CS 1110

Prelim 2 Review
Spring 2017

Exam Info

- Prelim 2: 7:30–9:00PM, Tuesday, April 25th
  - aa200 – jjm200 Baker Laboratory 200
  - jjm201 – sge200 Rockefeller 201
  - sge201 – zz200 Rockefeller 203
- Baker Lab 200, Rockefeller Hall 201, 203
- No Electronics, No Notes, Closed book.
- Bring your Cornell ID
- Put your Name & NetId on Each Page!!!

What is on the Exam?

- The big topics:
  - Nested Lists & Dictionaries (A3, Lab 8)
  - Recursion (A4, Lab 9)
  - Defining classes (Lab 10, Lab 11, A4)
  - Inheritance and subclasses (Lab 11)
  - Name Resolution
  - While Loops & Invariants

What is on the Exam?

- The big topics:
  - Nested Lists & Dictionaries (A3, Lab 8)
  - Recursion (A4, Lab 9)
  - Defining classes (Lab 10, Lab 11, A4)
  - Inheritance and subclasses (Lab 11)
  - Name Resolution
  - While Loops & Invariants

Nested Lists

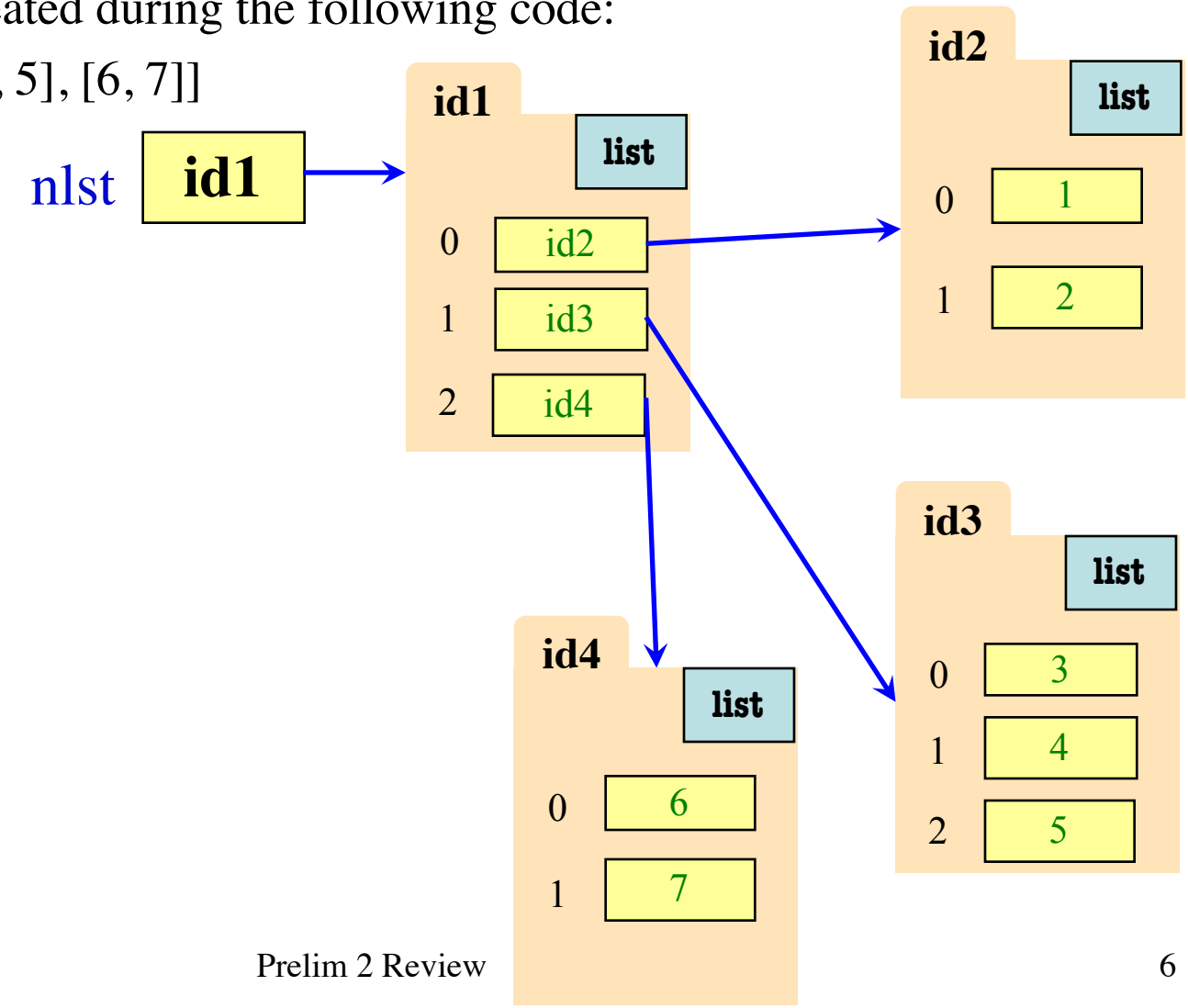
Diagram the objects created during the following code:

```
>>> nlst = [[1, 2], [3, 4, 5], [6, 7]]
>>> slice = nlst[1:]
>>> slice[1].append(0)
```

