Last Name: ____________ First: ____________ Netid: ____________

CS 1110 Prelim 1 October 6th, 2022

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	12	
3	21	
4	21	
5	21	
6	23	
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

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`.

2. [12 points total] **Short Answer Questions**.

   (a) [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.

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

```
>>> a = 3//2
>>> b = 3.0//2
```

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

   That variable `a` contains 1 because `//` is integer division (computing the number of times 2 goes *evenly* into 3).

   The variable `b` contains 1.0. While the meaning of the operation `//` is the same, 3.0 is a `float` and Python must either convert 2 to a `float` or 3.0 to an `int` to get the two to match. Python will chose to convert the 2 to 2.0, resulting in a `float` for the final answer.

   (c) [4 points] What is the difference between a *mutable* and an *immutable* type? Give an example of each shown in class.

   An *mutable* type is one where we can where the value are objects represented by folders, and we can change the contents of that folder. `RGB` and `Point3` are examples of mutable objects. As are lists (since all values in Python are objects).

   An immutable type is one where we do not use folders to represent the values, since we cannot change the contents of these folders Any of the basic types – `int`, `float`, `bool`, or `str` – are immutable types.

3. [21 points total] **Objects and Functions**.

   One of the most popular classes in Python is the `Date` class, which represents a month, day, and year. For this problem, all objects of this class have the following attributes:

Attribute	**Meaning**	**Invariant**
`month`	the month as number	`int` value between 1 and 12 (inclusive)
`day`	the day of the month	`int` value between 1 and `d.daysInMonth()` (inclusive)
`year`	the year	`int` value greater than 0

   In addition, all `Date` objects have the following method:

Method	**Description**
`d.daysInMonth()`	**Returns**: number of days in the current month of d.

   To create a new `Date` object, the constructor function is `Date(month,day,year)`.

Last Name: ______________ First: ______________ Netid: ______________

(a) [6 points] Implement the function below according to the specification.

```
def last_day(date):
    """Returns the date of the last day of the current month

    Example: if d is Date(2,12,2000), last_day(d) returns Date(2,29,2000)

    Precondition: date is a Date object"""

    m = date.month
    d = date.daysInMonth()
    y = date.year

    return Date(m,d,y)
```

(b) [15 points] Implement the function below according to the specification.

```
def next_day(date):
    """MODIFIES date to be the next calendar day

    Example: if d is Date(2,12,2000), next_day(d) modifies d to Date(2,13,2000)
             if d is Date(2,29,2000), next_day(d) modifies d to Date(3,1,2000)
             if d is Date(12,31,2000), next_day(d) modifies d to Date(1,1,2001)

    Precondition: date is a Date object"""

    if date.day == date.days_in_month():
        date.day = 1

        # We need to be careful with month invariant
        month = date.month + 1
        if month == 13:
            month = 1
            date.year += 1

        # Now it is safe to update the month
        date.month = month

    else:
        # Not the end of the month
        date.day += 1

    # Procedure does not have a return
```

Last Name: ______________ First: ______________ Netid: ______________

4. [21 points] **Call Frames.**

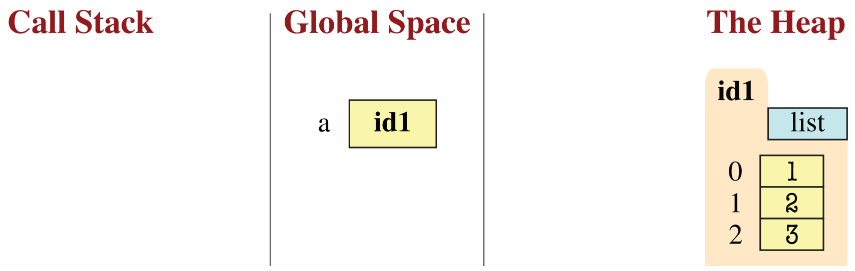
Consider the following function definitions.

```
def mixup(a):
    """Returns a mixed up copy of a
    Pre: a is a nonempty list of ints"""
    b = a[:1]
    c = merge(b,a[1:])
    return c
```

```
def merge(a,b):
    """Returns merging of a, b
    Pre: a, b are nonempty lists of ints"""
    b[a[0]] = a[0]
    return b+a

```

Assume `a = [1, 2, 3]` is a global variable referencing a list in the heap, as shown below.



