CS 1110

Prelim 2 Review
Fall 2020
Exam Info

• Prelim 2: Thursday, November 19th at 9:30 am
  ▪ In-person students in Barton Hall
  ▪ SDS students in 114 and 122 Gates
  ▪ Exam Seating contains room, time AND entrance!

• Online students will work in Gradescope
  ▪ Exam Seating contains your proctor information
  ▪ Proctor will contact you directly
  ▪ There are no plans hold mock exam unless you ask
Exam Info

• Prelim 2: Thursday, November 19th at 9:30 am
  ▪ In-person students in Barton Hall
  ▪ SDS students in 114 and 122 Gates
  ▪ Exam Seating contains room, time AND entrance!

Do not come to Barton early!
Do not crowd the entrance!

• Online students will work in Gradescope
  ▪ Exam Seating contains your proctor information
  ▪ Proctor will contact you directly
  ▪ There are no plans hold mock exam unless you ask
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
  ▪ Only look at fall exams; spring is VERY different
What is on the Exam?

- **Four or Five questions** on these topics:
  - Recursion (Labs 13 & 14, A4)
  - Iteration and Lists (Labs 12 & 15, A4, A6)
  - Defining classes (Labs 16-18, A6)
  - Drawing folders (In class, 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?

- **Four or Five** questions on these topics:
  - Recursion (Labs 13 & 14, A4)
  - Iteration and Lists (Labs 12 & 15, A4, A6)
  - Defining classes (Labs 16-18, A6)
  - Drawing folders (In class, A5)
  - Short Answer (Terminology, Potpourri)
- + 2 pts for writing your name and net-id
- Exact number depends on question length

11/16/20 Prelim 2 Review
What is on the Exam?

- **Recursion (Labs 13 & 14, A4)**
  - Will be given a function specification
  - Implement it using recursion
  - May have an associated call stack question

- **Iteration and Lists (Labs 12 & 15, A4, A6)**

- **Defining classes (Labs 16-18, A6)**

- **Drawing folders (In class, A5)**

- **Short Answer (Terminology, Potpourri)**
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."""
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
def filter(nlist):

    """Return: a copy of nlist (in order) with negative numbers."""
    if len(nlist) == 0:
        return []
    elif len(nlist) == 1:
        return nlist[:] if nlist[0] >= 0 else []  # THIS does the work
    # Break it up into halves
    left = filter(nlist[:1])
    right = filter(nlist[1:])
    # Combine
    return left + right
def filter(nlist):
    """Return: a copy of nlist (in order) with negative numbers."""
    if len(nlist) == 0:
        return []

    # Do the work by removing one element
    left = nlist[:1]
    if left[0] < 0:
        left = []
    right = filter(nlist[1:])

    # Combine
    return left + right
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."""
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

```python
def histogram(s):
    """Return: a histogram (dictionary) of the # of letters in string s."""
    if s == '':  # Small data
        return {}
    # left = { s[0]: 1 }. No need to compute this
    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
```

11/16/20 Prelim 2 Review
def skip(s):
    
    """Returns: copy of s
    Odd (from end) skipped"
    result = ''

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

• Call: skip('abc')
def skip(s):
    """Returns: copy of s
    Odd (from end) skipped"
    result = ''
    if (len(s) % 2 == 1):
        result = skip(s[1:])
    elif len(s) > 0:
        result = s[0]+skip(s[1:])
    return result

• Call: skip('abc')

1  s = 'abc'
2  s = 'c'
3  s = 'bc'
4
5  s = 'c'
6  s = ''

result

Done Line 6
What is on the Exam?

- Recursion (Labs 13 & 14, A4)
- Iteration and Lists (Labs 12 & 15, A4, A6)
  - Again, given a function specification
  - Implement it using a for-loop
  - May involve 2-dimensional lists
- Defining classes (Labs 16-18, A6)
- Drawing folders (In class, A5)
- Short Answer (Terminology, Potpourri)
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"""
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
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"""
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
What is on the Exam?

• Recursion (Labs 13 & 14, A4)
• Iteration and Lists (Labs 12 & 15, A4, A6)
• Defining Classes (Labs 16-18, 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 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

def getName(self):
    return self._name

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

    ....
    def getEmail(self):
        return self._email

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

Actual Exam Question
probably not this long.
Just for this practice.
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
    ....
def getBorn(self):
        return self._born
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
    ...

    # 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

Actual Exam Question
probably not this long.
Just for this practice.
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
    ...
    # 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+')'
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
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
    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

Actual Exam Question
will not be this long.
Just for this practice.
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
    ...
    # DEFINE INITIALIZER HERE
    def __init__(self, n, y, e=None, l='bronze'):
        super().__init__(n,y,e)
        self.setLevel(l)    # Setter handles asserts

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

Actual Exam Question will not be this long.
Just for this practice.

Using super() in place of self uses parent __str__
What is on the Exam?

- Recursion (Labs 13 & 14, A4)
- Iteration and Lists (Labs 12 & 15, A4, A6)
- Defining Classes (Labs 16-18, 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

```python
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 setState(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)
```

11/16/20 Prelim 2 Review 34
‘Execute’ the Following Code

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

• 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

Remember:
Commands outside of a function definition happen in global space
**Global Space**

Instance attributes in object folders

**Heap Space**

- id1
  - CongressMember
  - _name: 'Jack'
- id2
  - Senator
  - _name: 'Senator John'
  - state: 'NY'
  - _name: 'Senator Clint'

Methods and class attributes in class folders

- __init__(self, n)
- getName(self)
- __str__(self)
- setName(self, value)

- __init__(self, n, s)
- getState(self)
- __str__(self)
- setName(self, value)

Arrow is optional
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)

11/16/20 Prelim 2 Review
What is on the Exam?

- Recursion (Lab 7, A4)
- Iteration and Lists (Lab 8, A4, A6)
- Defining classes (Lab 9, A6)
- Drawing class folders (Lecture, A5)
- Short Answer (Terminology, Potpourri)
  - See the study guide
  - Look at the lecture slides
  - Read relevant book chapters

In that order
What is on the Exam?

- Recursion (Lab 7, A4)
- Iteration and Lists (Lab 8, A4, A6)
- Defining classes (Lab 9, A6)
- Drawing class folders (Lecture, A5)
- Short Answer (Terminology, Potpourri)

- See the study guide
- Look at the lecture slides
- Read relevant book chapters

Unlikely to happen

In that order
Any More Questions?