Review 2

Classes and Subclasses

Class Definition

class <*name*>(<superclass>):

    """Class specification"""

    class attributes

    initializer (__init__)

    definition of methods

    anything else

Class type to extend
(may need module name)

- Every class must extend *something*
- Most classes will extended object

Attribute Invariants

- Attribute invariants are important for programmer
  - Can look at them when writing methods
  - Any reader of the code will benefit as well

```
class Time(object):
    """An instance is a time of day
           hr: hour of the day [int in range 0..23]
           min: minute of the hour [int in range 0..59]
    """

    ...
```

Pay Attention to Specification

```
class Time(object):
    """An instance is a time of day
           hr: hour of the day [int in range 0..23]
           min: minute of the hour [int in range 0..59]
    """
    def __init__(                    ): #fill in here
        """Initializer
           Precondition: h represents the hour [int]
                         m represents the minute [int]
        """
        #implement me
```

Pay Attention to Specification

```
class Time(object):
    """An instance is a time of day
           hr: hour of the day [int in range 0..23]
           min: minute of the hour [int in range 0..59]
    """
    def __init__(self, h, m): #fill in here
        """Initializer
           Precondition: h represents the hour [int]
                         m represents the minute [int]
        """
        self.hr  = h
        self.min = m
```

Pay Attention to the Specification

```
class Time(object):
    """An instance is a time of day
        hr: hour of the day [int in range 0..23]
        min: minute of the hour [int in range 0..59]
    """
    def __init__(self, h, m): #fill in here
        """Initializer
            Precondition: h represents the hour [int]
                          m represents the minute [int]
        """
        self.hr  = h
        self.min = m
```

Pay Attention to the Specification

```
class Time(object):
    """An instance is a time of day
        hr: hour of the day [int in range 0..23]
        min: minute of the hour [int in range 0..59]
    """
    def __init__(self, h, m): #fill in here
        """Initializer
            Precondition: h represents the hour [int]
                          m represents the minute [int]
        """
        self.hr  = h
        self.min = m
```

Special Methods

- Start/end with underscores
  - __init__ for initializer
  - __str__ for str()
  - __repr__ for backquotes
- Actually defined in object
  - You are overriding them
  - Many more of them
- For a complete list, see

  http://docs.python.org/reference/datamodel.html

```
class Point(object):
    """Instances are points in 3D space"""
    ...

    def __init__(self,x=0,y=0,z=0):
        """Initializer: makes new Point"""
        ...
    def __str__(self):
        """Returns: string with contents""”
        ...
    def __repr__(self):
        """Returns: unambiguous string""”
        ...
```

Additional notes on classes

- Three steps to evaluate a Constructor Expression
  - Create a new object (folder) and put it in heap space
  - Execute the method __init__
    - Passes folder name to *self*, and passes other arguments in order
  - Return the object(folder) id
- Method call and relation to *self*
  - obj.method(args)
  - The object calling the method gets passed into the method as the first argument “self”
- Parameters with default values

Modified Question from Fall 2010

- An object of class `Course` (next slide) maintains a course name, the instructors involved, and the list of registered students, sometimes called the roster.
  1. State the purpose of an initializer. Then complete the body of the initializer of `Course`, fulfilling this purpose.
  2. Complete the body of method `add` of `Course`
  3. Complete the body of method `__eq__` of `Course`. If you write a loop, you do not need to give a loop invariant.
  4. Complete the body of method `__ne__` of `Course`. Your implementation should be a single line.

Modified Question from Fall 2010

```
class Course(object):
    """An instance is a course at Cornell.
    Maintains the name of the course, the roster
    (list of netIDs of students registered for it),
    and a list of netIDs of instructors.
        name: Course name [str]
        instructors: instructor net-ids
                     [nonempty list of string]
        roster: student net-ids
                [list of string, canbe empty]"""

    def __init__(self,name,b):
        """Instance w/ name, instructors b, no students.
        It must COPY b.  Do not assign b to instructors.
        Pre: name is a string, b is a nonemepty list"""
        # IMPLEMENT ME
```

```
    def add(self,n):
        """If student with netID n is not in roster, add
        student. Do nothing if student is already there.
        Precondition: n is a valid netID."""
        # IMPLEMENT ME

    def __eq__(self,ob):
        """Return True if ob is a Course with the same
        name and same set of instructors as this;
        otherwise return False"""
        # IMPLEMENT ME

    def __ne__(self,ob):
        """Return False if ob is a Course with the same
        name and same set of instructors as this;
        otherwise return True"""
        # IMPLEMENT ME IN ONE LINE
```