Nested Lists

Diagram the objects created during the following code:

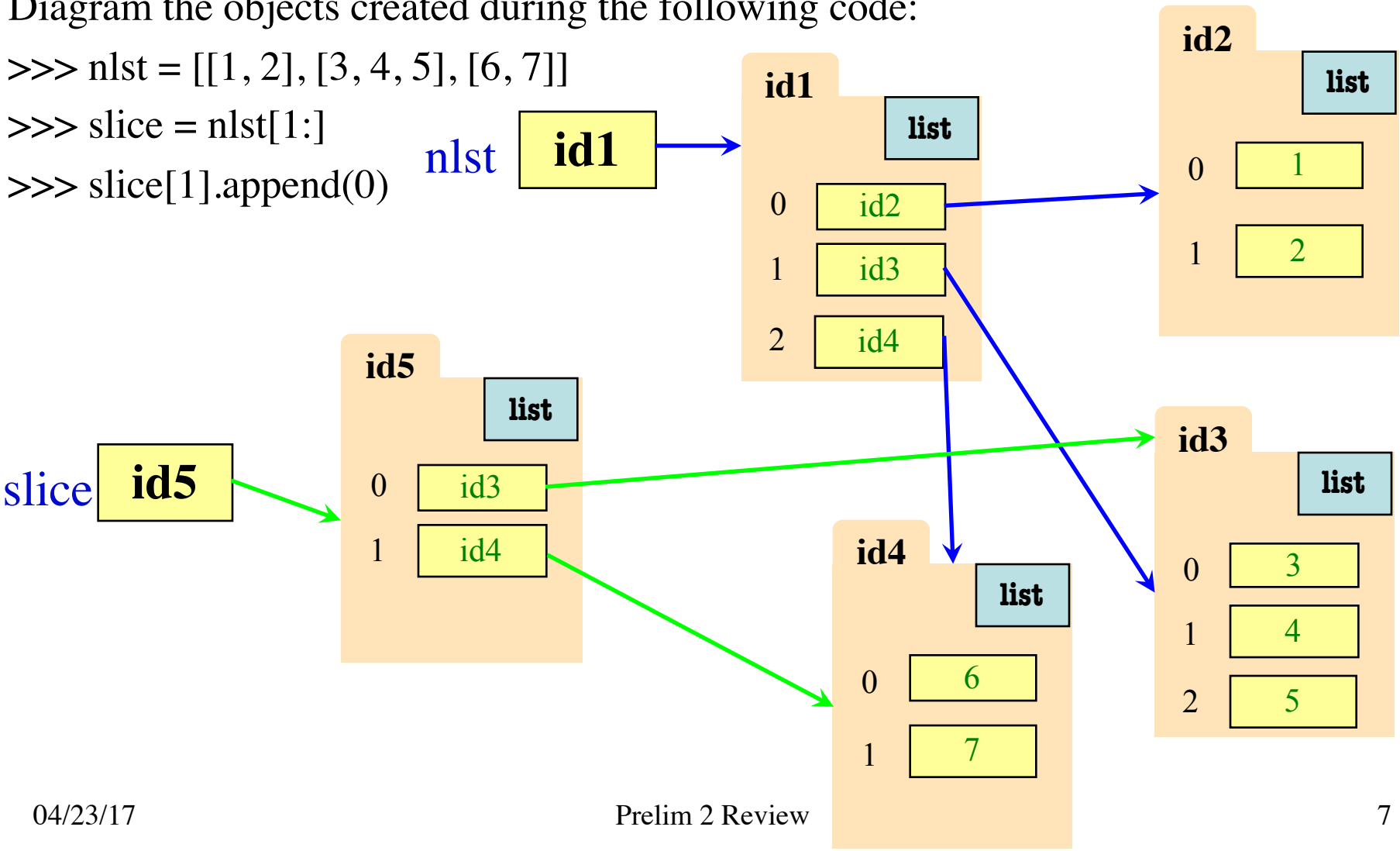
```
>>> nlst = [[1, 2], [3, 4, 5], [6, 7]]
>>> slice = nlst[1:]
>>> slice[1].append(0)
```



