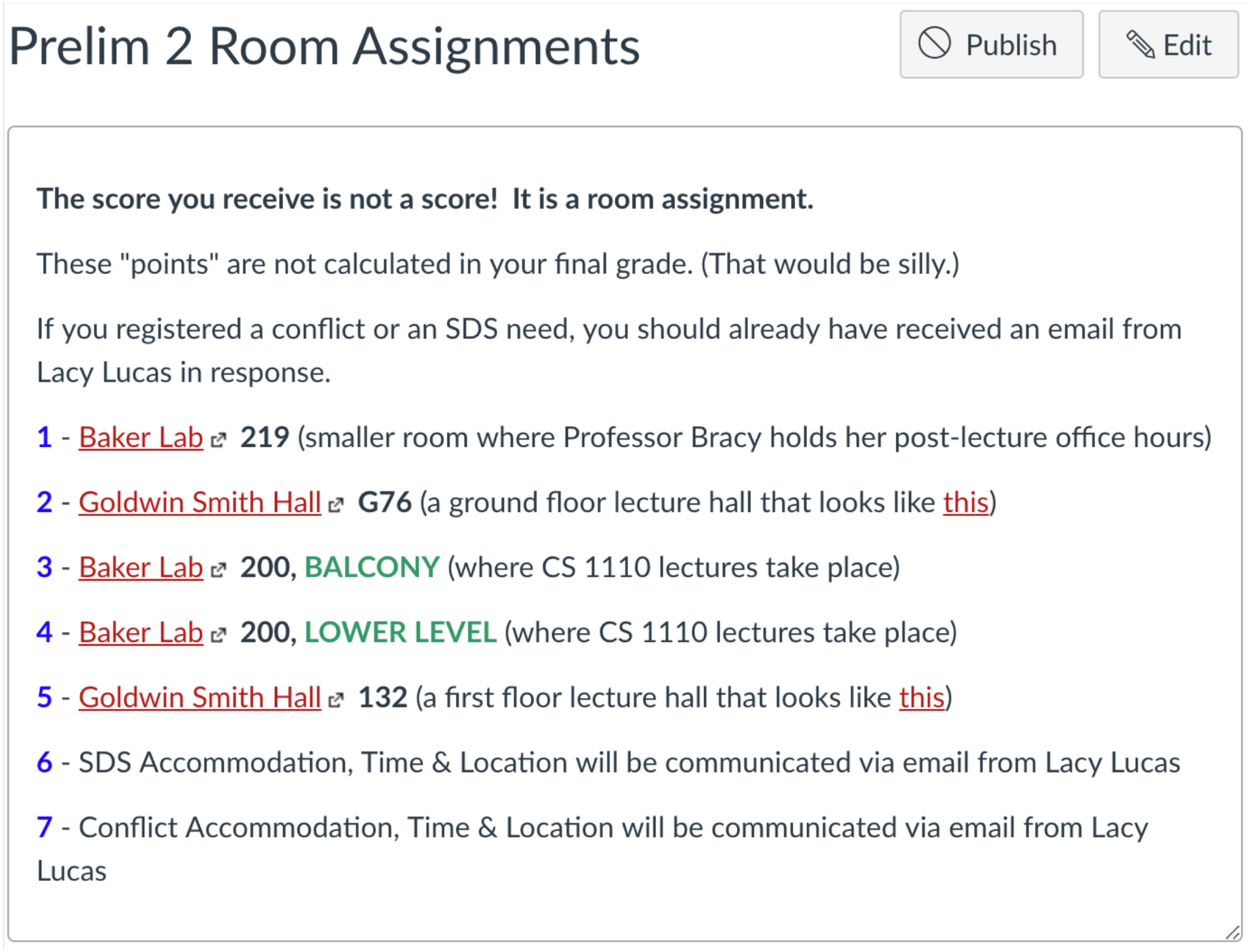
CS 1110

Prelim 2 Review
Spring 2019

Exam Info



Prelim 2 Room Assignments

The score you receive is not a score! It is a room assignment.

These "points" are not calculated in your final grade. (That would be silly.)

If you registered a conflict or an SDS need, you should already have received an email from Lacy Lucas in response.

