Last Name: ____________ First: ____________ Netid: ________ Section ________

CS 1110 Prelim 1 October 15th, 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, 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 recursion 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 and so on), **unless directed otherwise**.

Question	Points	Score
1	2	
2	18	
3	18	
4	18	
5	22	
6	22	
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.

Last Name: ____________ First: ____________ Netid: ________ Section ________

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.
`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.rfind(s1,n)`	**Returns**: index of the first character of the last occurrence of `s1` in `s` STARTING at position n. (-1 if `s1` does not occur in s from this position).
`s.count(s1)`	**Returns**: number of (non-overlapping) occurrences of substring `s1` 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.replace(a,b)`	**Returns**: A *copy* of `s` where all instances of substring `a` are replaced with the substring `b`.
`s.strip()`	**Returns**: A *copy* of `s` with all spaces at either the beginning or end removed from `s`.

List Functions and Methods

Function 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.index(y)`	**Returns**: index of the FIRST occurrence of `y` in `x` (an error occurs if `y` does not occur in `x`).
`x.count(y)`	**Returns**: number of times `y` appears in the list `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`).
`x.sort()`	Sorts the elements of `x` according to the regular Python order.

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

2. [18 points total] **Short Answer Questions**.

   (a) [6 points] What is the definition of a type in Python? List at least four examples of types that we have seen in class.

   A type is a collection of values together with the operations on them. The four basic types are `int`, `float`, `bool`, and `str`.

   (b) [4 points] What is an *expression*? What is a *statement*? Give an example of each.

   An expression is something that Python evaluates to a value. `1+1` is an example of an expression.

   A statement is a command for Python to do something. The assignment statement `x = 1+1` is a statement. In addition to evaluating `1+1`, it creates a variable `x` and stores the value in this variable.

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

   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.

   (d) [4 points] Consider the function `foo` defined below.

```
def foo():
    return 5
```

   What are the contents of the variables `x` and `y` after the assignments below? Explain the difference between the two.

```
x = foo()
y = foo
```

   `x` contains the value 5 because `foo()` is a function call. On the other hand, `y` contains the name of the folder storing the function definition, because `foo` is not a call.

3. [18 points] **String Slicing**. The game publisher Ubisoft has managed to buy the studio BioWare, and is changing the name of the entire company to UbiWare. As part of their company transition, they need to issue new e-mail addresses for all employees. The new e-mails should have the same names as the old ones, but in a different format.

   - UbiSoft e-mails have the form `last.first@ubisoft.com`
   - BioWare e-mails have the form `first.middle.last@bioware.com`
   - The new UbiWare e-mails will have the form `first.last@ubiware.com`

   Complete the function specified below

```
def make_email(s):
    """Returns: A string representing the new e-mail for s.

    The e-mail will have the UbiWare format described above.

    Examples:
       make_email('evans.bob@ubisoft.com') is 'bob.evans@ubiware.com'
       make_email('fred.e.smith@bioware.com') is 'fred.smith@ubiware.com'

    Precondition: s is a string representing a Ubisoft or a BioWare e-mail."""
    # Want variables for first and last names
    # This step is a style issue, but it unnecessary
    first = ''
    last = ''

    ppos = s.find('.')  # position of period
    apos = s.find('@')  # position of @ symbol

    # Check which company this is
    if '@ubisoft' in s:    # need @ to eliminate people named ubisoft
        last = s[:ppos]
        first = s[ppos+1:apos]
    else:
        spos = s.find('.',ppos+1) # position of second period
        first = s[:ppos]
        last = s[spos+1:apos]

    result = first+'.'+last+'@ubiware.com'
    return result
```

4. [18 points total] **Errors and Testing**.

Throughout this question, you will work with *string lists*. A string list looks like a list, but it is actually a string. They are useful because it is much easier to send text over the Internet than it is to send lists (for the same reason that a JSON is a string, but looks like a dictionary).