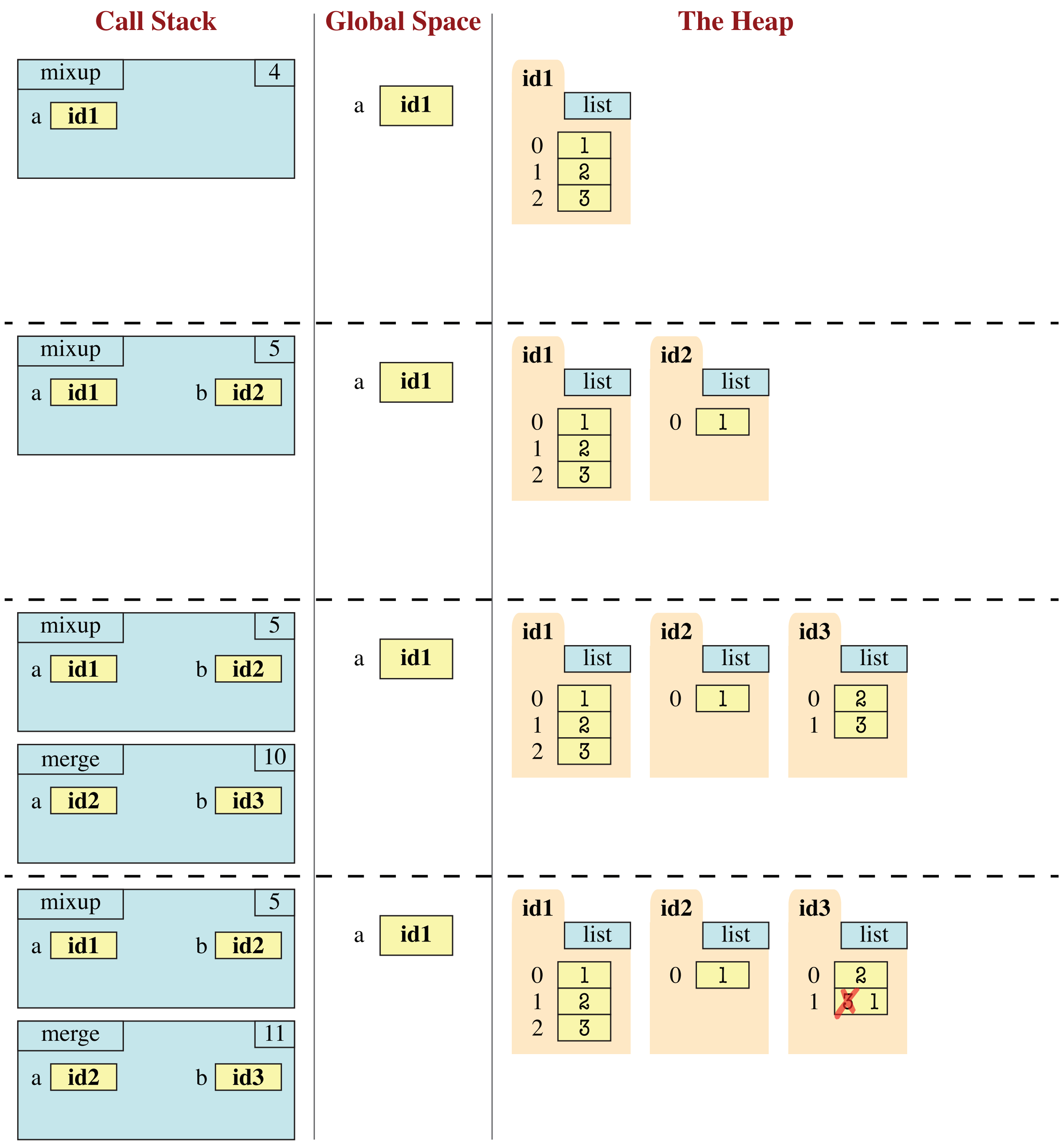
On the next two pages, diagram the evolution of the call

```
a = mixup(a)
```

