CS 1110

Prelim 2 Review
Fall 2025

Exam Info

- **Prelim 2**: Thursday, December 4th at 7:30 pm
  - Last name **A – Ki** in Baker Lab 200
  - Last name **Kl – Li** in Baker Lab 219
  - Last name **Lo – V** in Rockefeller 201
  - Last name **W – Z** in Rockefeller 203
  - SDS Students received an e-mail
- Exceptions ONLY if you filed a conflict
  - We expect you at time and room assigned

Studying for the Exam

- Read study guides, review slides online
  - Solution to review posted after review
- Review all labs and assignments
  - Solutions to Assignment 5 are in CMS
  - No solutions to code, but talk to TAs
- Look at exams from past years
  - Exams with solutions on course web page
  - Do not search for other course pages (e.g. Spring)

What is on the Exam?

- **Four or Five** questions on these topics :
  - Recursion (Labs 15 and 16; A4)
  - Iteration (Labs 13, 17, and 22; A4; A6)
  - Defining classes (Labs 18, 19, 20, & 21; A6)
  - Drawing folders (Lecture; A5)
  - Short Answer (Terminology, Potpourri)
- + 2 pts for writing your name and net-id
- Exact number depends on question length

What is on the Exam?

- Recursion (Labs 15 and 16; A4)
  - Will be given a function specification
  - Implement it using recursion
  - May have an associated call stack question
- Iteration (Labs 13, 17, and 22; A4; A6)
- Defining classes (Labs 18, 19, 20, and 21; A6)
- Drawing folders (Lecture; A5)
- Short Answer (Terminology, Potpourri)

Recursive Function (Fall 2017)

```
def filter(nlist):
    """Return: a copy of nlist (in order) with negative numbers.

    The order of the original list is preserved

    Example: filter([1,-1,2,-3,-4,0]) returns [1,2,0]

    Precondition: nlist is a (possibly empty) list of numbers."""
```

Recursive Function (Fall 2017)

```
def filter(nlist):
    """Return: a copy of nlist (in order) with negative numbers.

    The order of the original list is preserved

    Example: filter([1,-1,2,-3,-4,0]) returns [1,2,0]

    Precondition: nlist is a (possibly empty) list of numbers."""
```

Hint:

- Use divide-and-conquer to break up the list
- Filter each half and put back together

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 (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

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')
- 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!

What is on the Exam?

- Recursion (Labs 15 and 16; A4)
- Iteration (Labs 13, 17, and 22; A4; A6)
  - Again, given a function specification
  - Implement it using a for-loop
  - May involve 2-dimensional lists
- Defining classes (Labs 18, 19, 20, and 21; A6)
- Drawing folders (Lecture; A5)
- Short Answer (Terminology, Potpourri)

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"""
```

Example with 2D Lists (Like A6)

```
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"""
```

While-Loop Example

```
def dupevens(nums):
    """MODIFIES nums so that all even numbers are duplicated

    The duplicate value should appear immediately after the original.
    Example: If a = [1,2,3,4], dupevens(a) modifies a to [1,2,2,3,4,4]

    Precondition: nums is a list of integers"""
```

What is on the Exam?

- Recursion (Labs 15 and 16; A4)
- Iteration (Labs 13, 17 and 22; A4, A6)
- Defining Classes (Labs 18, 19, 20, and 21; A6)
  - Given a specification for a class
  - Also given a specification for a subclass
  - Will "fill in blanks" for both
- Drawing folders (Lecture; A5)
- Short Answer (Terminology, Potpourri)

```
class Customer(object):
    """Instance is a customer for our company"""
    # MUTABLE ATTRIBUTES:
    # _name: string or None if unknown
    # _email: string or None if unknown
    # IMMUTABLE ATTRIBUTES:
    # _born: 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, y, 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 PrefCustomer(Customer):
    """An instance is a 'preferred' customer"""
    # MUTABLE ATTRIBUTES (in addition to Customer):
    # _level: 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, y, 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
```

What is on the Exam?

- Recursion (Labs 15 and 16; A4)
- Iteration (Labs 13, 17, and 22; A4, A6)
- Defining classes (Labs 18, 19, 20, and 21; A6)
- Drawing class folders (Lecture; **A5**)
  - Given a skeleton for a class
  - Also given several assignment statements
  - Draw all folders and variables created
- Short Answer (Terminology, Potpourri)

Two Example Classes

```
class CongressMember(object):
    """Instance is legislator in congress"""
    # INSTANCE ATTRIBUTES:
    # _name: a string

    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 (additional):
    # _state: a string
    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
        super().__init__(n)
        self._state = s

    def __str__(self):
        return (super().__str__()+
                ' 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

What is on the Exam?

- Recursion (Labs 15 and 16; A4)
- Iteration (Labs 13, 17, and 22; A4; A6)
- Defining classes (Labs 18, 19, 20, and 21; A6)
- Drawing class folders (Lecture; A5)
- Short Answer (Terminology, Potpourri)
  - See the study guide
  - Look at the lecture slides
  - Look at the lecture demo code

In that order

Any More Questions?