A string list is string that starts and ends with square bracket characters, and contains a sequence of digits separated by commas. A string list has no spaces or characters other than digits or commas. So `'[1,2,3]'` is a valid string list, but `'[1, 2, 3]'` and `'[1.5,2,3]'` are not. The empty brackets `'[]'` is also a valid string list. String lists have the following functions:

```
def is_slist(s):
    """Return: True if the string s is a string list; False otherwise.
    Precondition: s is a string"""

def count_slist(s,value):
    """Return: The number of occurrences of value in string list s.
    Example: count_slist('[1,2,1]',1) returns 2, count_slist('[1,2,1]',3) returns 0
    Preconditions: s is a string list, value is an int."""
```

Do not implement these functions. Use their specifications to answer the questions below.

(a) [8 points] Provide at **least four** different test cases to verify that the function `count_slist` is working correctly. For each test case provide:

- The input, or function arguments.
- The expected output, or what the function should return.
- A short explanation of why that test is important, and 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) four 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**
`s='[]', a=1`	0	String list `s` is empty
s=`'[1,2,3]'`, a=1	1	One occurrence of `a`
s=`'[1,2,3]'`, a=4	0	No occurrences of `a`
s=`'[1,2,1]'`, a=1	2	Multiple occurrences of `a`

(b) [4 points] Write one or more `assert` statements to enforce the preconditions of the function `count_slist`. Your assert statements **do not** need to have error messages.
Hint: You may want to use `is_slist` to help with this problem.

```
assert type(s) == str # Check type of s (precondition to is_slist
assert is_slist(s) # Check if s is a string list
assert type(value) == int # Check type of value
```

(c) [6 points] The function defined below is buggy and does not work. There are (at least) two bugs in it. To help find the bugs, we have added several watch statements throughout the code. The result of running the code with these watch statements is shown to the right. Using this information as a guide, identify and fix the two bugs. Explain your fixes below. **Hint**: Do not "fix" the line marked NOT A BUG.

```
def get_slist(s,pos):
    """Returns: The int at position pos

    The result is an int, not a string.

    Examples:
       get_slist('[1,2]',0) is 1
       get_slist('[1,2]',1) is 2

    Preconditions: s a string list,
    pos is an integer less than the
    number of ints in s"""
    # Replace ] with , to aid search
    # Ex: '[1,2,3]' becomes '[1,2,3,'
    s.replace(']',',')
    print 's = '+s

    # Find first item in the list
    # Mark start of item (after the [)
    start = 1
    # Comma marks end of item
    end = s.index(',')
    print 'end = '+str(end)

    # Move forward through commas
    # start is item, end is comma
    for x in range(pos):
        print 'x = '+str(x)
        start = end
        print 'start = '+str(start)
        end = s.index(',',start+1)
        print 'end = '+str(end)

    # Precond. guarantees we can slice
    slice = s[start:end] # NOT A BUG
    print 'slice = '+slice
    return int(slice)
```

```
>>> get_slist('[1]',0)
s = [1]
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   File "<stdin>", line 19, in get_slist
      end = s.index(',')
ValueError: substring not found
>>> get_slist('[1,2,3]',1)
s = [1,2,3]
end = 2
x = 0
start = 2
end = 4
slice = ,2
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   File "<stdin>", line 34, in get_slist
      return int(slice)
ValueError: invalid value for int: ',2'
```

Error 1: The call `s.replace(']',',')` should instead be the assignment statement

`s = s.replace(']',',')`

Strings are not mutable, and the method `replace` does not change the contents of the string. It only makes a new copy of the string. So we have to assign the new copy to `s`.

Error 2: Inside of the body of the for loop, the assignment to `start` should be

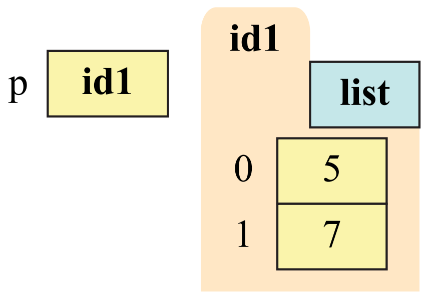
`start = end+1`

This is because the variable `end` always marks the position of a comma in the string. The next element starts *after* the comma. We have to change `start` if `slice` is not a bug.