Modified Question from Fall 2010

1. State the purpose of a initializer. Complete the body of the constructor of `Course`, fulfilling this purpose.
   - The purpose is to initialize instance attributes so that the invariants in the class are all satisfied.

```
def __init__(self, name, b):
    """Instance w/ name, instructors b, no students.
    Pre: name is a string, b is a nonemepty list"""
    self.name = name
    self.instructors = b[:]  # Copies b
    self.roster = []         # Satisfy the invariant!
```

Modified Question from Fall 2010

2. Complete the body of method add of Course

```
def add(self,n):
    """If student with netID n is not in roster, add
    student. Do nothing if student is already there.
    Precondition: n is a valid netID."""
    if not n in self.roster:
        self.roster.append(n)
```

Modified Question from Fall 2010

3. Complete body of method __eq__ of Course.

```
def __eq__(self,ob):
    """Return True if ob is a Course with the same name and same
    set of instructors as this; otherwise return False"""
    if not (isinstance(ob,Course)):
        return False
    # Check if instructors in ob are in this
    for inst in ob.instructors:
        if not inst in self.instructors:
            return False
    # If instructors of ob are those in self, same if length is same
    return self.name==ob.name and len(self.instructors)==len(ob.instructors)
```

Modified Question from Fall 2010

4. Complete body of method __ne__ of Course. Your implementation should be a single line.

```
def __ne__(self,ob):
    """Return False if ob is a Course with the same  name and
    same set of instructors as this; otherwise return True"""
    # IMPLEMENT ME IN ONE LINE
    return not self == ob # Calls __eq__
```

Modified Question from Fall 2010

- An instance of Course always has a lecture, and it may have a set of recitation or lab sections, as does CS 1110. Students register in the lecture and in a section (if there are sections). For this we have two other classes: Lecture and Section. We show only components that are of interest for this question
- Do the following:
  - Complete the constructor in class Section
  - Complete the method add in Section
- Make sure invariants are enforced at all times

Modified Question from Fall 2010

```
class Lecture(Course):
    """Instance is a lecture, with list of sections
       seclist: sections associated with lecture.
                [list of Section; can be empty]
    """

    def __init__(self, n, ls):
        """Instance w/ name, instructors ls, no students.
        It must COPY ls.  Do not assign ls to instructors.
        Pre: n is a string, ls is a nonemepty list"""
        # IMPLEMENT ME
```

```
class Section(Course):
    """Instance is a section associated w/ a lecture"""
       mainlecture: lecture this section is associated.
                    [Lecture; should not be None]"""

    def __init__(self, n, ls, lec):
        """Instance w/ name, instructors ls, no
        students AND primary lecture lec.
        Pre: n a string, ls list, lec a Lecture"""
        # IMPLEMENT ME

    def add(self,n):
        """If student with netID n is not in roster of
        section,  add student to this section AND the
        main lecture.  Do nothing if already there.
        Precondition: n is a valid netID."""
        # IMPLEMENT ME
```

Modified Question from Fall 2010

```
class Lecture(Course):
    """Instance is a lecture, with list of sections
       seclist: sections associated with lecture.
                [list of Section; can be empty]
    """

    def __init__(self, n, ls):
        """Instance w/ name, instructors ls, no students.
        It must COPY ls.  Do not assign ls to instructors.
        Pre: n is a string, ls is a nonemepty list"""
        Course.__init__(self, n, ls)
        self.seclist = []
```

