CS 1110

Prelim 2 Review Part 1
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

- No Electronics, Calculators, Notes, or Books
- Bring Your Cornell ID
- 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

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

```
class Customer(________):
    """Instance is a customer for our company
    Attributes:
        name: last name [string]
        email: e-mail address [string or None if unknown]
        purchases: number of items bought, [int >= 0]
        spent: money spend at our company [float >= 0.0]"""
```

```
class Customer(object):
    """Instance is a customer for our company
    Attributes:
        name: last name [string]
        email: e-mail address [string or None if unknown]
        purchases: number of items bought, [int >= 0]
        spent: money spend at our company [float >= 0.0]"""

    def __init__(________________):
        """Initialize a new Customer with name n, optional email e, and
        no purchases or spending
        Pre: n is a string, e is a string or None"""
```

Object = Not a Subclass

```
class Customer(object):
 """Instance is a customer for our company
   Attributes:
      name: last name [string]
      email: e-mail address [string or None if unknown]
      purchases: number of items bought, [int >= 0]
      spent: money spend at our company [float >= 0.0]"""

   def __init__(self, n, e=None):
      """Initialize a new Customer with name n, optional email e, and
      no purchases or spending
      Pre: n is a string, e is a string or None"""
```

Optional Attribute e

```
class Customer(object):
 """Instance is a customer for our company
   Attributes:
      name: last name [string]
      email: e-mail address [string or None if unknown]
      purchases: number of items bought, [int >= 0]
      spent: money spend at our company [float >= 0.0]"""

   def __init__(self, n, e=None):
      """Initialize a new Customer with name n, optional email e, and
      no purchases or spending
      Pre: n is a string, e is a string or None"""
      self.name = n
      self.email = e
      self.purchases = 0
      self.spent = 0.0
```

Note: not everything you need to initialize is in the parameters!

```
class Customer(object):
 """Instance is a customer for our company
   Attributes:
      name: last name [string]
      email: e-mail address [string or None if unknown]
      purchases: number of items bought, [int >= 0]
      spent: money spend at our company [float >= 0.0]"""“

   def __str__(________):
      """“Returns String Representation of this customer:
      Name (email, if exists)”””
```

```
class Customer(object):
 """Instance is a customer for our company
   Attributes:
      name: last name [string]
      email: e-mail address [string or None if unknown]
      purchases: number of items bought, [int >= 0]
      spent: money spend at our company [float >= 0.0]"""“

   def __str__(self):
      """“Returns String Representation of this customer:
      Name (email, if exists)”””
      if self.email is None:
         return self.name
      else:
         return self.name +'('+self.email+')'
```

```
class Customer(object):
    """Instance is a customer for our company
    Attributes:
        name: last name [string]
        email: e-mail address [string or None if unknown]
        purchases: number of items bought, [int >= 0]
        spent: money spend at our company [float >= 0.0]"""

    def makePurchase(__________):
        ""“Update customer after making a purchase of c dollars
        Pre: c float >= 0.0 ”””
```

```
class Customer(object):
    """Instance is a customer for our company
    Attributes:
        name: last name [string]
        email: e-mail address [string or None if unknown]
        purchases: number of items bought, [int >= 0]
        spent: money spend at our company [float >= 0.0]"""

    def makePurchase(self, c):
        """“Update customer after making a purchase of c dollars
        Pre: c float >= 0.0 ”””
        self.purchases += 1;
        self.spent += c;
```

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

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

    def __init__(____________________):
        """Initialize a new PrefCustomer with name n, optional email e, and
        no purchases or spending, and level l
        Pre: n is a string, e is a string or None"""
```

Superclass in the Header

We are "overloading" the initializer

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

    def __init__(self, n, l, e=None):
        """Initialize a new PrefCustomer with name n, optional email e, and
        no purchases or spending, and level l
        Pre: n is a string, e is a string or None"""
        Customer.__init__(self,n,e=e)
        self.level = l
```

Call the Superclass initializer explicitly as a helper!

