Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 1 October 12th, 2023

This 90-minute exam has 6 questions worth a total of 99 points. Read over 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 should not use loops or recursion on this exam. Beyond that, you may use any Python feature that you have learned in class (if-statements, try-except, lists), **unless directed otherwise**.

Question	Points	Score
1	2	
2	16	
3	16	
4	23	
5	22	
6	20	
Total:	99	

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: _____

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

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.count(s1)`	**Returns**: the number of times `s1` occurs in `s`
`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.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.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or numbers; it returns False otherwise.
`s.islower()`	**Returns**: True if `s` is *has at least one letter* and all letters are lower case; it returns False otherwise (e.g. `'a123'` is True but `'123'` is False).
`s.isupper()`	**Returns**: True if `s` is *has at least one letter* and all letters are upper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.lower()`	**Returns**: A copy of `s` with all letters lower case.
`s.upper()`	**Returns**: A copy of `s` with all letters upper case.

List Functions 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**: the number of times `y` occurs in `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.index(y,n)`	**Returns**: index of the first occurrence of `y` in `x` STARTING at position n (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`.

Last Name: _____ First: _____ Netid: _____

2. [16 points total] **Short Answer Questions**.

(a) [5 points] What is the definition of a type in Python? List at least four examples of types built into Python.

A type is a collection of values together with the operations on them. The four basic types are `int`, `float`, `bool`, and `str`. However, we would also allow `list` or `tuple`.

(b) [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).

A specification is written as a docstring at the start of the body with the following:

- This docstring starts with a single line summary (includin the value returned).
- This is followed by one or more paragraphs giving more detail about the function.
- Each parameter is identified and described in the specification.
- There is a precondition for each parameter, identifying what values it can take.

(c) [4 points] Consider the following code:

```
1  def func1(x):
2      print('Start one')
3      try:
4          print('Try one')
5          y = func2(x)
6      except:
7          print('Recover one')
8          y = 2
9      print('Done one')
10     return y
11
```

```
12 def func2(x):
13     print('Start two')
14     try:
15         print('Try two')
16         y = func3(x)
17     except:
18         print('Recover two')
19         assert x < 0, 'Bad!'
20     print('Done two')
21     return y
22
```

```
23 def func3(x):
24     print('Start three')
25     y = x/0
26     print('Done three')
27     return y
28
29
30
31
```

What is printed out when we call `func1(2)`?

```
'Start one'
'Try one'
'Start two'
'Try two'
'Start three'
'Recover two'
'Recover one'
'Done one'
```

(d) [3 points] What is a watch? What is a trace? What purpose do they serve?

A *watch* is a print statement used to display the contents of a variable. A *trace* is a print statement used to visualize program flow. Both are useful in debugging code. They allow us to visualize code execution.

3. [16 points] **String Slicing**.

Implement the function below according the specification. Remember to refer to the reference guide on the second page.

```
def next_id(nid):
    """Returns the netid following nid.

    The returned netid has the same letters as nid, except that it is all lower
    case. The number is incremented by 1.

    Examples: next_id('jma4') returns 'jma5'
              next_id('rd99') returns 'rd100'
              next_id('WmW2') returns 'wmw3'

    Precondition: nid is 2 or 3 letters following by digits"""

    # Find the digits
    if nid[2].isalpha():
        pos = 3
    else:
        pos = 2

    # Increment the number
    n = int(nid[pos:])
    n = n+1

    # Adjust the letters and glue number
    pref = nid[:pos]
    pref = pref.lower()
    return pref+str(n)
```

Last Name: _____ First: _____ Netid: _____

4. [23 points] **Call Frames.**

Consider the following function definitions.

```
1  def shuffle(a):
2      """Returns a shuffled copy of a
3      Pre: a is a nonempty list"""
4      b = cut(a,True)
5      c = cut(a,False)
6      return c+b
7
```

