**: number of elements in list x; it can be 0.
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).

The last three list methods are all procedures. They return the value None.

2. [14 points total] **Short Answer Questions**.

   (a) [3 points] What values are printed out to the screen after the following commands?

```
>>> a = [1,2,3]
>>> b = a[:]
>>> c = a
>>> c.insert(1,5)
>>> print(a)
>>> print(b)
```

(b) [3 points] What is a *statement*? What is an *expression*? How do they differ?

(c) [4 points] What is a *parameter*? What is an *argument*? How are they related?

(d) [4 points] Describe how we write a function specification in this class. We are looking for you to identify four important parts of the specification (though not every function specification has all of these parts).

3. [22 points] **Call Frames.**

Consider the following function definitions.

```
1  def magnitude(p):                           8  def square(x):
2      """Returns: dist to origin              9      """Returns: square of x"""
3      Precondition: p a list of 2 coords"""  10      return x*x
4      x = square(p[0])
5      y = square(p[1])
6      return (x+y) ** 0.5
7
```

The function `magnitude` treats a list like a 2-dimensional point and computes its distance from the origin. Assume that `q = [3.0,4.0]` is a global variable referencing a list in heap space, as shown on the next page. On the next two pages, diagram the evolution of the call

    d = magnitude(q)

Diagram the state of the *entire call stack* for the function `magnitude` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **nine** diagrams, not including the (pre-call) diagram shown.

You should draw also the state of global space and heap space at each step. You can ignore the folders for the function definitions. Only draw folders for lists or objects. You are also allowed to write "unchanged" if no changes were made to either global or heap space.

Hint: Pay close attention to the line numbers. They are different than those in the assignment.

Call Frames	**Global Space**	**Heap Space**

q | id1

id1

	list
0	3.0
1	4.0

Call Frames | **Global Space** | **Heap Space**

4. [20 points] **String Slicing**.

   Recall that a Cornell netid is a string with 2 or 3 initials, followed by an arbitrary number of digits. We not worry about case for netids, so `'wmw2'`, `'WMW2'` and `'Wmw2'` are all the same.

   Two netids are called *twins* if they have the same initials, and their numbers only differ by one. So `'wmw2'` and `'wmw3'` are twins, as are `'js209'` and `'JS210'`.

   Implement the function below. **You may not use loops**.

```
def twinsies(netid1,netid2):
    """Returns: True if netid1 and netid2 are twins, off by 1; False otherwise.

    Example: twinsies('wmw2','WmW3') is True (case is ignored when comparing)

    Precondition: netid1, netid2 are strings representing net-ids."""
```

5. [22 points total] **Testing and Debugging**.

(a) [10 points] Consider the following function header and specification:

```
def pairs(s1,s2):
    """Returns: The number of adjacent pairs of s2 inside s1

    Example: pairs('aabaa','a') is 2
             pairs('eeee','e') is 3

    Precondition: s1 is a nonempty string of lower case letters.
    Precondition: s2 is a single lower-case letter."""
```

Do not implement this function. Instead, write down a list of at least **five 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.*

(b) [12 points] You worked with the function `pigify` in lab. This function takes a string and converts it into Pig Latin according to the following rules:

1. The vowels are `'a'`, `'e'`, `'i'`, `'o'`, `'u'`, as well as any `'y'` that is *not* the first letter of a word. All other letters are consonants.

2. If the English word begins with a vowel, we append `'hay'` to the end of the word to get the Pig Latin equivalent.

3. If the English word starts with `'q'`, we assume it is followed by `'u'` (this is part of the precondition). We move `'qu'` to the end of the word, and append `'ay'`.

4. If the English word begins with a consonant, we move all the consonants up to the first vowel (if any) to the end and add `'ay'`.

Below are implementations of `pigify`, its helper `first_vowel`, as well as a new function, `sentipig`. The function `sentipig` allows us to apply Pig Latin to "sentences". However, for simplicity, we assume that the sentences have all lower case letters and no punctuation. They are really just multiple words separated by spaces.

There are **at least four bugs** in the code below. These bugs are across all functions and are not limited to a single function. To help find the bugs, we have added several print statements throughout the code. The result of running the code with these print statements shown on the next page. Using this information as a guide, identify and fix the four bugs on the page after this print-out. You should explain your fixes.

Hint: Pay close attention to the specifications. These versions of the functions are slightly different from those you worked with in lab.

```
1   def first_vowel(w):
2       """Returns: position of the first
3       vowel, or len(w) if no vowels.
4
5       Precondition: w is a nonempty string
6       with only lowercase letters"""
7       minpos = len(w) # no vowels found yet
8       vowels = 'aeio'
9
10      for v in vowels:
11          print('Looking for '+v)    # Trace
12          pos = w.find(v)
13          print('Pos is '+str(pos)) # Watch
14          if pos != -1 and pos < minpos:
15              minpos = pos
16
17      pos = w.find('y')
18      print('y Pos is '+str(pos))   # Watch
19      if pos != -1 and pos < minpos:
20          minpos = pos
21
22      return minpos
23
24
25  def pigify(w):
26      """Returns: copy of w in Pig Latin
27
28      Precondition: w a nonempty string
29      with only lowercase letters. If w
30      starts with 'q', w[1] == 'u'."""
31      pos = first_vowel(w)
32      if pos == 0:       # Starts w/ vowel
33          print('Vowel start')      # Trace
34          result = w+'hay'
35      elif w[0] == 'q': # Starts with q
36          print('Q start')          # Trace
37          resalt = w[2:]+'quay'
38      else:              # Standard case
39          print('Consonant start')  # Trace
40          result = w[pos:]+w[:pos]+'ay'
41
42      return result
```

```
43
44
45  def sentipig(w):
46      """Returns: Copy of sentence w with all
47      words converted to Pig Latin
48
49      Example: sentipig('barn owl') is
50      'arnbay owlhay'
51
52      Precondition: w a string of lower case
53      words, each separated by a space"""
54      result = ''      # Accumulator
55      start = 0        # Start of a word
56
57      for pos in range(len(w)):
58          if w[pos] == ' ':
59              # Watch
60              print('Space at pos '+str(pos))
61              word = w[start:pos]
62              # Watch
63              print('Word is '+repr(word))
64              result += pigify(word)+' '
65              # Watch
66              print('Result is '+repr(result))
67              start = pos
68
69      word = w[start:] # Last word
70      # Watch
71      print('Last word is '+repr(word))
72      result += pigify(word)
73
74      return result
75
76
77  # Remaining lines are blank
78
79
80
81
82
83
84
```