__str__, makePurchase "Inherited" from Parent Class

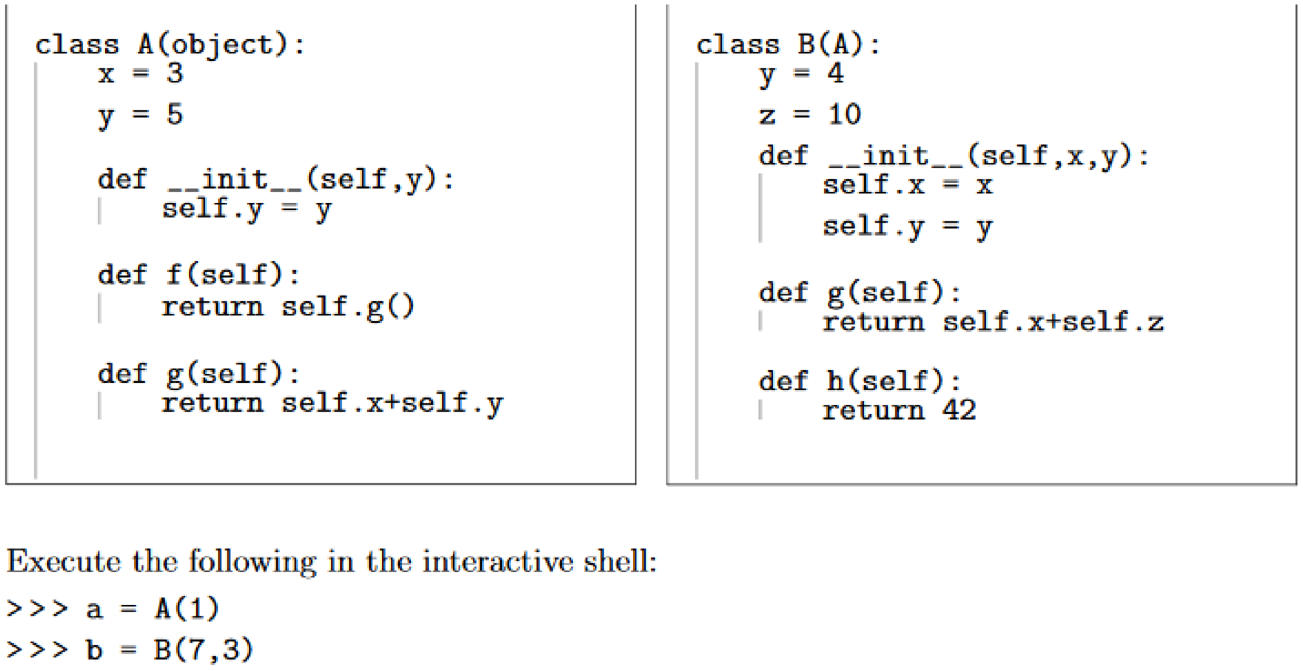
Notes on ‘self’

- What is ‘self’?
  - Not just a random thing you stick in front of stuff in Classes!!!
  - Contains the ID of the object on which the method was called
- Why is self.method() preferred to ClassName.method(self) ?
  - If a class is extended with a subclass, self may refer to an object of the subclass, and method() may be overloaded in the subclass.

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

Name Resolution (from P2 Fall 2013)



```
class A(object):
    x = 3
    y = 5

    def __init__(self,y):
        self.y = y

    def f(self):
        return self.g()

    def g(self):
        return self.x+self.y
```

```
class B(A):
    y = 4
    z = 10
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def g(self):
        return self.x+self.z

    def h(self):
        return 42
```