1 - Baker Lab **219** (smaller room where Professor Bracy holds her post-lecture office hours)

2 - Goldwin Smith Hall **G76** (a ground floor lecture hall that looks like this)

3 - Baker Lab **200, BALCONY** (where CS 1110 lectures take place)

4 - Baker Lab **200, LOWER LEVEL** (where CS 1110 lectures take place)

5 - Goldwin Smith Hall **132** (a first floor lecture hall that looks like this)

6 - SDS Accommodation, Time & Location will be communicated via email from Lacy Lucas

7 - Conflict Accommodation, Time & Location will be communicated via email from Lacy Lucas

What is on the Exam?

- Questions from the following topics:
  - Iteration and Lists, Dictionaries, Tuples
    - Nested lists, nested loops
  - Recursion
  - Classes & Subclasses
  - While loops

What is on the Exam?

- Questions from the following topics:
  - Iteration and Lists, Dictionaries, Tuples
    - Nested lists, nested loops
  - Recursion
  - Classes & Subclasses
  - While loops

Iteration - For-loops

- Make sure you always keep in mind what the function is supposed to do
    - Are we modifying the sequence directly?
    - Do we need to have an accumulator variable?
- Remember what the loop variable represents
    - Is the loop variable each element(value)?
    - Is the loop variable the position(index)?
- Same goes for nested-loops

Iteration - For-loops

- Two ways to implement the for-loop

for x in list:

- x represents each value inside the list
- Modifying x does not modify the list

for x in range(len(list)):

- x represents each index inside the list
- Modifying list[x] modifies the list

Implement Using Iteration

```
def evaluate(p, x):
    """Returns: The evaluated polynomial p(x)
    We represent polynomials as a list of floats.  In other words

        [1.5, –2.2, 3.1, 0, –1.0] is 1.5 – 2.2x + 3.1x**2 + 0x**3 – x**4

    We evaluate by substituting in for the value x.  For example

      evaluate([1.5,–2.2,3.1,0,–1.0], 2) is 1.5–2.2(2)+3.1(4)–1(16) = –6.5
      evaluate([2], 4) is 2

    Precondition: p is a list (len > 0) of floats, x is a float"""
```

Implement Using Iteration

```
def evaluate(p, x):
    """Returns: The evaluated polynomial p(x)

    Precondition: p is a list (len > 0) of floats, x is a float"""
    sum = 0
    xval = 1
    for c in p:
        sum = sum + c*xval      # coefficient * (x**n)
        xval = xval * x
    return sum
```

Implement Using Iteration

```
def evaluate(p, x):
    """Returns: The evaluated polynomial p(x)

    Precondition: p is a list (len > 0) of floats, x is a float"""
    sum = 0
    xval = 1
    for c in p:
        sum = sum + c*xval
        xval = xval * x
    return sum
```

In the first iteration, we add (1st element * 1) to the sum, and then we change the xval to xval * x, so that in the second iteration we can add (2nd element * x)

Example 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"""
```

Example with 2D Lists (Like A6)

```
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
```

```
[ 4, 5, 6]

[ [ 4, 5, 6],
  [3, 1, 2],
  [9, 0, 5] ]
```

What is on the Exam?

- Questions from the following topics:
  - Iteration and Lists, Dictionaries, Tuples
    - Nested lists, nested loops
  - Recursion
  - Classes & Subclasses
  - While loops

Recursion

1. Base case
2. Recursive case
3. Ensure the recursive case makes progress towards the base case

Base Case

- Create cases to handle smallest units of data
- Ideal base cases depend on what type of data is being handled and what the function must do on that data

Recursive Case

- Divide and conquer: how to divide the input so that we can call the function recursively on smaller input
- When calling the function recursively, assume that it works exactly as the specification states it does -- don’t worry about the specifics of your implementation here
- Use this recursive call to handle the rest of the data, besides the small unit being handled

Make Progress

- Recursive calls must always make some sort of "progress" towards the base cases
- This is the only way to ensure the function terminates properly
- Risk having infinite recursion otherwise

- Please check the Recursion Session slides on the Schedule tab of the course website!!!

Recursive Function (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."""
```

Recursive Function

```
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

Recursive Function

```
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
```

What is on the Exam?

- Questions from the following topics:
  - Iteration and Lists, Dictionaries, Tuples
    - Nested lists, nested loops
  - Recursion
  - Classes & Subclasses
    - Defining Classes
    - Drawing Class folders
  - While loops

```
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
        _name: last name [string or None if unknown]
        _email: e-mail address [string or None if unknown]
    Immutable attributes:
        _born: birth year [int > 1900; -1 if unknown]"""