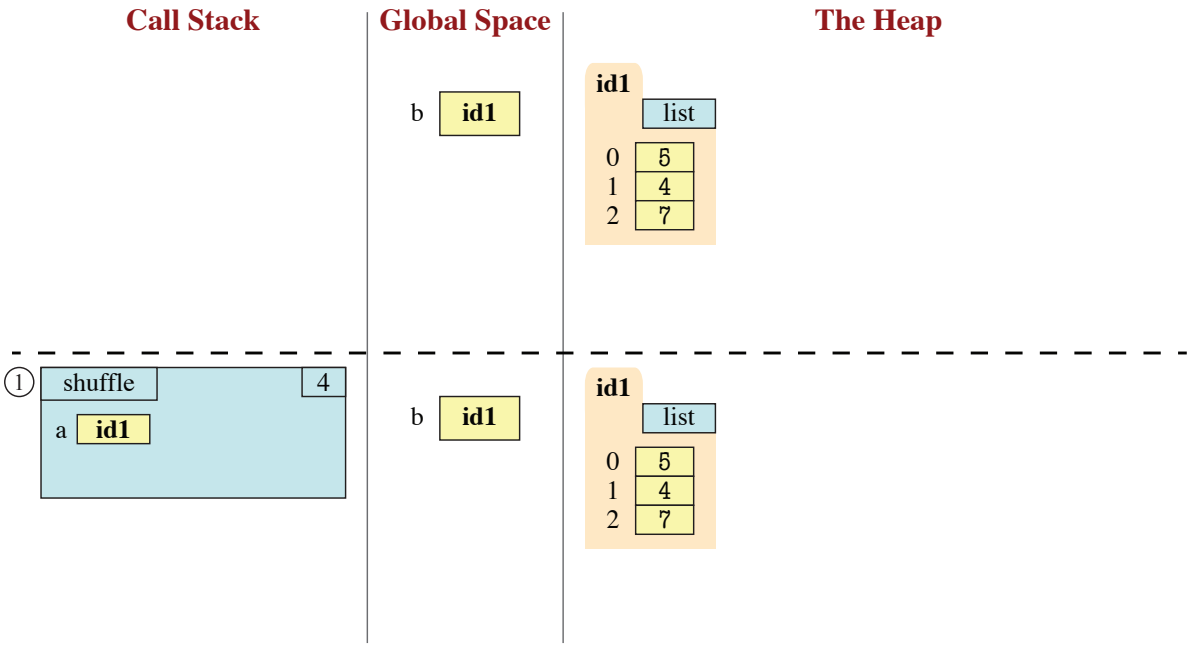
```
8  def cut(lst,up):
9      """Returns a slice of lst
10     Pre: lst is nonempty list, up a bool"""
11     if up:
12         return lst[:1]
13     else:
14         return lst[1:]
```

Assume `b = [5, 4, 7]` is a global variable referencing a list in the heap, as shown below. **Starting on this page**, and continuing on the next two pages, diagram the evolution of the function call