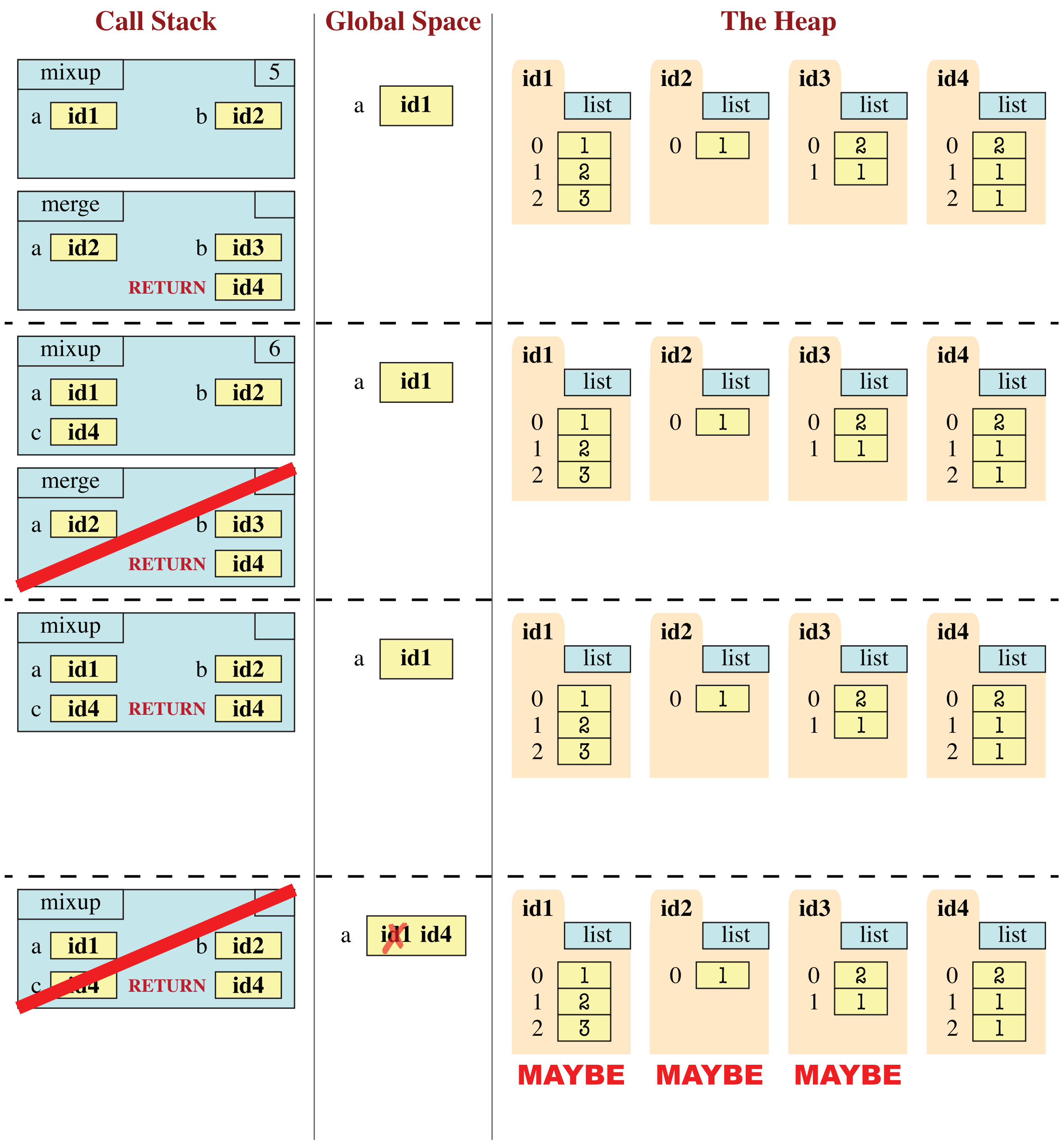
Diagram the state of the *entire call stack* for the function `mixup` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **eight** diagrams, not including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

5. [21 points total] **Testing and Debugging**.

   (a) [9 points] The function `ordinalize` is very similar to the function `anglicize` demonstrated in class. Instead of the normal word for a number, it produces the *ordinal* word. So 1 becomes `'first'`, 23 becomes `'twenty third'`, and so on.

   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.

```python
def ordinalize(n):
    """Returns: Ordinal word for n.

    Precondition: n an int in 0..99"""

    # Small numbers are simple
    if n < 20:
        print('If < 20')            # TRACE
        word = ord1to19(n)
        print('word: '+word)        # WATCH

    # Big numbers may need 2 words
    if n > 10:
        # Get the first word
        print('If > 10')            # TRACE
        part1 = tens(n // 10)
        print('part1: '+part1)      # WATCH

        # Check if we need 2nd word
        if n % 10 == 0:
            print('No 2nd word')    # TRACE
            part1 = ithify(part1)
            print('part1: '+part1) # WATCH
        else:
            print('Has 2nd word')   # TRACE
            part2 = ' '+ord1to19(n % 10)
            print('part2: '+part2) # WATCH

        word = part1+part2
        print('word: '+word)        # WATCH

    return word


def ithify(word):
    """Returns word with 'ieth' at end

    Precondition: word ends in 'y'"""
    print('ithify: '+word)          # WATCH
    result = word[:-1]+'ieth'
    print('result: '+result)        # WATCH
    return result

def tens(n):
    """Returns: tens-word for n

    Precondition: n an int in 2..9"""
    print('tens: '+str(n))          # WATCH
    words = ['twenty','thirty','forty','fifty',
             'sixty','eighty','ninety']
    result = words[n-2]
    print('result: '+result)        # WATCH
    return result

def ord1to19(n):
    """Returns: Ordinal word for n.

    Precondition: n an int in 1..19"""
    print('ord1to19: '+str(n))      # WATCH
    words = ['first','second','third','fourth',
             'fifth','sixth','seventh','eighth',
             'ninth','tenth','eleventh','twelfth',
             'thirteenth','fourteenth','fifteenth',
             'sixteenth','seventeenth',
             'eighteenth','nineteenth'']
    result = words[n-1]
    print('result: '+result)        # WATCH
    return result
```

