Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 1 October 19th, 2021

This 90-minute exam has 6 questions worth a total of 100 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	8	
3	20	
4	23	
5	25	
6	22	
Total:	100	

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. [8 points total] **Short Answer**.

   (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) [3 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.

3. [20 points] **Call Frames.**

   Consider the following function definitions.

```
1  def double(a):
2      """Returns copy of a but doubled
3      Pre: a is a list"""
4      b = a[:]
5      c = a+b
6      return c
7
```

```
8  def mystery(a):
9      """Modifies a and returns something
10     Pre: a is a nonempty list"""
11     a[-1] = a[0]
12     return double(a)
13
14
```

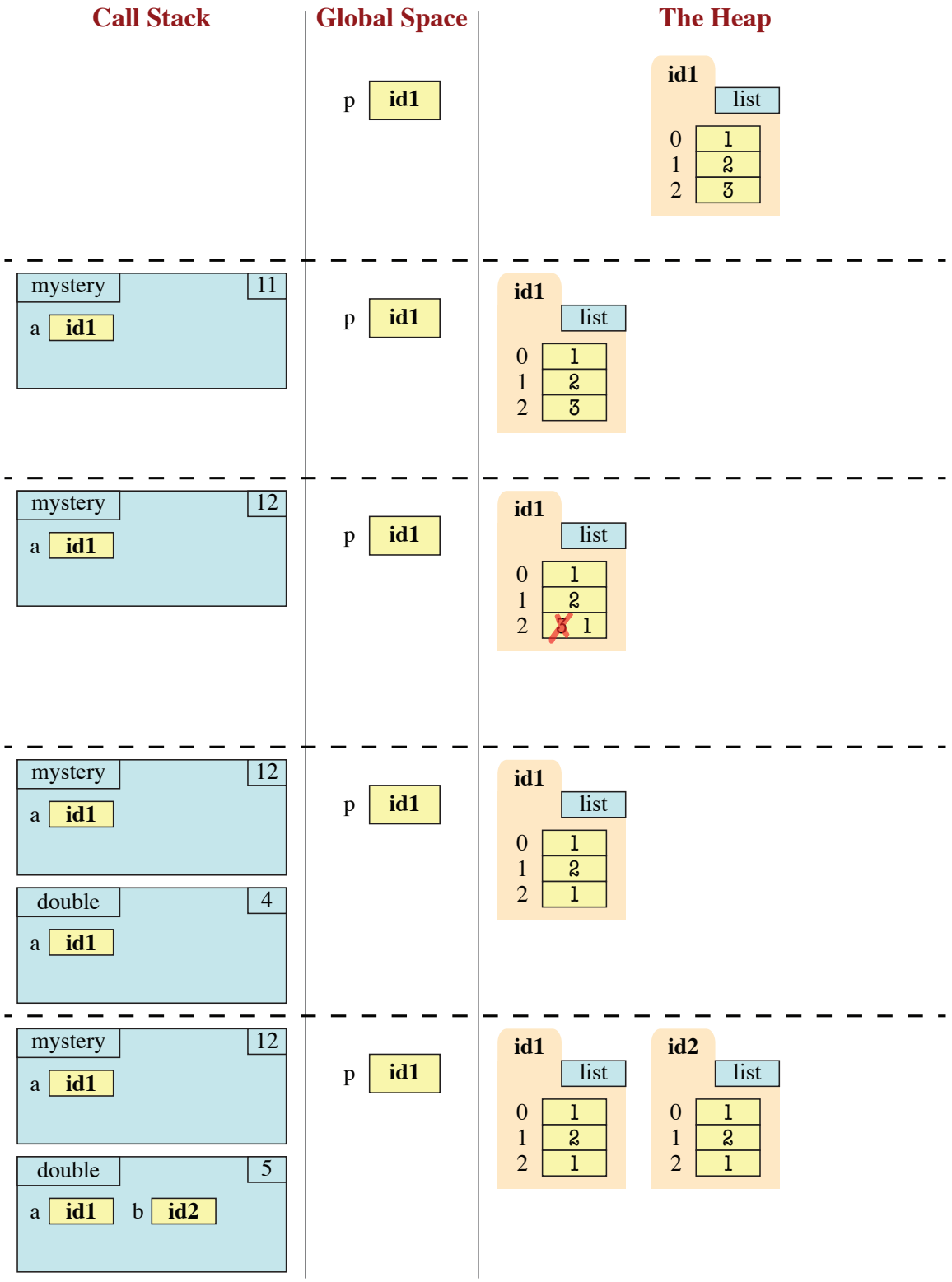
   Assume `p = [1, 2, 3]` is a global variable referencing a list in the heap, as shown on the next page. On the next two pages, diagram the evolution of the call