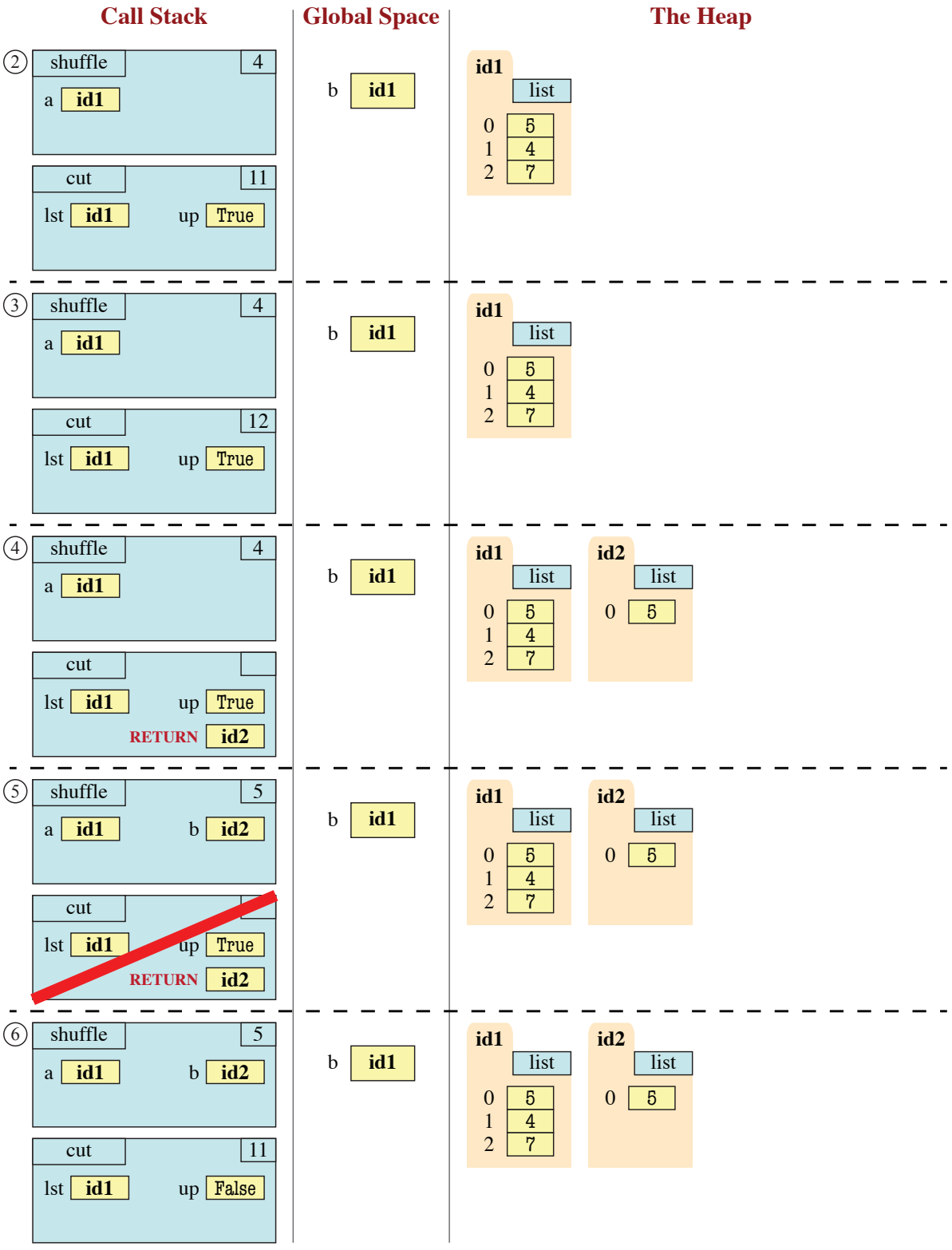
```
c = shuffle(b)
```

Diagram the state of the *entire call stack* for the function `shuffle` 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 **eleven** diagrams, not including the first diagram on the next page

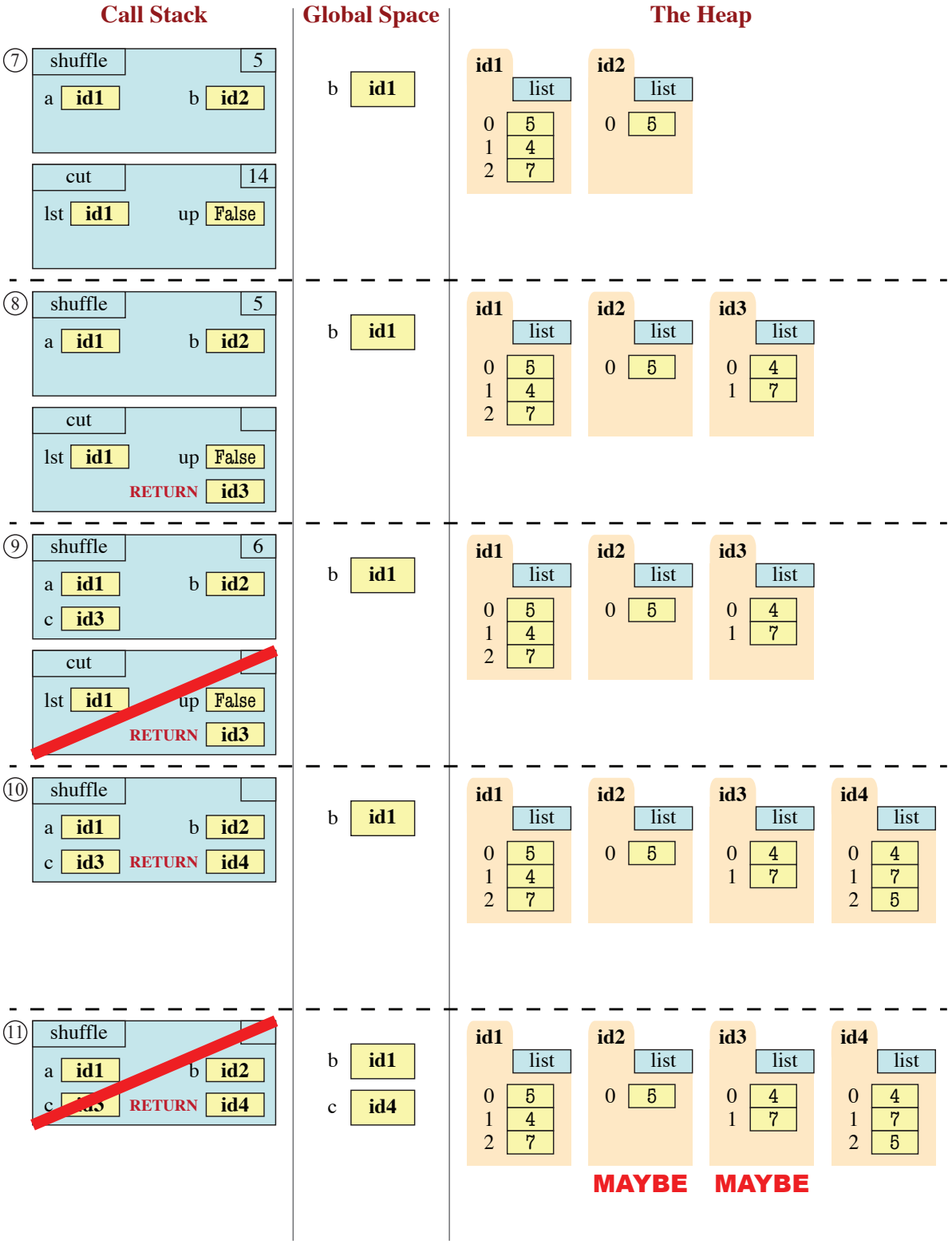
You should draw also the state of global space and the heap at each step. You can ignore the folders for the function definitions. Only draw folders for lists or objects. To help conserve time, you are allowed (and **encouraged**) to write "unchanged" if no changes were made to either a call frame, the global space, or the heap.