Nested Lists

Diagram the objects created during the following code:

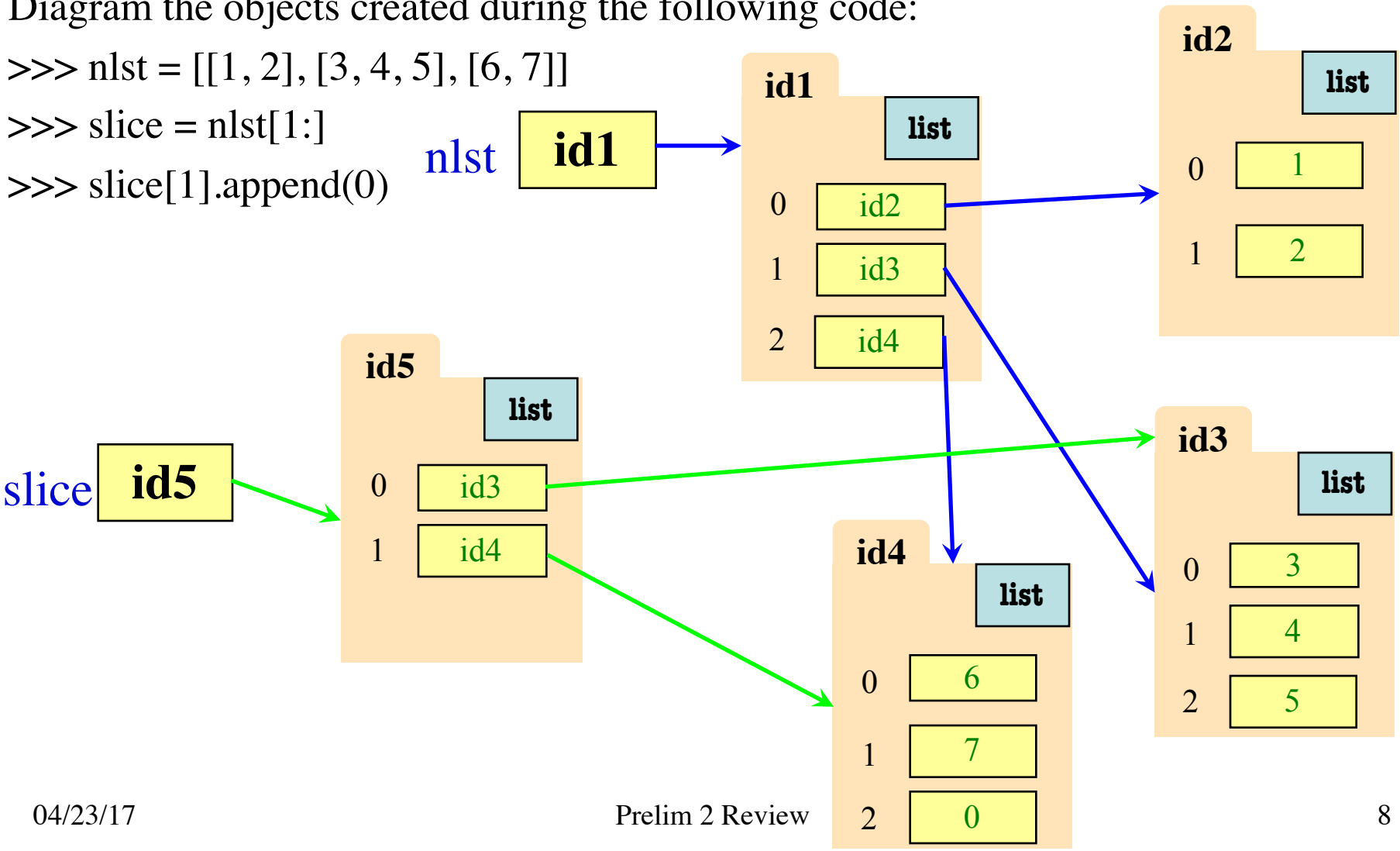
```
>>> nlst = [[1, 2], [3, 4, 5], [6, 7]]
>>> slice = nlst[1:]
>>> slice[1].append(0)
```