    # DEFINE GETTERS/SETTERS HERE
    # Enforce all invariants and enforce immutable/mutable restrictions

    # DEFINE INITIALIZER HERE
    # Initializer: Make a Customer with last name n, birth year y, e-mail address e.
    # E-mail is None by default
    # Precondition: parameters n, b, e satisfy the appropriate invariants

    # OVERLOAD STR() OPERATOR HERE
    # Return: String representation of customer
    # If e-mail is a string, format is 'name (email)'
    # If e-mail is not a string, just returns name
```

```
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
        _name: last name [string or None if unknown]
        _email: e-mail address [string or None if unknown]
    Immutable attributes:
        _born: birth year [int > 1900; -1 if unknown]"""

    # DEFINE GETTERS/SETTERS HERE
    def getName(self):
        return self._name
```

Getter

```
    def setName(self,value):
        assert value is None or type(value) == str
        self._name = value
```

Setter

```
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
        _name: last name [string or None if unknown]
        _email: e-mail address [string or None if unknown]
    Immutable attributes:
        _born: birth year [int > 1900; -1 if unknown]"""

    # DEFINE GETTERS/SETTERS HERE
    ....
    def getEmail(self):
        return self._email

    def setEmail(self,value):
        assert value is None or type(value) == str
        self._email = value
```

Getter

Setter

```
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
        _name: last name [string or None if unknown]
        _email: e-mail address [string or None if unknown]
    Immutable attributes:
        _born: birth year [int > 1900; -1 if unknown]"""

    # DEFINE GETTERS/SETTERS HERE
    ....
    def getBorn(self):
        return self._born
```

Getter

Immutable.
No Setter!

```
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
        _name: last name [string or None if unknown]
        _email: e-mail address [string or None if unknown]
    Immutable attributes:
        _born: birth year [int > 1900; -1 if unknown]"""

    # DEFINE GETTERS/SETTERS HERE
    …
    # DEFINE INITIALIZER HERE
    def __init__(self, n, y, e=None):
        assert type(y) == int and (y > 1900 or y == -1)
        self.setName(n)  # Setter handles asserts
        self.setEmail(e)  # Setter handles asserts
        self._born = y     # No setter
```

```
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
        _name: last name [string or None if unknown]
        _email: e-mail address [string or None if unknown]
    Immutable attributes:
        _born: birth year [int > 1900; -1 if unknown]"""

    # DEFINE GETTERS/SETTERS HERE
    …
    # DEFINE INITIALIZER HERE
    …
    # OVERLOAD STR() OPERATOR HERE
    def __str__(self):
        if self._email is None:
            return = '' if self._name is None else self._name
        else:
            s = '' if self._name is None else self._name
            return s+'('+self._email+')'
```

None or str

If not None, always a str

```
class PrefCustomer(Customer):
    """An instance is a 'preferred' customer
    Mutable attributes (in addition to Customer):
        _level: level of preference [One of 'bronze', 'silver', 'gold'] """

    # DEFINE GETTERS/SETTERS HERE
    # Enforce all invariants and enforce immutable/mutable restrictions

    # DEFINE INITIALIZER HERE
    # Initializer: Make a new Customer with last name n, birth year y,
    # e-mail address e, and level l
    # E-mail is None by default
    # Level is 'bronze' by default
    # Precondition: parameters n, b, e, l satisfy the appropriate invariants

    # OVERLOAD STR() OPERATOR HERE
    # Return: String representation of customer
    # Format is customer string (from parent class) +', level'
    # Use __str__ from Customer in your definition
```

```
class PrefCustomer(Customer):
    """An instance is a 'preferred' customer
    Mutable attributes (in addition to Customer):
        _level: level of preference [One of 'bronze', 'silver', 'gold'] """

    # DEFINE GETTERS/SETTERS HERE
    def getLevel(self):
        return self._level

    def setLevel(self,value):
        assert type(value) == str
        assert (value == 'bronze' or value == 'silver' or value == 'gold')
        self._level = value
```

Getter

Setter

```
class PrefCustomer(Customer):
    """An instance is a 'preferred' customer
    Mutable attributes (in addition to Customer):
        _level: level of preference [One of 'bronze', 'silver', 'gold'] """

    # DEFINE GETTERS/SETTERS HERE
    …
    # DEFINE INITIALIZER HERE
    def __init__(self, n, y, e=None, l='bronze'):
        Customer.__init__(self,n,y,e)
        self.setLevel(l)    # Setter handles asserts

