Last Name:		First:		Netid:		Section	

CS 1110 Prelim 1 October 16th, 2014

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).

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.

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

(a) [2 points] What are the four basic, immutable types in Python?

`int`, `float`, `bool`, and `str`.

(b) [4 points] Consider the two assignment statements

```
>>> a = 1/2
>>> b = 1.0/2
```

What are the values in the variables `a` and `b`? Explain your answers.

That variable `a` contains `0`, while `b` contains `0.5`. The expression assigned to `a` uses `int` division, which always drops the remainder.

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

A *parameter* is a variable in the parentheses at the start of a function definition.
An *argument* is an expression in the parentheses of a function call.
A function call evaluates the arguments and plugs the result into the parameters before executing the function body.

(d) [4 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.

(e) [4 points] The variable `d` contains a reference to a dictionary. Fill in the missing lines so that Python (in interactive mode) gives you the output shown below.

```
>>> d
{ 'a':1, 'b':3 }

>>> d['a'] = 2

>>> d['c'] = 4

>>> d

{ 'a':2, 'b':3, 'c':4 }
```

3. [18 points] **String Slicing**.

For this question, you may find the following functions and methods may be 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).
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.islower()	**Returns**: True if s is *not empty* and its elements are all lowercase letters; it returns False otherwise.
s.isupper()	**Returns**: True if s is *not empty* and its elements are all uppercase letters; it returns False otherwise.

Recall that a Cornell netid is a string with either 2 or 3 letters (case does not matter), followed by a number. Use this to implement the function below.

```
def isnetid(s):
    """Return: True if s is a valid netid; False otherwise

    A valid netid is 2 or 3 letters followed by a number.

    Examples: 'wmw2' is a valid netid, but 'wmw2a' and 'w2' are not.

    Precondition: s is string.
    # Make sure it is long enough
    if len(s) < 3:
        return False

    # See if it is 2 or 3 letters
    if s[2].isdigit():
        pos = 2
    else:
        pos = 3

    # Check each half and return
    prefix = s[:pos].isalpha()
    suffix = s[pos:].isdigit()
    return prefix and suffix
```

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

Consider the following function specification.

```
def replace(s,org,rep):
    """Return: the string s with all instances of character org replaced by rep.
    Example: replace('banana', 'n', 't') returns 'batata'
    Precondition: s, org, rep are all strings whose characters are lowercase letters.
    org is exactly one letter. s and rep can be empty"""
```

Do not implement this function. Use the specification to answer the questions below.

(a) [6 points] Write one or more assert statements to enforce the preconditions of the function replace. Your assert statements **do not** need to have error messages.
Hint: You may want to look at the table on the previous page.

```
assert type(s) == str # Check type
assert type(org) == str and type(rep) == str # Check types
assert len(s) == 0 or s.islower() # Empty strings are special
assert len(rep) == 0 or rep.islower() # Empty strings are special
assert len(org) == 1 and org.islower() # a is one character
```

(b) [12 points] In the space below,provide at **least six** different test cases to verify that the replace 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.

When you make test cases, you should look very closely at the preconditions. This is how we determine whether or not the test cases are "different enough". Below are the different solutions we were thinking of. If you had (at least) six test cases that matched 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='', org='a', rep='b'	''	String s is empty
s='abc', org='x', rep='y'	'abc'	Nothing is replaced
s='abc', org='a', rep=''	'bc'	String b is empty (deletion)
s='abc', org='a', rep='x'	'xbc'	Normal replacement
s='abba', org='a', rep='x'	'xbbx'	Repeated replacement
s='abc', org='a', rep='xy'	'xybc'	String b is multiple characters
s='abc', org='a', rep='a'	'xybc'	Replacement leaves string unchanged

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

The functions `sum_front` and `sum_back` take a list as input and sum either the first two or last two elements, respectively. They are defined below.

```
def sum_front(p):
    """Return: sum of first 2 items
    Pre: p is a list, len(p) >= 2"""
1   return p[0]+p[1]
```