Nested Lists

Diagram the objects created during the following code:

```
>>> nlst = [[1, 2], [3, 4, 5], [6, 7]]
>>> slice = nlst[1:]
>>> slice[1].append(0)
```



Function with 2D Lists

```
def max_cols(table):
    """Returns: Row with max value of each column

    We assume that table is a 2D list of floats (so it is a list of rows and
    each row has the same number of columns.  This function returns
    a new list that stores the maximum value of each column.

    Examples:
        max_cols([ [1,2,3], [2,0,4], [0,5,2] ]) is [2,5,4]
        max_cols([ [1,2,3] ]) is [1,2,3]

    Precondition: table is a NONEMPTY 2D list of floats"""
```

Function with 2D Lists

```
def max_cols(table):
    """Returns: Row with max value of each column
    Precondition: table is a NONEMPTY  2D list of floats"""
    # Use the fact that table is not empty
    result = table[0][:] # Make a copy, do not modify table.
    # Loop through rows, then loop through columns
    for row in table:
        for k in range(len(row))
            if row[k] > result[k]
                result[k] = row[k]
    return result
```

Dictionaries

- Key-value pairs, unique keys
- Creation: dic = {'a': 1, 'b': 2, 'c': 3}
- Access: dic['a']
- Modification: dic['a'] = 5
- Add new key: dic['d'] = 7
- Delete key: del dic['c']
- Does not have a specific order! Not indexable

What is on the Exam?

- The big topics:
  - Nested Lists & Dictionaries (A3, Lab 8)
  - Recursion (A4, Lab 9)
  - Defining classes (Lab 10, Lab 11, A4)
  - Inheritance and subclasses (Lab 11)
  - Name Resolution
  - While Loops & Invariants

Recursion

- What kind of questions might be asked?
  - Will be given a function specification
  - Implement it using recursion
  - May have an associated call stack question
- Divide and Conquer
  - Base case
    - Decide what to do on “small” data
  - Recursive case
    - Decide how to break up your data into smaller pieces
  - Decide how to combine your answers