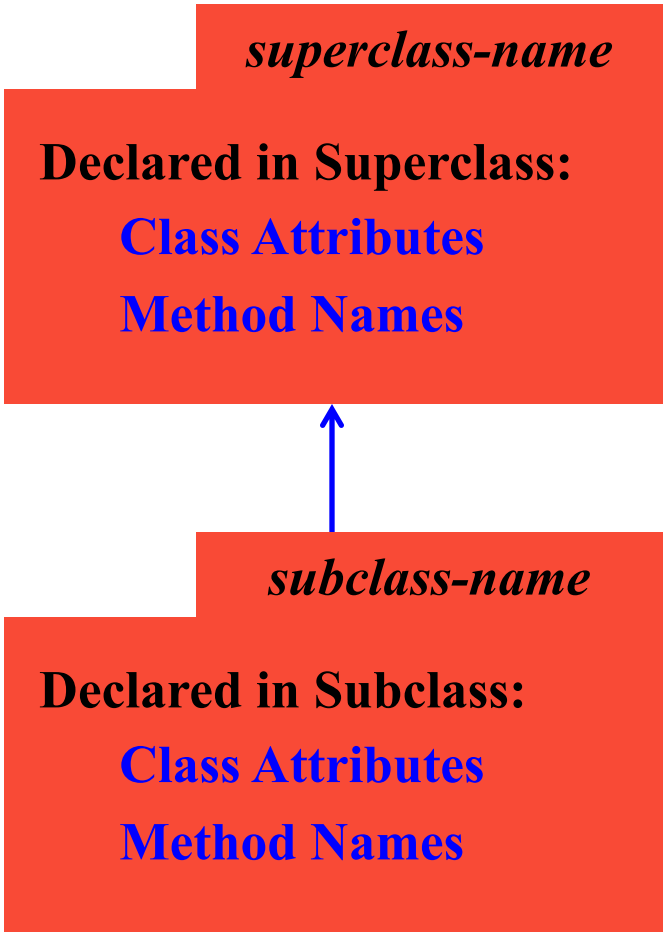
```
class Section(Course):
    """Instance is a section associated w/ a lecture"""
       mainlecture: lecture this section is associated.
                    [Lecture; should not be None]"""

    def __init__(self, n, ls, lec):
        """Instance w/ name, instructors ls, no
        students AND primary lecture lec.
        Pre: n a string, ls list, lec a Lecture"""
        # IMPLEMENT ME

    def add(self,n):
        """If student with netID n is not in roster of
        section,  add student to this section AND the
        main lecture.  Do nothing if already there.
        Precondition: n is a valid netID."""
        # IMPLEMENT ME
```

Modified Question from Fall 2010

```
def __init__(self, n, ls, lec):
    """Instance w/ name, instructors ls
    no students AND main lecture lec.
    Pre: n a string, ls list,
    lec a Lecture"""
    Course.__init__(self,n,ls)
    self.mainlecture = lec
```

```
def add(self,n):
    """If student with netID n is not in
    roster of section,  add student to
    this section AND the main lecture.
    Do nothing if already there.
    Precondition: n is a valid netID."""
    # Calls old version of add to
    # add to roster
    Course.add(self,n)
    # Add to lecture roster
    self.mainlecture.add(n)
```

Diagramming Subclasses



Important Details:

- Draw a line from subclass to the parent class
- Do not duplicate inherited methods and attributes
- Include initializer and operators with methods
- Class attributes are a box with (current) value

Modified Fall 2015

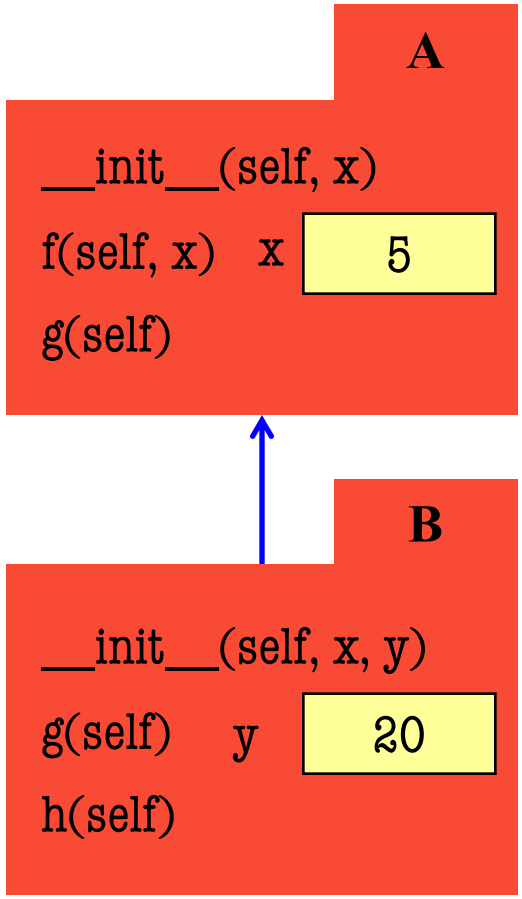
```
class A(object):
    x = 5
    def __init__(self, x):
        self.y = x
    def f(self, x):
        self.x = x
    def g(self):
        return self.x+self.y
```