   ```
   q = mystery(p)
   ```

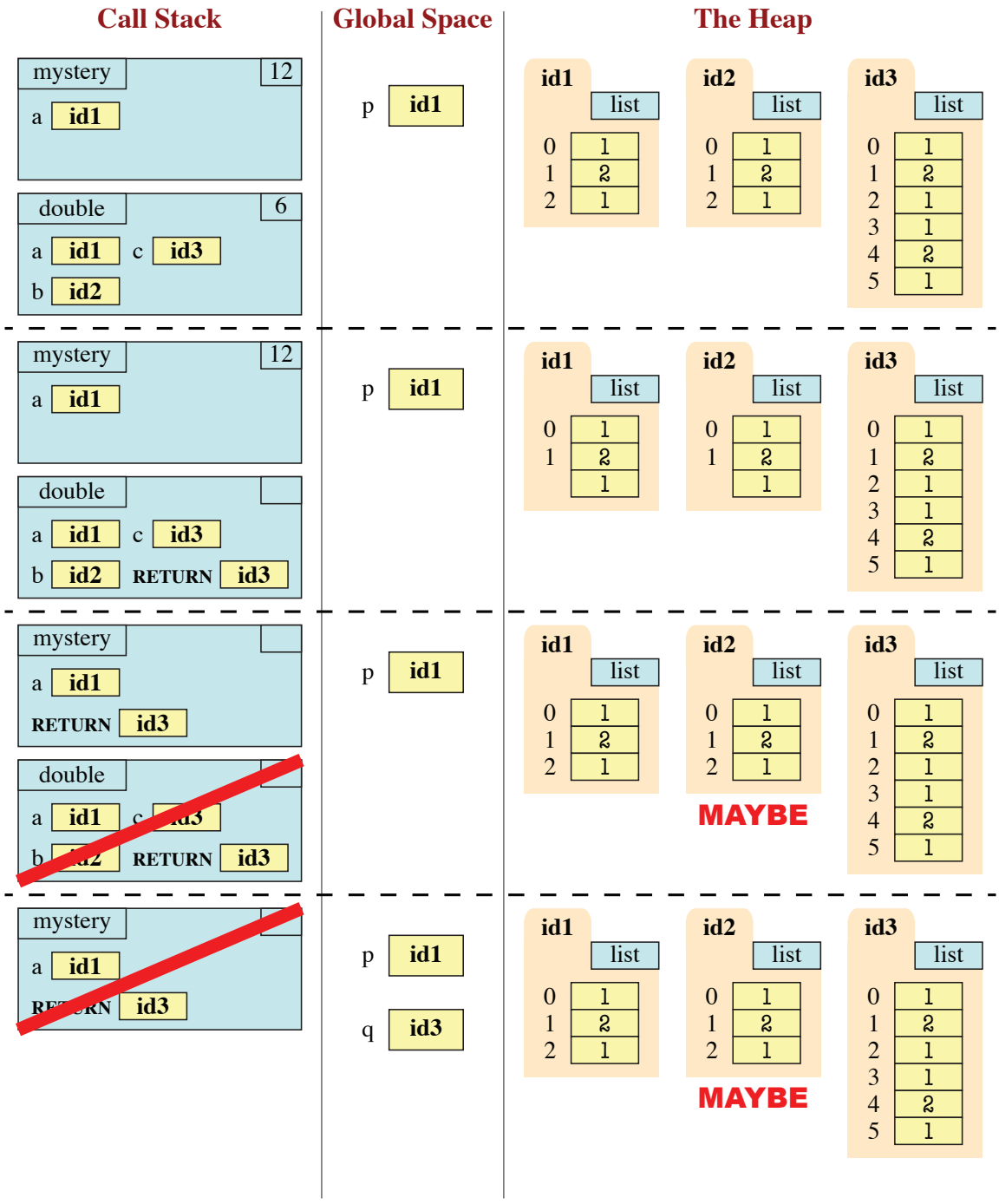
   Diagram the state of the *entire call stack* for the function `mystery` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **eight** diagrams, not including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

   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. You are also allowed to write "unchanged" if no changes were made to either global space or the heap.

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

4. [23 points total] **Testing and Debugging**.

   (a) [9 points] The function `expand` takes a date in the compact form `'9/21/21'` and expands it into the longer form `'Sep 21st, 2021'`. All months are abbreviated to three letters, but everything else is spelled out. Two digit years before 50 are assumed to be in the 21st century (20XX). Otherwise, the year is assumed to be in the 20th century (19XX). We will consider 2000 to be technically in the 21st century for this problem.