Recursion with nested lists

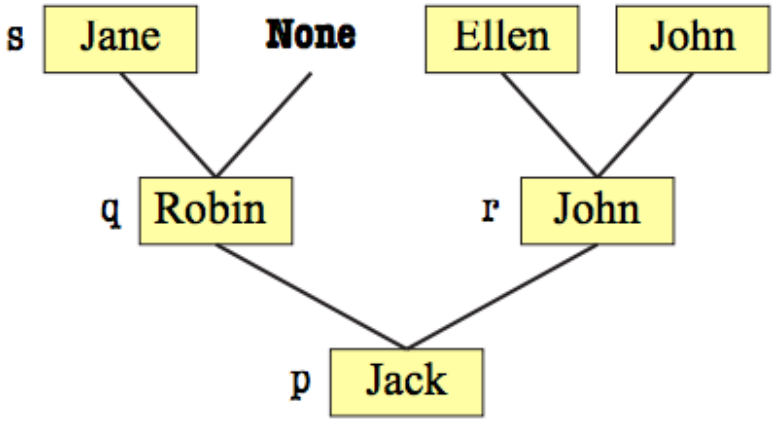
```
def flatten(lst):
    """Return: a COPY of the flattened version of the list lst.

    lst is a potentially nested list. A flattened version of lst means to take the nested
    list and turn it into a one-dimentional list.

    Example: flatten([]) returns [],
             flatten([[1], 2, 3]) returns [1, 2, 3]
             flatten([1, [2, 3], [[4], [[]], 5, [6, 7, 8]], 9]) returns [1, 2, 3, 4, 5, 6, 7, 8, 9]

    Precondition: lst is a list or an int"""
```

Recursion with nested lists

```
def flatten(lst):
    """Return: a COPY of the flattened version of the list lst
    Precondition: lst is a list or an int"""

    if type(lst) == int:
        return [lst]
    if lst == []:
        return []

    left = flatten(lst[0])
    right = flatten(lst[1:])

    return left + right
```

Recursion with objects (Modified FA16)

```
class Person(object):
    """Instance is a person/family tree
    INSTANCE ATTRIBUTES:
        name: First name [nonempty str]
        mom: Mom's side [Person or None]
        dad: Dad's side [Person or None]
    """

    ...
```



To make person s in the right picture, you do s = Person('Jane', None, None)
To make person q, you use the assignment q = Person('Robin', s, None)
A *geneaology list* is defined recursively as follows:

- It is a nonempty list with exactly three elements.
- The first element is a nonempty string, respresenting the person's name.
- The last two elements are either None or genealogy lists.

For example, the geneaology list of s is ['Jane', None, None]
The geneaology list of q is ['Robin', ['Jane', None, None], None]

Recursion with objects

```
def geneology_list(person):
    """Return: A geneology list of the Person object, person.

    For example, using the objects on the previous slide,
    geneology_list(s) returns ['Jane', None, None]
    geneology_list(q) returns ['Robin', ['Jane', None, None], None]

    Precondition: person is a Person object
    """
```

Recursion with objects

```
def geneology_list(person):
    """Return: A geneology list of the Person object, person.
    Precondition: person is a Person object """
    if person.mom is None:
        mom = None
    else:
        mom = geneology_list(person.mom)

    if person.dad is None:
        dad = None
    else:
        dad = geneology_list(person.dad)

    return [person.name, mom, dad]
```

Recursion with Dictionaries (Fall 2014)

```
def histogram(s):
    """Return: a histogram (dictionary) of the # of letters in string s.

    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.

    Example: histogram("") returns { },
             histogram('abracadabra') returns {'a':5,'b':2,'c':1,'d':1,'r':2}

    Precondition: s is a string (possibly empty) of just letters."""
```

Recursion with Dictionaries (Fall 2014)

```
def histogram(s):
    """Return: a histogram (dictionary) of the # of letters in string s.

    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.

    Precondition: s is a string (possibly empty) of just letters."""
```

Hint:

- Use divide-and-conquer to break up the string
- Get two dictionaries back when you do
- Pick one and insert the results of the other

Recursion with Dictionaries (Fall 2014)

```
def histogram(s):
    """Return: a histogram (dictionary) of the # of letters in string s."""
    if s == '':                                  # Small data
        return {}

    # We know left is { s[0]: 1 }.  No need to compute
    right = histogram(s[1:])

    if s[0] in right:                            # Combine the answer
        right[s[0]] = right[s[0]]+1
    else:
        right[s[0]] = 1
    return right
```