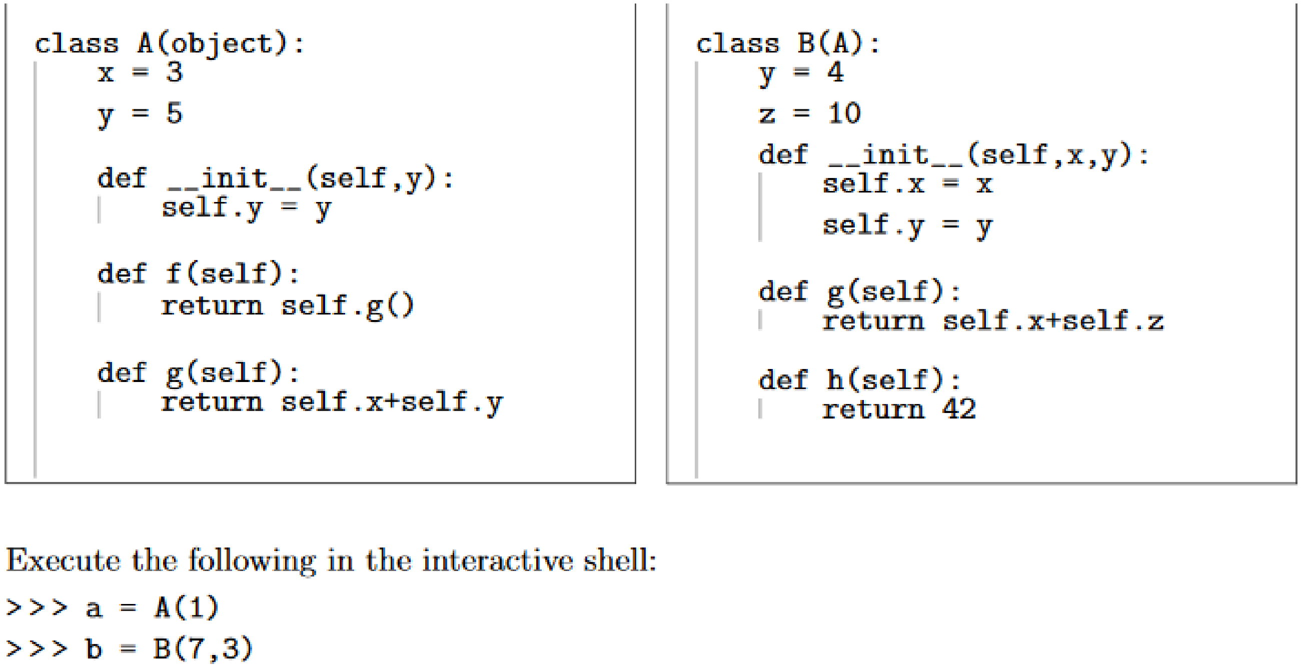
Execute the following in the interactive shell:

```
>>> a = A(1)
>>> b = B(7,3)
```

Execute the Following:

1) a.y →
2) a.z →
3) b.y →
4) B.y →

Name Resolution (from P2 Fall 2013)



```
class A(object):
    x = 3
    y = 5

    def __init__(self,y):
        self.y = y

    def f(self):
        return self.g()

    def g(self):
        return self.x+self.y
```

```
class B(A):
    y = 4
    z = 10
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def g(self):
        return self.x+self.z

    def h(self):
        return 42
```