    # OVERLOAD STR() OPERATOR HERE
    def __str__(self):
        return Customer.__str__(self)+', '+self._level
```

explicit calls uses method in parent class as helper

Two Example Classes

```
class CongressMember(object):
    """Instance is legislator in congress
    Instance attributes:
        _name: Member's name [str]"""

    def getName(self):
        return self._name

    def setName(self,value):
        assert type(value) == str
        self._name = value

    def __init__(self,n):
        self.setName(n)  # Use the setter

    def __str__(self):
        return 'Honorable '+self.name
```

```
class Senator(CongressMember):
    """Instance is legislator in congress
    Instance attributes (plus inherited):
        _state: Senator's state [str]"""
    def getState(self):
        return self._state

    def setName(self,value):
        assert type(value) == str
        self._name = 'Senator '+value

    def __init__(self,n,s):
        assert type(s) == str and len(s) == 2
        CongressMember.__init__(self,n)
        self._state = s

    def __str__(self):
        return (CongressMember.__str__(self)+
                    ' of '+self.state)
```

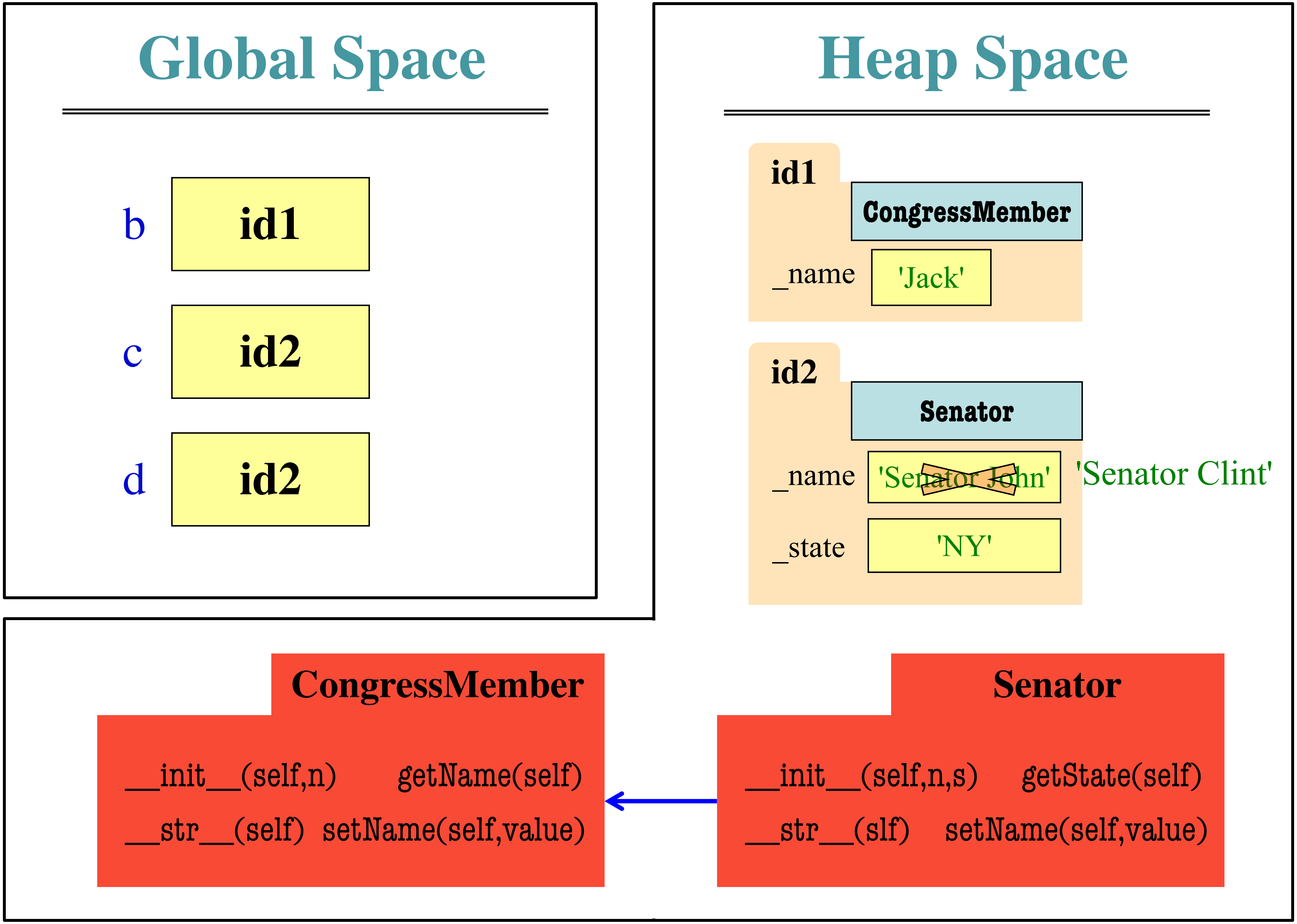
‘Execute’ the Following Code