Recursion and the call stack

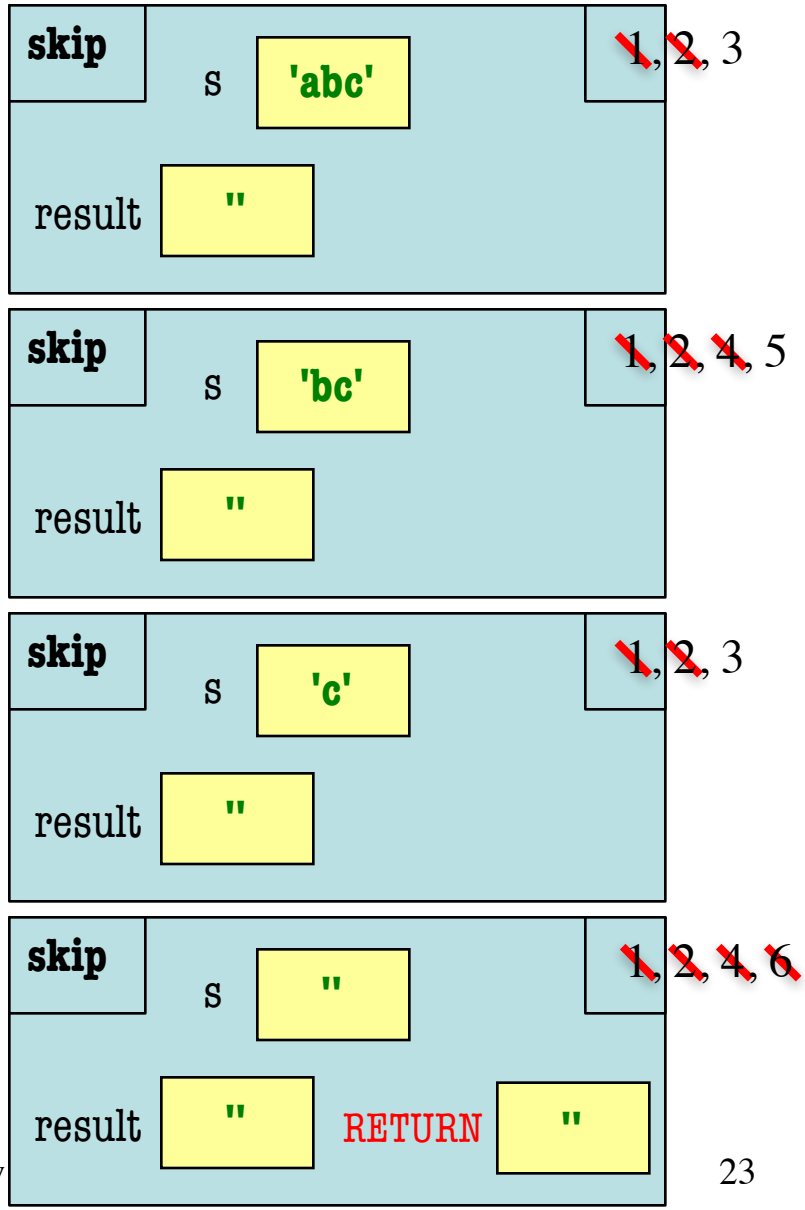
```
def skip(s):
    """Returns: copy of s
    Odd (from end) skipped"""
1   result = ''
2   if (len(s) % 2 = 1):
3       result = skip(s[1:])
4   elif len(s) > 0:
5       result = s[0]+skip(s[1:])
6   return result
```

- **Call**: skip('abc')
- Recursive call results in four frames (why?)
  - Consider when 4th frame completes line 6
  - Draw the entire call stack at that time
- Do not draw more than four frames!