5. [22 points] **Call Frames.**

Consider the following function definition and specification.

```
def reverse(q):
    """Return: a copy of q, reversed
    Precondition: q is a list"""
    r = []                          1
    for x in q:                     2
        r.insert(0,x)               3
    return r                        4
```

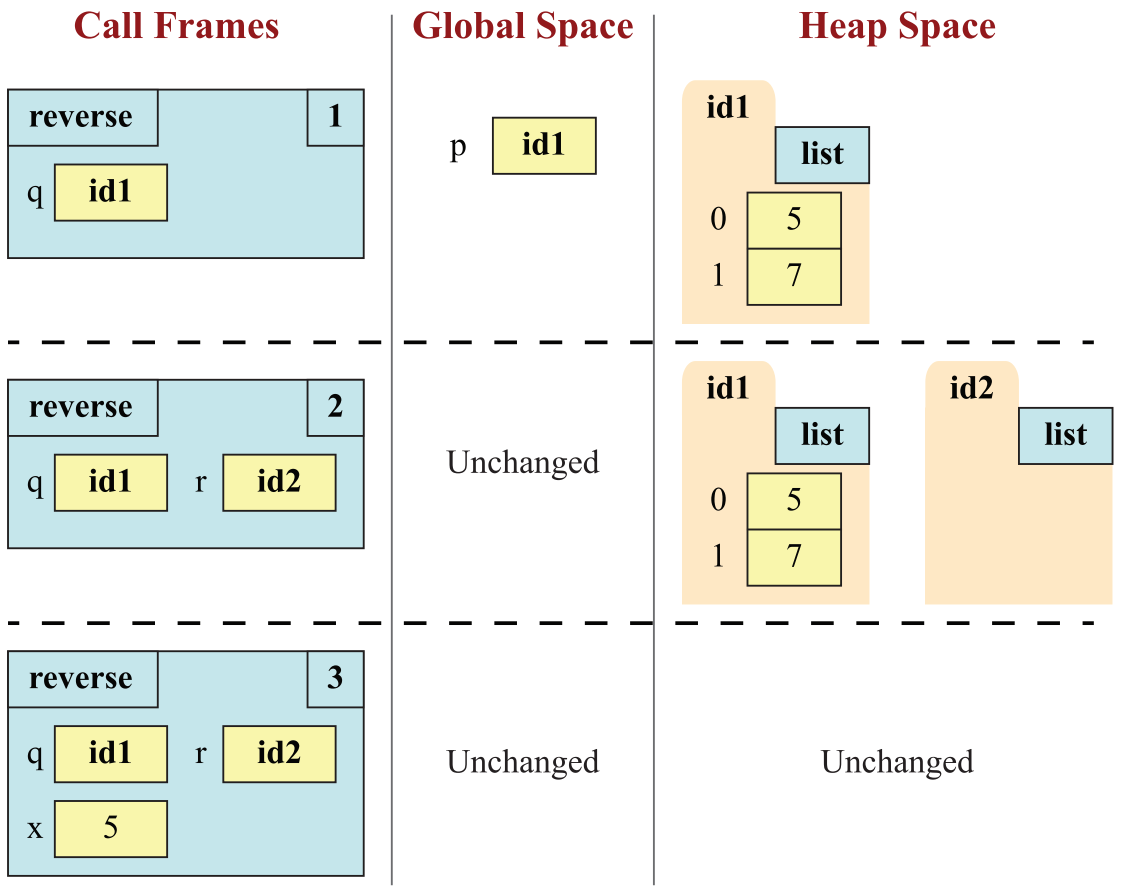


Assume that `p = [5,7]` is a global variable that stores a reference to a list in heap space, as shown above. On this page and the next, diagram the evolution of the call