```
>>> b = CongressMember('Jack')
>>> c = Senator('John', 'NY')
>>> d = c
>>> d.setName('Clint')
```

Remember:

Commands outside of a function definition happen in global space

- Draw two columns:
  - **Global space**
  - **Heap space**
- Draw both the
  - Variables created
  - Object folders created
  - Class folders created
- If an attribute changes
  - Mark out the old value
  - Write in the new value



Global Space

b id1

c id2

d id2

Heap Space

id1 CongressMember

_name 'Jack'

id2 Senator

_name ~~'Senator John'~~ 'Senator Clint'

_state 'NY'

CongressMember

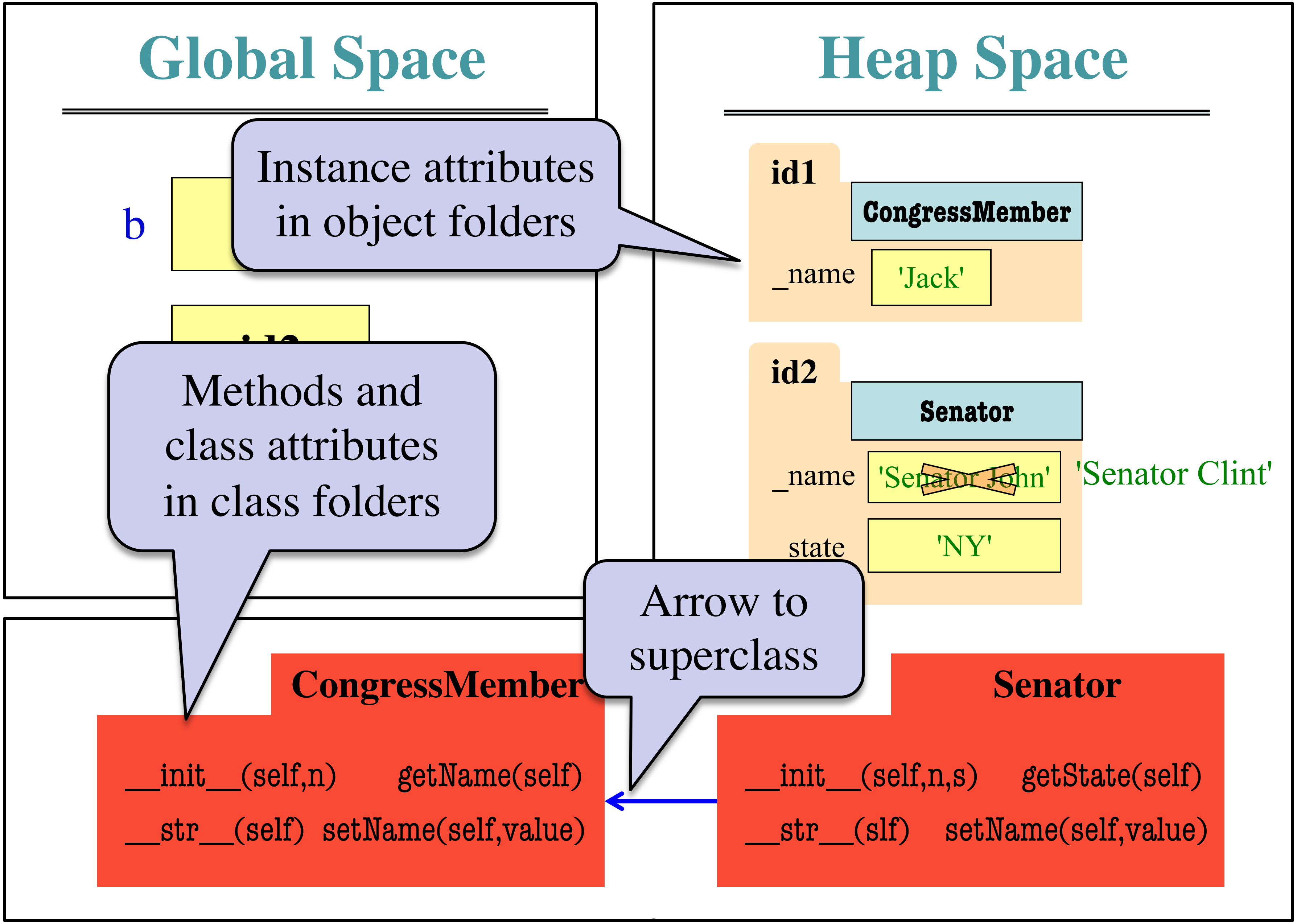
__init__(self,n) getName(self)

__str__(self) setName(self,value)

Senator

__init__(self,n,s) getState(self)

__str__(slf) setName(self,value)



Global Space

b

Heap Space

id1

CongressMember

_name 'Jack'

id2

Senator

_name ~~'Senator John'~~ 'Senator Clint'

state 'NY'

CongressMember

__init__(self,n) getName(self)

__str__(self) setName(self,value)

Senator