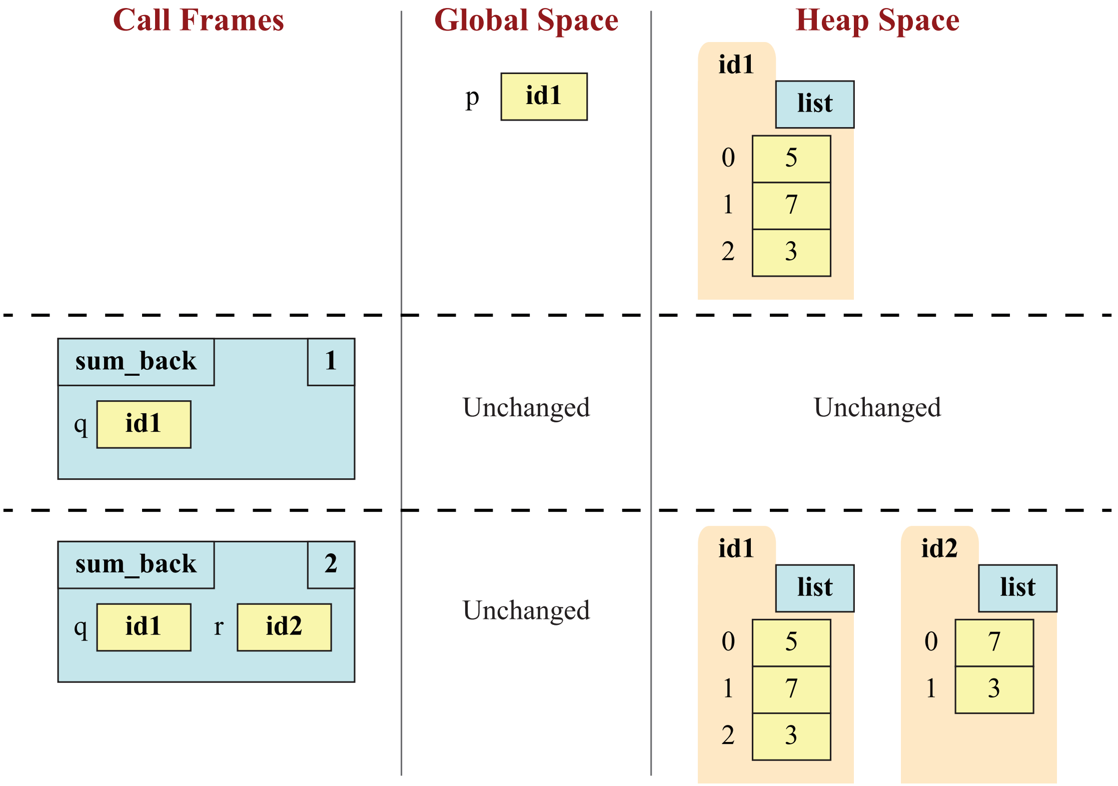
```
def sum_back(q):
    """Return: sum of last 2 items
    Pre: q is a list, len(q) >= 2"""
1   r = q[len(q)-2:]
2   result = sum_front(r)
3   return result
```

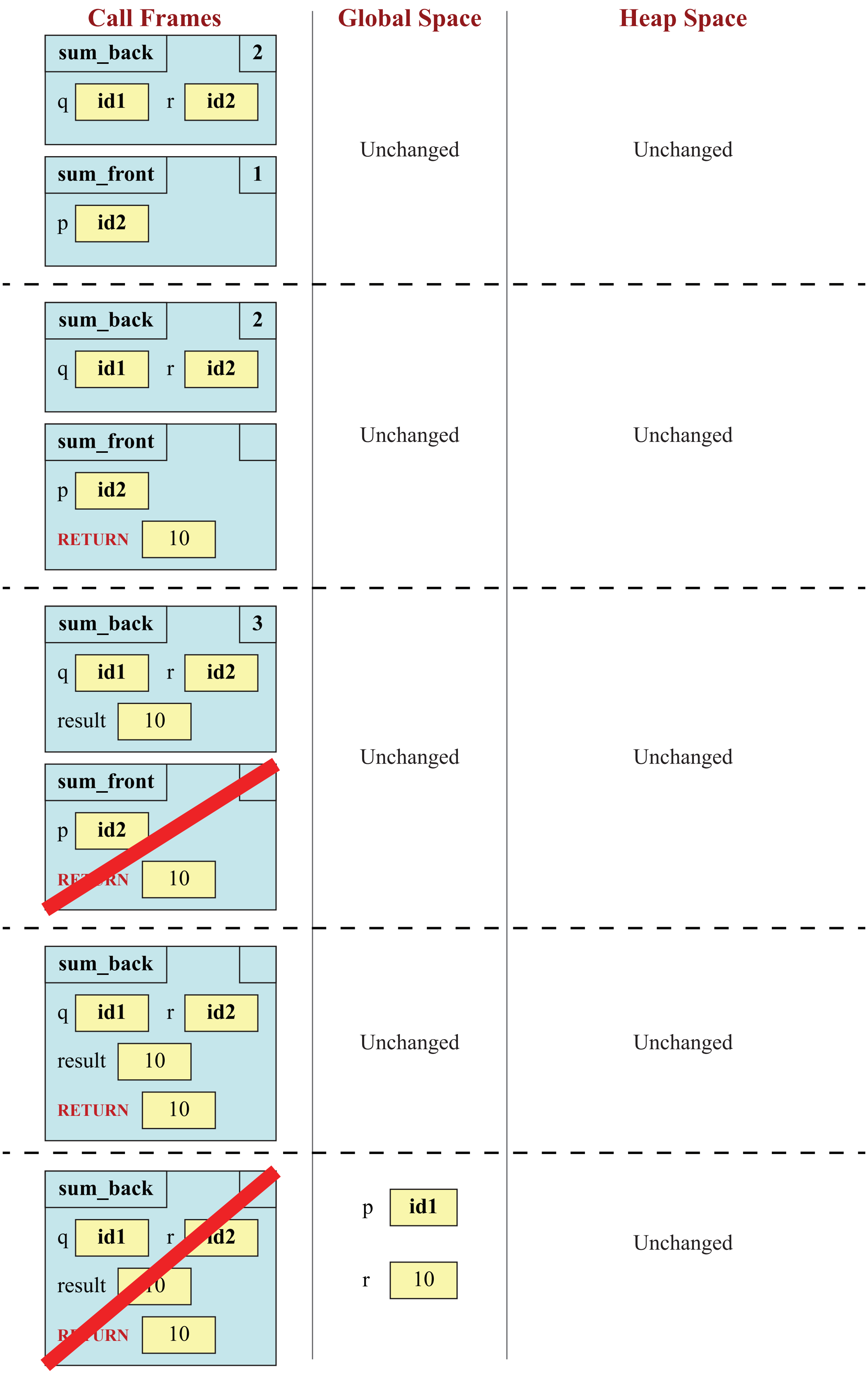
Assume that `p = [5,7,3]` is a global variable that stores a reference to a list in heap space, as shown below. On this page and the next, diagram the evolution of the call

```
r = sum_back(p)
```