Call Stack Question

```
def skip(s):
    """Returns: copy of s
    Odd (from end) skipped"""
1   result = ''
2   if (len(s) % 2 = 1):
3       result = skip(s[1:])
4   elif len(s) > 0:
5       result = s[0]+skip(s[1:])
6   return result
```

- **Call**: skip('abc')



Call Stack Question

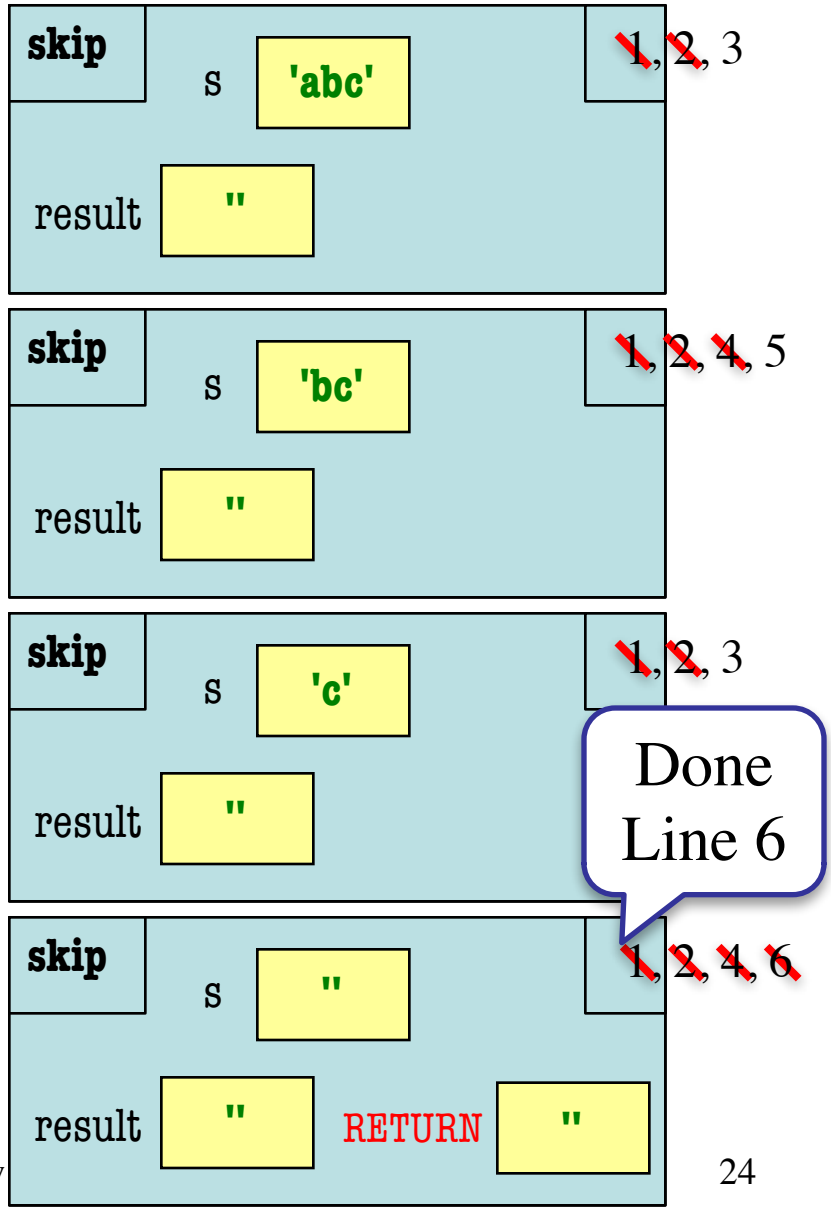
```
def skip(s):
    """Returns: copy of s
    Odd (from end) skipped"""
1   result = ''
2   if (len(s) % 2 = 1):
3       result = skip(s[1:])
4   elif len(s) > 0:
5       result = s[0]+skip(s[1:])
6   return result
```

s = 'abc'
s = 'c'

s = 'bc'

s = ''

- **Call**: skip('abc')



Good Luck!