Tests:

```
>>> ordinalize(12) # 'twelfth'
If < 20
ord1to19: 12
result: twelfth
word: twelfth
If > 10
tens: 1
result: ninety
part1: ninety
Has 2nd word
ord1to19: 2
result: second
part2:  second
word: ninety second
'ninety second'
```

First Bug:

While the first if-statement on line 7 is correct, the code has made a mistake in the second if-statement on line 13. As a result, both if-statements are executed and the result of the second one overwrites the first. While there are many solution to this problem, the simplest one is to replace line 13 with

```
else:
```

Indeed, this problem shows exactly why we like to use `else` and `elif` instead of separate if-statements.

```
>>> ordinalize(30) # 'thirtieth'
If > 10
tens: 3
result: thirty
part1: thirty
No 2nd word
ithify: thirty
result: thirtieth
part1: thirtieth
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "debug.py", line 29, in ordinalize
    word = part1+part2
UnboundError: 'part2' ref before assignment
```

Second Bug:

The variable `part2` is created on line 26. However, that assignment is only executed when the expression on line 20 is false. From the trace we can see that this did not happen, so the variable was never created. We need to change the code to make sure that we assign a value to `part2` no matter what the results are of that if-statement. Again, there are many possible fixes, but the simplest solution is to add the following line just before either line 20 or line 23

```
part2 = ''
```

```
>>> ordinalize(99) # 'ninty ninth'
If > 10
tens: 9
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "debug.py", line 16, in ordinalize
    part1 = tens(n // 10)
  File "debug.py", line 52, in tens
    result = words[n-2]
IndexError: list index out of range
```

Third Bug:

This bug is caused because the value `n-2` is 7 while the list defined on lines 50 and 51 only contains seven elements (remember that list indices start at 0). As the precondition of the function is satisfied, this is clearly a problem with `tens`. Looking closely at the list we notice that we forgot the word `'seventy'`. Hence the correct thing to do is change lines 50 and 51 to

```
words = ['twenty','thirty','forty','fifty',
         'sixty','seventy','eighty','ninety']
```

(b) [8 points] Consider the following function specification:

```
def unique(lst):
    """Returns: The number of unique elements in the list.

    Example: unique([1, 5, 5]) returns 2.

    Precondition: lst is a list. Any value in lst must be an int."""
```

Do not implement this function. Instead, we want you to write at least **four 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.

As with all test cases, we look at the preconditions to determine "different enough". There are **a lot** of possible answers for this problem. Below were four we were thinking of.

Input	**Output**	**Reason**
`lst=[]`	`0`	Empty list
`lst=[1,2,3]`	`3`	No duplicates
`lst=[1,2,2]`	`2`	One set of duplicates
`lst=[1,2,2,3,3]`	`3`	Multiple sets of duplicates

(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 xpand(s):
    """Returns a copy of s with all 'x' replaced by 'xxx'

    Example: xpand('extra') returns 'exxxtra'

    Precond: s is a string with at least one 'x'.
    There are no adjacent 'x' characters in s."""

    assert type(s) == str
    assert 'x' in s
    assert not 'xx' in s
```

6. [23 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 replace_ab(s, a, b, c):
    """Returns copy of s with the FIRST occurrence of a or b replaced by c

    Only the substring (a or b) that occurs FIRST will be replaced by c. If
    neither a nor b is a substring of s, then the string is unchanged.

    Examples: replace_ab('abba', 'a', 'b', 'c') returns 'cbba'
              replace_ab('abba', 'b', 'a', 'c') returns 'cbba'
              replace_ab('acaba', 'ca', 'ab', 'd') returns 'adba'
              replace_ab('adda', 'a', 'b', 'c') returns 'cdda'
              replace_ab('xyz', 'a', 'b', 'c') returns 'xyz'

    Preconditions: s, a, b, and c are all strings of lowercase letters.
    a and b are nonempty and start with different letters."""

    # Find the positions
    aloc = s.find(a)
    bloc = s.find(b)

    # If neither is there, stop
    if aloc == -1 and bloc == -1:
        return s
    elif aloc == -1:     # Replace b
        # Find the ends of b to cut out
        pos1 = bloc
        pos2 = bloc+len(b)
    elif bloc == -1:     # Replace a
        # Find the ends of a to cut out
        pos1 = aloc
        pos2 = aloc+len(a)
    elif aloc < bloc:    # Replace a
        # Find the ends of a to cut out
        pos1 = aloc
        pos2 = aloc+len(a)
    else:                # Replace b
        # Find the ends of b to cut out
        pos1 = bloc
        pos2 = bloc+len(b)

    # Glue c into the correct position
    return s[:pos1]+c+s[pos2:]
```