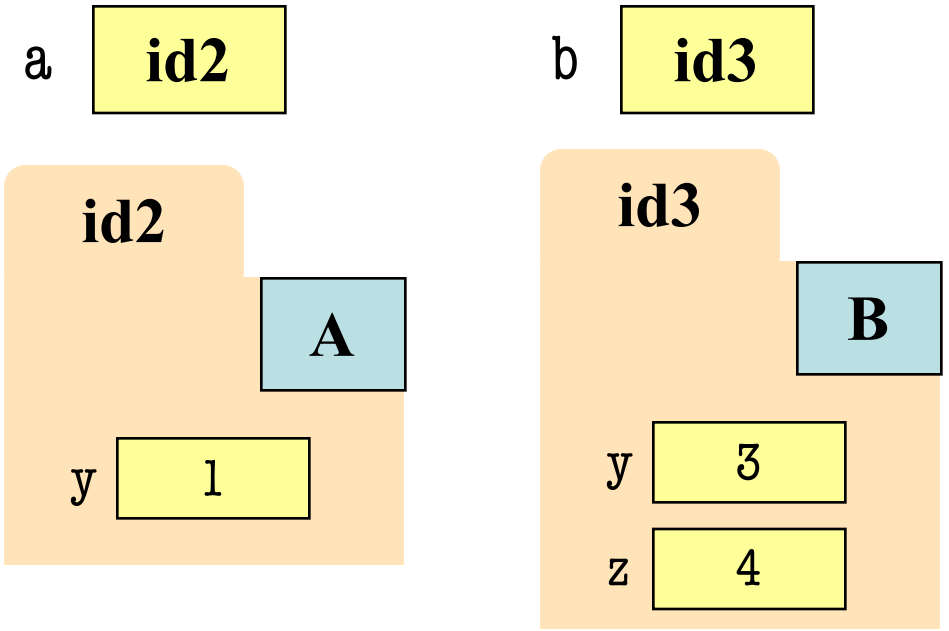
```
class B(A):
    y = 20
    def __init__(self, x, y):
        self.y = 42
        A.__init__(self, x)
        self.z = y
    def g(self):
        return self.y*self.x
    def h(self):
        self.f(self.z)
        return self.g()
```

Draw the folders and global variables after executing:

```
>>> a = A(1)
>>> b = B(3,4)
```

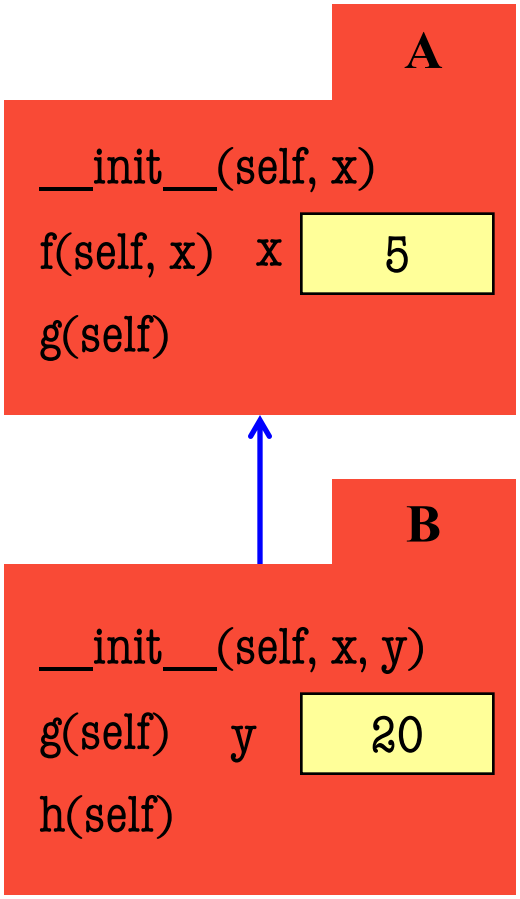
Modified Fall 2015

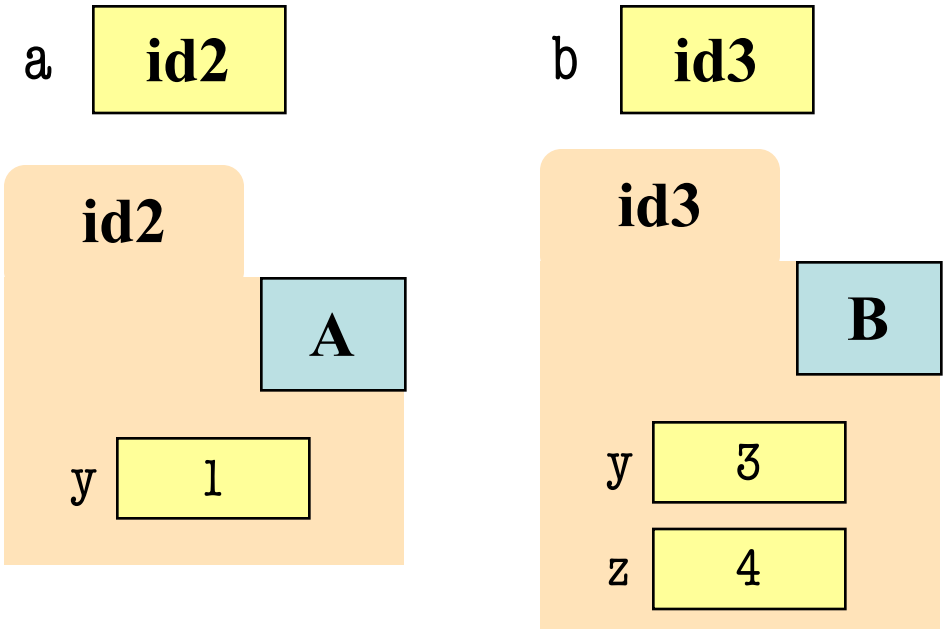





```
>>> a = A(1)
>>> b = B(3,4)
```