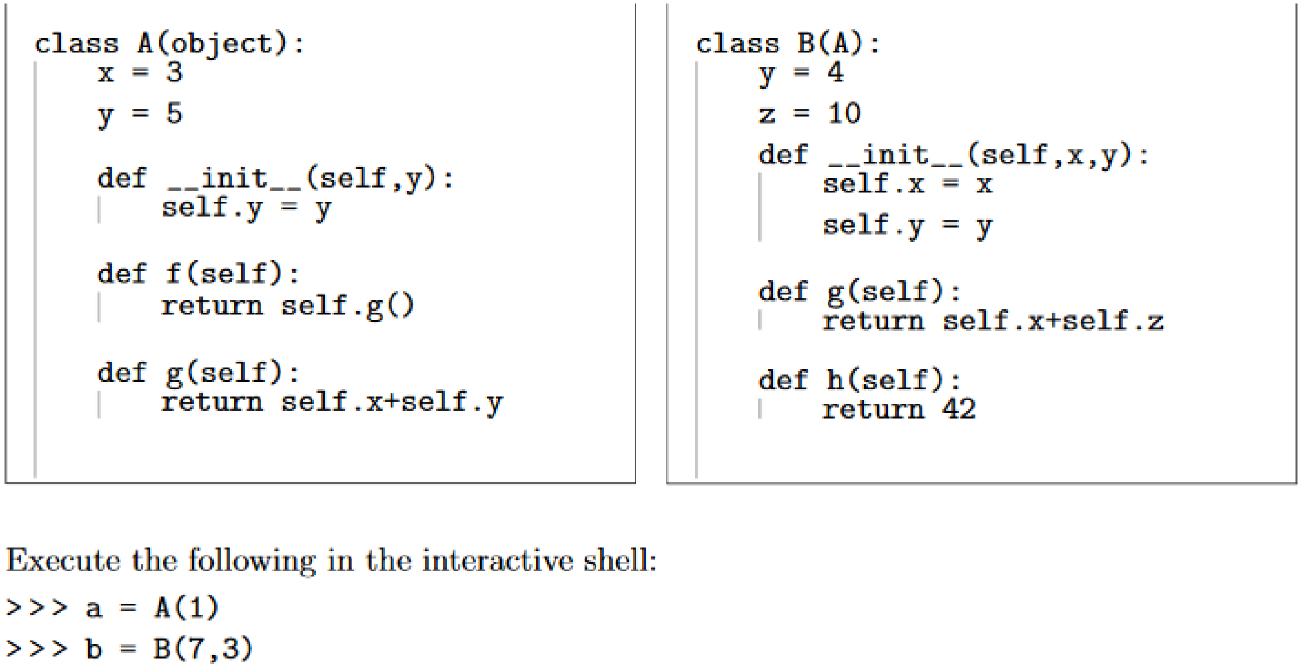
Execute the following in the interactive shell:

```
>>> a = A(1)
>>> b = B(7,3)
```

Execute the Following:

1) a.y → 1
2) a.z → error
3) b.y → 3
4) B.y → 4

Name Resolution (from P2 Fall 2013)



```
class A(object):
    x = 3
    y = 5

    def __init__(self,y):
        self.y = y

    def f(self):
        return self.g()

    def g(self):
        return self.x+self.y
```

```
class B(A):
    y = 4
    z = 10
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def g(self):
        return self.x+self.z

    def h(self):
        return 42
```

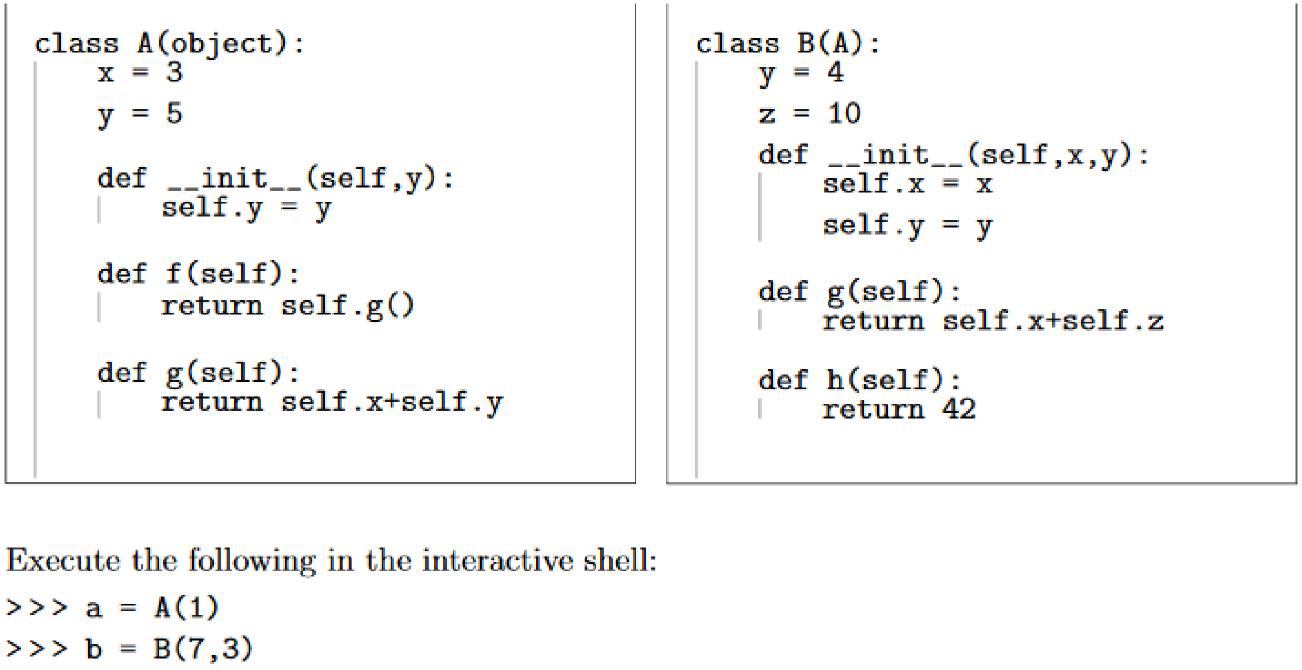
Execute the following in the interactive shell:

```
>>> a = A(1)
>>> b = B(7,3)
```