   There are ***at least* three bugs** in the code below. These bugs are potentially spread across multiple functions. We have added several print statements throughout the code, and shown the results on the next page. Using this information as a guide, identify and fix the three bugs on the next page. Remember that specifications are always correct, and any deviation between code and a specification is a bug. You **must** explain your fixes.

```
1  def expand(s):
2      """Returns english date expansion of s
3
4      Years before 50 are 20XX. Others are 19XX.
5
6      Precond: s a string for a VALID 'm/d/y'.
7      m and d are either one or two digits.
8      y is always two digits"""
9      div1 = s.find('/')
10     print('div1 is '+str(div1))     # WATCH
11     div2 = s.find('/',div1+1)
12     print('div2 is '+str(div2))     # WATCH
13     m = int(s[:div1])
14     print('m is '+str(m))           # WATCH
15     y = int(s[div2+1:])
16     print('y is '+str(y))           # WATCH
17
18     m = monthize(m)
19     print('month is '+str(m))       # WATCH
20     d = dayize(s[div1+1:div1+3])
21     print('day is '+str(d))         # WATCH
22     y = yearize(y)
23     print('year is '+str(y))        # WATCH
24
25     return m+' '+d+', '+y
26
27 def monthize(m):
28     """Returns the month abbrev for m
29
30     Precond: 1 <= m <= 12 is an int"""
31     # Combined TRACE and WATCH
32     print('monthize on '+str(m))
33     # The list is NOT a bug
34     items = ['Jan','Feb','Mar','Apr','May',
35              'Jun','Jul','Aug','Sep','Oct',
36              'Nov','Dec']
37     return items[m-1]
38
39
40
```

```
41 def dayize(s):
42     """Returns s with the correct suffix
43
44     Precond: s a str of one or two digits"""
45     # Combined TRACE and WATCH
46     print('dayize on '+s)
47     if len(s) == 2 and s[0] == '1':
48         print('teens day')         # TRACE
49         suff = 'th'
50     elif s[-1] == '1':
51         print('ends in 1')         # TRACE
52         suff = 'st'
53     elif s[-1] == 2:
54         print('ends in 2')         # TRACE
55         suff = 'nd'
56     elif s[-1] == '3':
57         print('ends in 3')         # TRACE
58         suff = 'rd'
59     else:
60         print('other ending')      # TRACE
61         suff = 'th'
62
63     return s+suff
64
65 def yearize(y):
66     """Returns 4 digit year for y (as str)
67
68     Years before 50 are 20XX. Others are 19XX.
69
70     Precond: 0 <= y <= 99 is an int"""
71     # Combined TRACE and WATCH
72     print('yearize on '+str(y))
73     if y < 50:
74         print('< 50')              # TRACE
75         pref = '20'
76     if y > 50:
77         print('> 50')              # TRACE
78         pref = '19'
79
80     return pref+str(y)
```

Tests:

```
>>> expand('9/22/19') # 'Sep 22nd, 2019'
div1 is 1
div2 is 4
m is 9
y is 19
monthize on 9
month is Sep
dayize on 22
other ending
day is 22th
yearize on 19
< 50
year is 2019
Sep 22th, 2019
```

```
>>> expand('12/7/41') # 'Dec 7th, 2041'
div1 is 2
div2 is 4
m is 12
y is 41
monthize on 12
month is Dec
dayize on 7/
other ending
day is 7/th
yearize on 41
< 50
year is 2041
Dec 7/th, 2041
```

```
>>> expand('12/15/50') # 'Dec 15th, 1950'
div1 is 2
div2 is 5
m is 12
y is 50
monthize on 12
month is Dec
dayize on 15
teens day
day is 15th
yearize on 50
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "debug.py", line 22, in expand
    y = yearize(y)
  File "debug.py", line 80, in yearize
    return pref+str(y)
UnboundLocalError: unknown variable 'pref'
```

First Bug:

This is a subtle type error that we put into the code by accident, and then left in as part of the exam. Note the `elif` statement on line 53

```
elif s[-1] == 2:
```

The expression `s[-1]` is a string, while `2` is an int. They can never be equal. The correct code is therefore

```
elif s[-1] == '2':
```

Second Bug:

This bug is *not* in `dayize`. Instead, it is a slicing error that violates the precondition of that function. On line 20, you have the code

```
d = dayize(s[div1+1:div1+3])
```

This only works if the day is represented by two digits. The correct code is

```
d = dayize(s[div1+1:div2])
```

Third Bug:

This bug is caused because the variable `pref` is not assigned a value. That is because neither of the two if-statements in `yearize` are executed. The correct thing to do is to take this if-statement

```
if y > 50:
```

and replace it with `else:`

(b) [10 points] Consider the following function specification:

```
def replace(s,a,b):
    """Returns the string s with all instances of character a replaced by b.

    Example: replace('banana', 'n', 't') returns 'batata'

    Precond: s, a, b are all strings with no characters other than lowercase
    letters. a is exactly one letter."""
```

Do not implement this function. Instead, we want you to write at least **six test cases** below. 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, you should explain why it is substantially different from the others.