You need to diagram the state of the call stack when this function starts, for each line executed, and when the frame is erased. In addition, since `sum_back` calls `sum_front` as a helper function, you also need to diagram how this function affects the call stack (start of the function, each line executed, and frame erased). We have already begun the diagramming for you below, with the start of the function `sum_back`. There are **six** more diagrams to draw.

In each diagram, you should draw the state of global space and heap space in addition to the call stack. However, to reduce the amount that you have to draw, you are allowed to write "unchanged" if no changes were made to either global space or heap space. We have done that in the first diagram (the start of `sum_back`) below. If there are changes, redraw everything.




Call Frames
Global Space
Heap Space
sum_back 2
q id1 r id2
sum_front 1
p id2
Unchanged
Unchanged
sum_back 2
q id1 r id2
sum_front
p id2
RETURN 10
Unchanged
Unchanged
sum_back 3
q id1 r id2
result 10
sum_front
p id2
RETURN 10
Unchanged
Unchanged
sum_back
q id1 r id2
result 10
RETURN 10
Unchanged
Unchanged
sum_back
q id1 r id2
result 10
RETURN 10
p id1
r 10
Unchanged

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

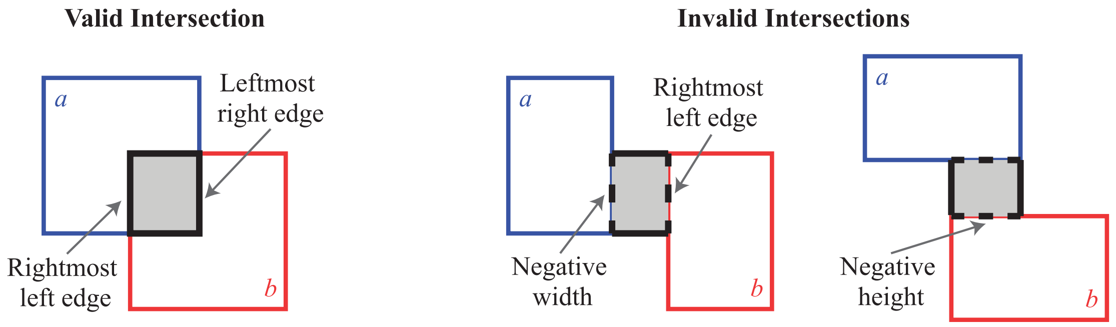
Rectangle objects are very common in computer graphics; they are used to indicate a region of pixels on your screen. Rectangles have four attributes with the following invariants.

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 stored as integers. As you saw in one of the labs, y-coordinates in pixel space 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 its 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.

One of the most important things to do with Rectangles is intersect them. The intersection of two overlaping rectangles is also a Rectangle. As you can see from the picture below, you take the rightmost left edge of the two rectangles to get the left edge of the intersection. Similarly, you take the leftmost right edge, the topmost bottom edge, and the bottom-most top edge.



If the two rectangles do not overlap, this process will produce an intersection that has either negative width or negative height. This violates the invariant, so we say the intersection is invalid in this case. This is illustrated above on the right.

Using the illustration as a guide, implement the function specified on the next page. All the information that you need is on this page. Remember that the `min` and `max` functions are built into Python. You should be able to implement the function without additional helpers, but you can write helper functions if you wish.

```
def intersection(r1,r2):
    """Return: the rectangle that is the intersection of r1 and r2.
    If the intersection is invalid, return False instead.

    Precondition: r1 and r2 are Rectangle objects."""

    # Get the right and bottom edge of each
    right1 = r1.x+r1.width
    bottom1 = r1.y+r1.height
    right2 = r2.x+r2.width
    bottom2 = r2.y+r2.height

    # Find edges of intersection
    left = max(r1.x,r2.x)
    right = min(right1,right2)
    top = max(r1.y,r2.y)
    bottom = min(bottom1,bottom2)

    # Compute width/height to make intersection
    width = right-left
    height = bottom-top

    # Return the right value
    if width < 0 or height < 0:
        return False

    return Rectangle(left,top,width,height)
```