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

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.*

There are many different possible answers to this question. Below are the different solutions we were thinking of. If you had (at least) five test cases that were close to the ones below, you got full credit. Otherwise, we checked if your test cases were *different enough*, and awarded you 2 points for each test.

Inputs		Output	Reason
`s1='bbb'`	`s2='a'`	0	String `s2` not in `s1`
`s1='aba'`	`s2='a'`	0	String `s2` in `s1`, but no adjacent pairs
`s1='aab'`	`s2='a'`	1	Single adjacent pair, character appears twice
`s1='aaba'`	`s2='a'`	1	Single adjacent pair, character appears later
`s1='aabaa'`	`s2='a'`	2	Multiple, non-overlapping adjacent pairs
`s1='aaaa'`	`s2='a'`	3	Multiple, overlapping adjacent pairs

(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'`.

Last Name: _____ First: _____ Netid: _____

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 '+repr(pos)) # Watch
14         if pos != -1 and pos < minpos:
15             minpos = pos
16
17     pos = w.find('y')
18     print('y Pos is '+repr(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 def sentipig(w):
44     """Returns: Copy of sentence w with all
45     words converted to Pig Latin
46
47     Example: sentipig('barn owl') is
48     'arnbay owlhay'
49
50     Precondition: w a string of lower case
51     words, each separated by a space"""
52     result = ''     # Accumulator
53     start = 0       # Start of a word
54
55     for pos in range(len(w)):
56         if w[pos] == ' ':
57             # Watch
58             print('Space at pos '+repr(pos))
59             word = w[start:pos]
60             # Watch
61             print('Word is '+repr(word))
62             result += pigify(word)+' '
63             # Watch
64             print('Result is '+repr(result))
65             start = pos
66
67     word = w[start:] # Last word
68     # Watch
69     print('Last word is '+repr(word))
70     result += pigify(word)
71
72     return result
73
74
75 # Remaining lines are blank
76
77
78
79
80
81
82
83
84
```

Last Name: _____ First: _____ Netid: _____

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 62, in sentipig
    result += pigify(word)
  File "pigify.py", line 42, in pigify
    return result
UnboundLocalError: local variable 'result'
  referenced before assignment
```

```
>>> sentipig('weird quirk')
Space at pos 5
Word is 'weird'
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
Consonant start
Result is 'eirdway '
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
eirdway 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:

The bug for the test `sentipig('blue')` is in `first_vowel`. We are missing the vowel `'u'`. To fixe this, Line 8 should be

```
vowels = 'aeiou'
```

Second Bug:

The bug for the test `sentipig('quick')` is in `pigify`. Even though the trace shows that we have gone to the correct `elif`, we have misspelled the variable `result`, causing the crash. To fix this, Line 37 should be

```
result = w[2:]+'quay'
```

Third Bug:

The bug for `sentipig('duck quack')` is in `sentipig`. When there are multiple words, we are accidentally including the space in the later words. To fix this, Line 65 should be