Tests:

```
>>> sentipig('blue')
Last word is 'blue'
Looking for a
Pos is -1
Looking for e
Pos is 3
Looking for i
Pos is -1
Looking for o
Pos is -1
y Pos is -1
Consonant start
ebluay
```

```
>>> sentipig('quick')
Last word is 'quick'
Looking for a
Pos is -1
Looking for e
Pos is -1
Looking for i
Pos is 2
Looking for o
Pos is -1
y Pos is -1
Q start
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pigify.py", line 67, in sentipig
    result += pigify(word)
  File "pigify.py", line 42, in pigify
    return result
UnboundLocalError: local variable 'result'
  referenced before assignment
```

```
>>> sentipig('wierd quirk')
Space at pos 5
Word is 'wierd'
Looking for a
Pos is -1
Looking for e
Pos is 2
Looking for i
Pos is 1
Looking for o
Pos is -1
y Pos is -1
Consonant start
Result is 'ierdway '
Last word is ' quirk'
Looking for a
Pos is -1
Looking for e
Pos is -1
Looking for i
Pos is 3
Looking for o
Pos is -1
y Pos is -1
Consonant start
ierdway irk quay
```

```
>>> sentipig('yellow owl')
Space at pos 6
Word is 'yellow'
Looking for a
Pos is -1
Looking for e
Pos is 1
Looking for i
Pos is -1
Looking for o
Pos is 4
y Pos is 0
Vowel start
Result is 'yellowhay '
Last word is ' owl'
Looking for a
Pos is -1
Looking for e
Pos is -1
Looking for i
Pos is -1
Looking for o
Pos is 1
y Pos is -1
Consonant start
yellowhay owl ay
```

First Bug:

Second Bug:

Third Bug:

Fourth Bug:

6. [20 points] **Objects and Functions**.

Rectangle objects are very common in computer graphics; they are used to indicate a region of pixels on your screen. Rectangles have four attributes with the following invariants.
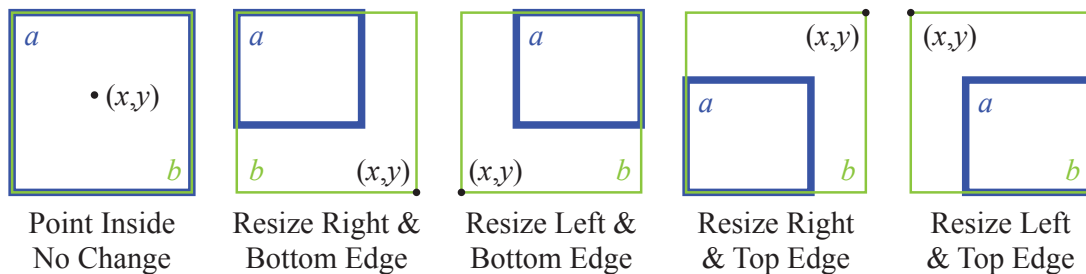
Attribute	Meaning	Invariant
x	position of left edge	int value
y	position of top edge	int value
width	distance from left to right edge	int value $>= 0$
height	distance from top to bottom edge	int value $>= 0$

The invariants all specify `int` values because pixel positions are whole numbers. Furthermore, y-coordinates on the screen get larger **downwards**; the origin is the top left corner of the screen.

To make a Rectangle object, call the function `Rectangle(x,y,w,h)` (do not worry about the module), giving the values for the attributes in order. The constructor enforces the object invariants, and will cause an error if you violate them. Note that it is perfectly okay to have a rectangle whose width or height is equal to 0.

Sometimes we want to *expand* a rectangle to include a point. An expansion is the minimal rectangle containing both the original rectangle and the point. This expansion has at least two edges in common with the original rectangle, since we only need to move two edges to the point.

For example, suppose we have a point (x,y) and a rectangle a. The illustration below shows how we might expand a to get a new rectangle b. In the first picture, the point is inside of the rectangle, so there is nothing to do. In the other cases, we have to resize two edges to meet the point (x,y). These are not the only cases (sometimes we only need to resize one of the two edges), but they are enough to give the idea.

| Point Inside No Change | Resize Right & Bottom Edge | Resize Left & Bottom Edge | Resize Right & Top Edge | Resize Left & Top Edge |

Using the illustration above, implement the function on the next page.

Hint. When you write your if-statements, only use just one of `x` or `y` in each if-statement, *not both*. This will make the function a lot simpler and cut down on the number of if-statements. You are also a lot less likely to miss a case this way.

```python
def expand(rect,x,y):
    """Expands this rectangle to include the point (x,y).

    This function is a procedure and modifies the object rect.

    Precondition: rect is a Rectangle object. x and y are ints."""
```