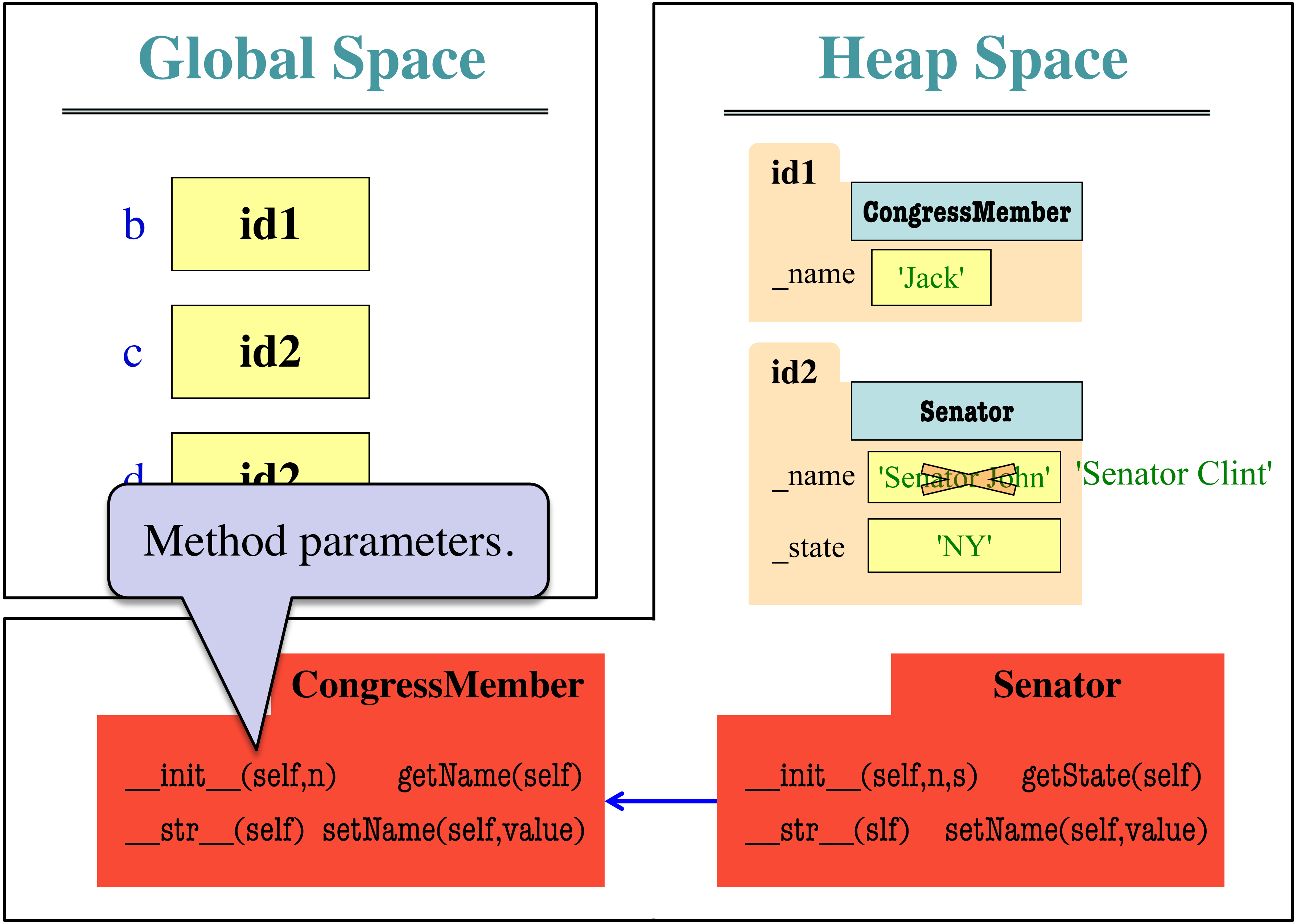
__init__(self,n,s) getState(self)

__str__(slf) setName(self,value)

Instance attributes in object folders

Methods and class attributes in class folders

Arrow to superclass



Global Space

b id1

c id2

d id2

Heap Space

id1 CongressMember

_name 'Jack'

id2 Senator

_name ~~'Senator John'~~ 'Senator Clint'

_state 'NY'

CongressMember

__init__(self,n) getName(self)

__str__(self) setName(self,value)

Senator

__init__(self,n,s) getState(self)

__str__(slf) setName(self,value)

Method parameters.

Method Overriding

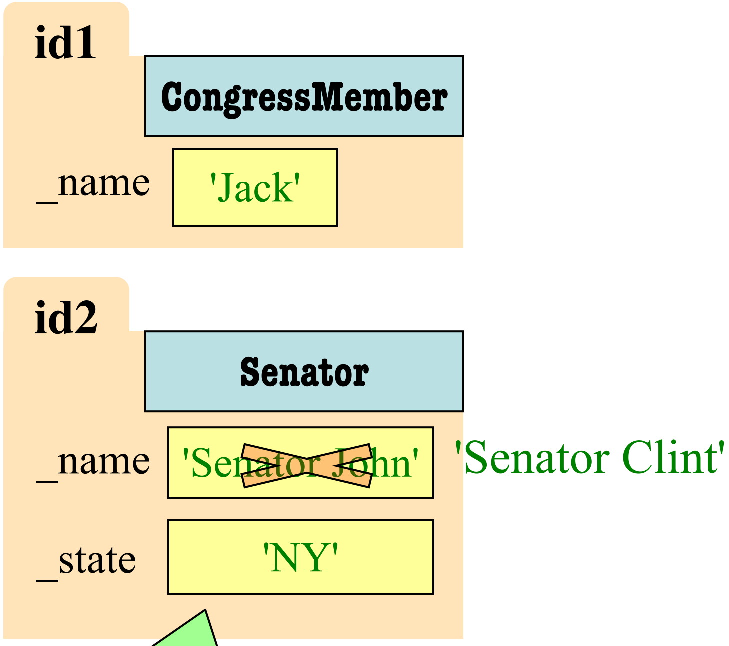
```
class Senator(CongressMember):
    """Instance is legislator in congress
    Instance attributes (plus inherited):
        _state: Senator's state [str]"""
    def getState(self):
        return self._state

    def setName(self,value):
        assert type(value) == str
        self._name = 'Senator '+value

    def __init__(self,n,s):
        assert type(s) == str and len(s) == 2
        Senator.__init__(self,n)
        self._state = s

    def __str__(self):
        return (Senator.__str__(self)+
                ' of '+self.state)
```

Heap Space



id1

CongressMember

_name 'Jack'

id2

Senator

_name ~~'Senator John'~~ 'Senator Clint'

_state 'NY'

__init__ calls setter as a helper

What is on the Exam?

- Questions from the following topics:
  - Iteration and Lists, Dictionaries, Tuples
    - Nested lists, nested loops
  - Recursion
  - Classes & Subclasses
  - While loops
    - Need to understand what the loop is doing

While-loop

- Broader notion of “keep working until done”
- Must explicitly ensure that you are “moving towards” the end
- You explicitly manage what happens each iteration

```
while <condition>:
    <statement1>
    <statement2>
```

While-loop

- Loop through a list of ints and modify the original list by adding one to each one of item

```
idx = 0
while idx < len(list):
    list[idx] = list[idx] + 1
    idx = idx + 1
```

Any More Questions?