```
start = pos+1
```

Note that we would accept stripping the space. However, that solution has to be applied to **both** lines 67 **and** 59.

Fourth Bug:

The bug for `sentipig('yellow owl')` is in `first_vowel`. This function is treating the first `'y'` as a vowel. To fix this, Line 17 should be

```
pos = w.find('y',1)
```

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

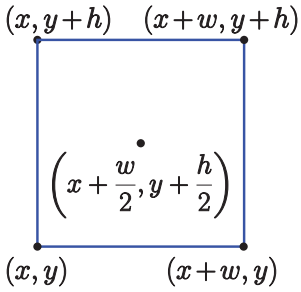
Rectangle objects are very common in computer graphics; they are used to indicate a region of pixels on your screen. Rectangles have four attributes with the following invariants.

Attribute	Meaning	Invariant
`x`	position of left edge	`int` value
`y`	position of bottom edge	`int` value
`width`	distance from left to right edge	`int` value >= 0
`height`	distance from bottom to top edge	`int` value >= 0

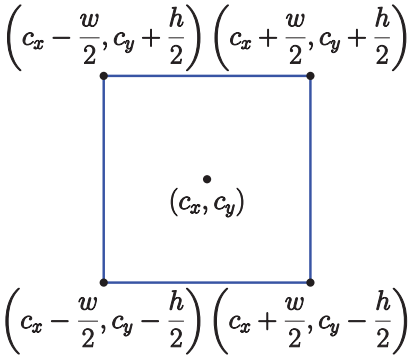
The invariants all specify `int` values because pixel positions are whole numbers.

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.

While we only have (x, y) for the bottom left corner of the rectangle, we can use the width and height to compute the positions of the other corners (as well as the center), as shown below:



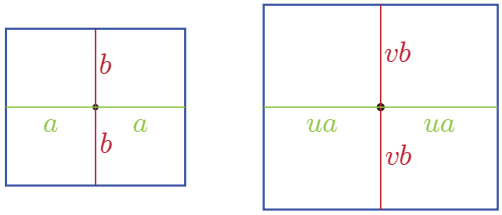
Positions computed from bottom left



Positions computed from the center

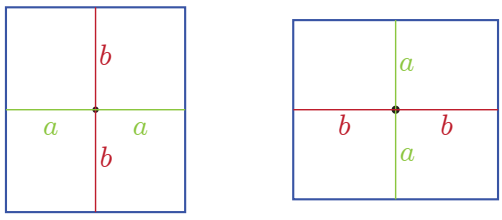
However, both of the functions on the page requires you to compute a new rectangle from the value of the center point. Use the picture above to see how to go back-and-forth between the center and the bottom left corner.

Hint: All of these functions are straight-forward as long as long as you understand the two illustrations below:



Original Scaled Copy

Figure 1: Illustration of function `scale(rect,u,v)`



Before After

Figure 2: Illustration of function `rotate(rect)`

(a) [9 points] Implement the function below according to the specification.

```
def scale(rect,u,v):
    """Returns a new rectangle that is the original scaled by (u,v)

    Scaling multiplies the width by u and the height by v to change the size
    of the rectangle. The new rectangle has the same center as the original.
    The original rectangle should not be modified.

    Preconditions: rect is a rectangle, u and v are floats > 0."""
    # Compute the center
    cx = rect.x+rect.width/2       # Would accept rect.width//2 or similar
    cy = rect.y+rect.height/2      # Would accept rect.height//2 or similar

    # Find the bottom left corner
    x = cx - u*rect.width/2        # Would accept rect.width//2 or similar
    y = cy - v*rect.height/2       # Would accept rect.height//2 or similar

    # Find the new width and height
    w = u*rect.width
    h = v*rect.height

    # We will take round or any other way to make ints
    return Rect(int(x),int(y),int(w),int(h))
```

(b) [11 points] Implement the function below according to the specification.

```
def rotate(rect):
    """MODIFIES the rectangle so that is rotated 90 degrees

    Rotation swaps the width and height, but the center of the rectangle
    remains unchanged.

    Preconditions: rect is a rectangle"""

    # Compute the center
    cx = rect.x+rect.width/2       # Would accept rect.width//2 or similar
    cy = rect.y+rect.height/2      # Would accept rect.height//2 or similar

    # Swap the width and height
    w = rect.height
    h = rect.width

    # Find the bottom left corner
    # We will take round or any other way to make ints
    rect.x = int(cx - w/2)
    rect.y = int(cy - h/2)

    # Update the width and height
    rect.width  = w
    rect.height = h
```