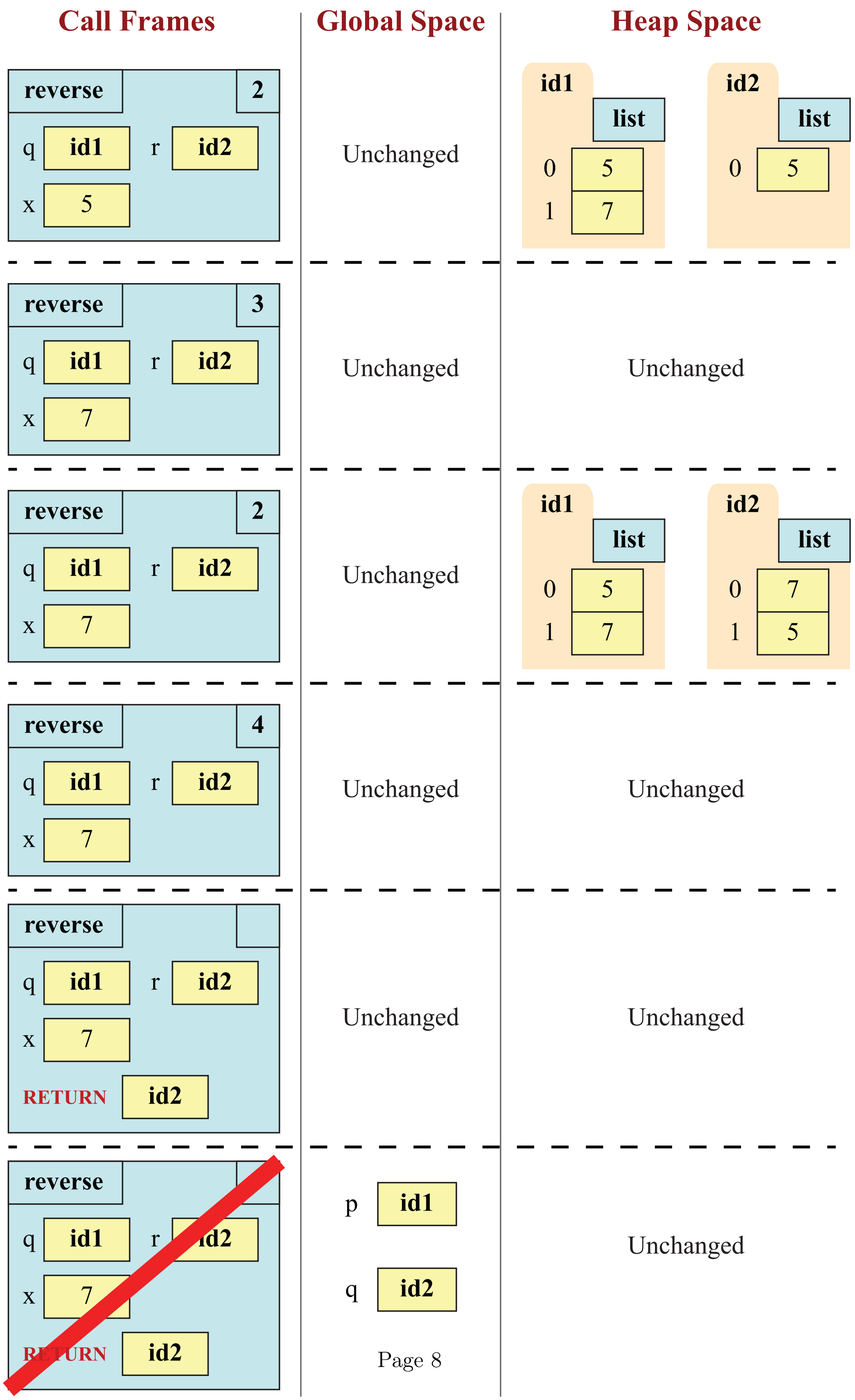
```
q = reverse(p)
```

Diagram the state of the call frame for the function `reverse` when it starts, for each line executed, and when the frame is erased. **You do not need to draw any other call frames**; you can ignore the frame for `insert`. We have already drawn the first step for you. There are **eight** more diagrams to draw, each separated by a dotted line.

In addition to the call frame, draw the state of global space and heap space. You are allowed to write "unchanged" if no changes were made to either global or heap space.



Last Name: ____________ First: ____________ Netid: ________ Section ________



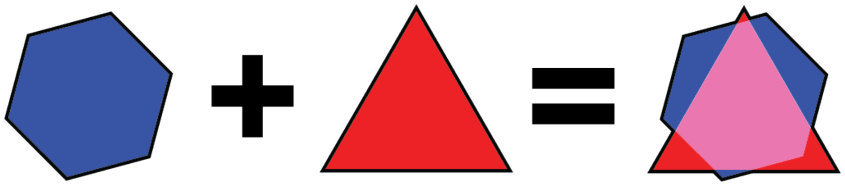
6. [22 points total] **Objects and Functions.**

Remember the class `RGB` from Assignment 3. Objects of this class have three attributes: `red`, `green`,and `blue`. These values must be integers between 0 and 255; assigning any other value to them will result in an error.