There are many answers to this question. Here are some of the ones we had in mind.

Input	Output	Reason
s='', a='a', b='b'	''	String s is empty
s='abc', a='x', b='y'	'abc'	Nothing is replaced
s='abc', a='a', b=''	'bc'	String b is empty (deletion)
s='abc', a='a', b='x'	'xbc'	Normal replacement
s='abba', a='a', b='x'	'xbbx'	Repeated replacement
s='abc', a='a', b='xy'	'xybc'	String b is multiple characters

(c) [4 points] **Do not implement the function specified below.** Instead, use assert statements to enforce the precondition. You do *not* need to provide error messages.

```
def isnetid(s):
    """Returns True if the user identifier s is a registered netid

    Precond: s is a nonempty string of only letters and digits. All letters
    must be lowercase. The string s cannot start with a digit."""

    assert type(s) == str
    assert s.isalnum() and s.islower()
    assert len(s) > 0
    assert not s[0].isdigit()
```

Last Name: _____ First: _____ Netid: _____

5. [25 points] **String Slicing**.

Implement the function below. You **may not use a for-loop to implement this function** (and a for-loop is not necessary). Simply use the functions and methods provided on the reference page. Pay close attention to the examples to better understand the function.

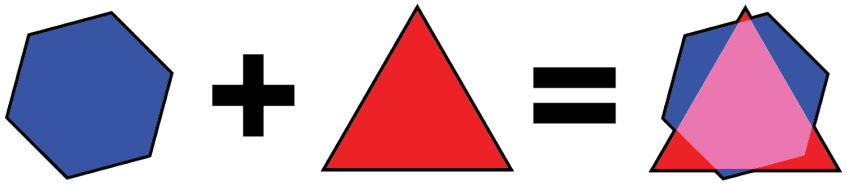
```
def initials(name):
    """Returns (up to three) initials for the given name

    A word is a (sub)string composed of only letters. A name is a sequence of words
    separated by (single) spaces. The initials of a name are the first letters of
    each word. If there are more than three words, this function only chooses the
    initials of the first two words, and the last word. The initials returned are
    all lower case.

    Examples: initials('Walker') returns 'w'
              initials('Walker White') returns 'ww'
              initials('Walker McMillan White') returns 'wmw'
              initials('Daniel Michael Blake Day Lewis') returns 'dml'

    Precond: name is a string of letters and spaces. There are no adjacent
    spaces. The name does not begin or end with a space."""

    # Get the first initial
    first = name[0]

    # Get position of the first space (if any)
    pos = name.find(' ')
    if (pos == -1):
        # There are no other words for initials
        middle = ''
        last = ''
    elif name.count(' ') > 1:
        # There is (at least one) middle name
        middle = name[pos+1]
        # Find the last name
        pos = name.rfind(' ')
        last = name[pos+1]
    else:
        # There is no middle name
        middle = ''
        last = name[pos+1]

    # Glue together and convert to lower case
    initials = first+middle+last
    initials = initials.lower()
    return initials
```

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. `RGB` has a fourth attribute: `alpha`, which has the same restrictions. This means that when you create an `RGB` object, the proper constructor call is `RGB(r,g,b,a)` (do not worry about `introcs` for this question). We ignored this attribute in Assignment 3, but we will use it here.

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) [8 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 colors color1 and color2, storing 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 the same
    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)

    # No return value
```

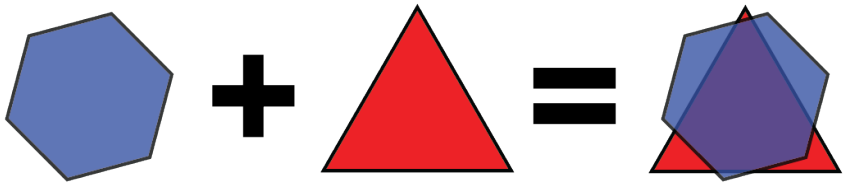
(b) [14 points] A more popular form of color composition is alpha-blending. The attribute `alpha` represents the amount of transparency of the color on top. An `alpha` of 255 means the color is completely opaque, while an `alpha` of 0 is completely transparent.

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:

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

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, rounding to the nearest integer.

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



```
def alpha_blend(color1,color2):
    """Returns a new color blending (via the formula above) color1 and color2.

    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, remembering to round
    red = int(round(red*255))
    green = int(round(green*255))
    blue = int(round(blue*255))
    alpha = int(round(alpha*255))

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