Execute the Following:

1) a.f() →
2) b.f() →
3) a.f →
4) A.g(b) →

Name Resolution (from P2 Fall 2013)



```
class A(object):
    x = 3
    y = 5

    def __init__(self,y):
        self.y = y

    def f(self):
        return self.g()

    def g(self):
        return self.x+self.y
```

```
class B(A):
    y = 4
    z = 10
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def g(self):
        return self.x+self.z

    def h(self):
        return 42
```

Execute the following in the interactive shell:

```
>>> a = A(1)
>>> b = B(7,3)
```

Execute the Following:

1) a.f() → 4
2) b.f() → 17
3) a.f → <method A.f >
4) A.g(b) → 10

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

Invariants

- What’s an Invariant?
  - An assertion (usually a condition) that is supposed to "always" be true in a piece of code
  - If temporarily invalidated, must make it true again
- Loop Invariant – An assertion that should be true before and after every iteration of the loop
- Class Invariant – assertion on value of attribute
  - E.g. [int, 0…maxValue]

While Loop Development Tips

- Initialize: Make Invariant True to start
- Terminate: Figure out where your loop should stop and translate this into your while loop condition
- Write the loop body to make progress toward termination and keep invariant True
- Note: Pay attention to range:

  Note: a..b <=> range(a, b+1)

While Loop Function

```
def e_approximate(x, tol):
    """Returns: an integer giving the number of taylor series
    terms neccessary for an approximation of e^x that is
    between -tol and +tol of the actual value.
    You can assume the math module is imported.
    A taylor series approximation for e^x is the sum of
    x^n / factorial(n), plus 1. That is to say, a two term
    approximation is:
        e^x ~ 1 + x^1/factorial(1) + x^2/factorial(2)

    Pre: x is an int, tol a float. """
```

You would be given more details of any math concept used on an exam

While Loop Function

```
def e_approximate(x, tol):
    """“Spec"""
    target = math.exp(x)
    #invariant: approx is the taylor series approximation of e^x
    #with n terms
```

While Loop Function

```
def e_approximate(x, tol):
    """“Spec"""
    target = math.exp(x)
    #invariant: approx is the taylor series approximation of e^x
    #with n terms
    n = 0
    approx = 1.0
```

While Loop Function

```
def e_approximate(x, tol):
    """“Spec"""
    target = math.exp(x)
    #invariant: approx is the taylor series approximation of e^x
    #with n terms
    n = 0
    approx = 1.0
    while abs(approx - target) > tol:
```

While Loop Function

```
def e_approximate(x, tol):
    """“Spec"""
    target = math.exp(x)
    #invariant: approx is the taylor series approximation of e^x
    #with n terms
    n = 0
    approx = 1.0
    while abs(approx - target) > tol:
        n += 1
        approx += x**n / float(math.factorial(n))
    return n
```

Note: setting n=1 and flipping the while loop statements violates the invariant!

Good Luck!