As some of you may have realized, `RGB` actually has another attribute: `alpha`. This attribute has the same restriction as the others: it must be an integer between 0 and 255. This means that when you create an `RGB` object, the proper constructor call is `RGB(r,g,b,a)` (do not worry about referencing `colormodel` for this question). We ignored this attribute in Assignment 3, but we will use it in this part of the test.

The alpha attribute is used for *color composition*. Each pixel on your computer monitor can only show only one color at a time. If we want to show multiple images on top of each other, we have to combine the colors together.

(a) [10 points] The simplest form of color composition is straight addition. We add the two red values to get the new red, the two green values to get the new green, and so on. The result of this is very similar to mixing colors in primary school, as shown below.



Note that the attribute invariants can never be violated. If adding together two color results in an attribute that is greater than 255, we use the value of 255 instead. With this in mind, implement the following procedure according to the specification.

```
def add_color(color1,color2):
    Adds the color color1 to color2 and stores the result in color1.

    Addition is on each atttribute (including alpha) as described above.

    This function is a PROCEDURE and has no return value.

    Preconditions: color1 and color2 are RGB objects.

    # Add red, and handle the case we go over 255
    red = color1.red+color2.red
    color1.red = min(red,255)

    # Handle the remaining attributes
    green = color1.green+color2.green
    color1.green = min(green,255)

    blue = color1.blue+color2.blue
    color1.blue = min(blue,255)

    alpha = color1.alpha+color2.alpha
    color1.alpha = min(alpha,255)
```

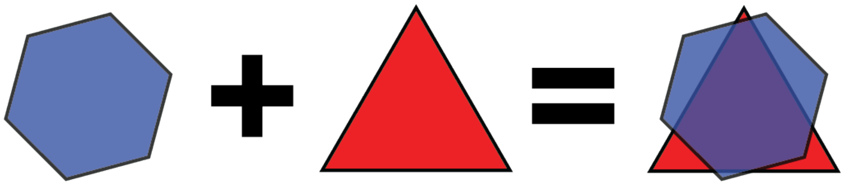
(b) [12 points] A more popular form of color composition is alpha-blending. This allows one image to be transparent on top of another. The attribute `alpha` represents the amount of transparency. An `alpha` of 255 means the color is completely opaque, while an `alpha` of 0 means the color is completely transparent.

To perform alpha-blending, you must convert the `red`, `green`, `blue`, and `alpha` values to the range 0 to 1, just as in Assignment 3. You then apply the following formula:

$R' = \alpha_1 R_1 + (1-\alpha_1)R_2$	**Combine red attributes of colors 1 and 2**
$G' = \alpha_1 G_1 + (1-\alpha_1)G_2$	**Combine green attributes of colors 1 and 2**
$B' = \alpha_1 B_1 + (1-\alpha_1)B_2$	**Combine blue attributes of colors 1 and 2**
$\alpha' = \alpha_1 + (1-\alpha_1)\alpha_2$	**Combine alpha attributes of colors 1 and 2**

In the formula above, R_1, G_1, ... are the attributes of the first color, R_2, G_2, ... are the attributes of the second color, and R', G', ... are the attributes of the new color. Remember to convert back to the range 0..255 when done.

In practice, alpha-blending looks like the picture below.



Use this formula (and what you know from Assignment 3) to implement the function below.

```
def alpha_blend(color1,color2):
    Returns: a new color that is the blend of color1 and color2.

    Blending follows the rules of the formula above.

    Preconditions: color1 and color2 are RGB objects.

    # Convert the alpha of color1 to 0..1
    alpha = color1.alpha/255.0

    # Compute the new colors as a value 0..1
    red = alpha*color1.red/255.0+(1-alpha)*color2.red/255.0
    green = alpha*color1.green/255.0+(1-alpha)*color2.green/255.0
    blue = alpha*color1.blue/255.0+(1-alpha)*color2.blue/255.0
    alpha = alpha+(1-alpha)*color2.alpha/255.0

    # Convert to integers
    red = int(red*255)   # rounding allowed, but not necessary
    green = int(green*255)
    blue = int(blue*255)
    alpha = int(alpha*255)

    # Construct a new RGB object and return it
    return RGB(red,green,blue,alpha)
```