Modified Fall 2015





What is…

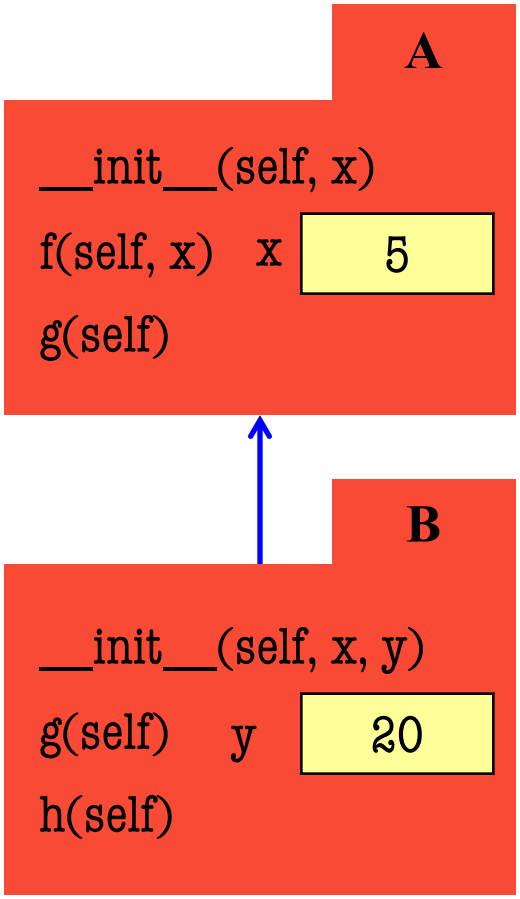
(1) a.y

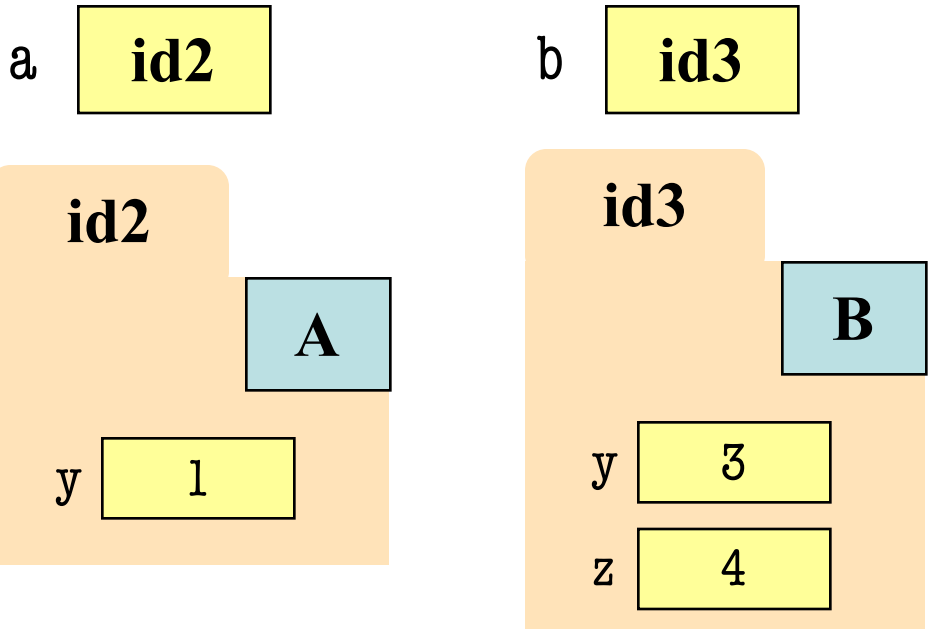
(2) b.x

(3) b.y

(4) B.y

Modified Fall 2015





What is…

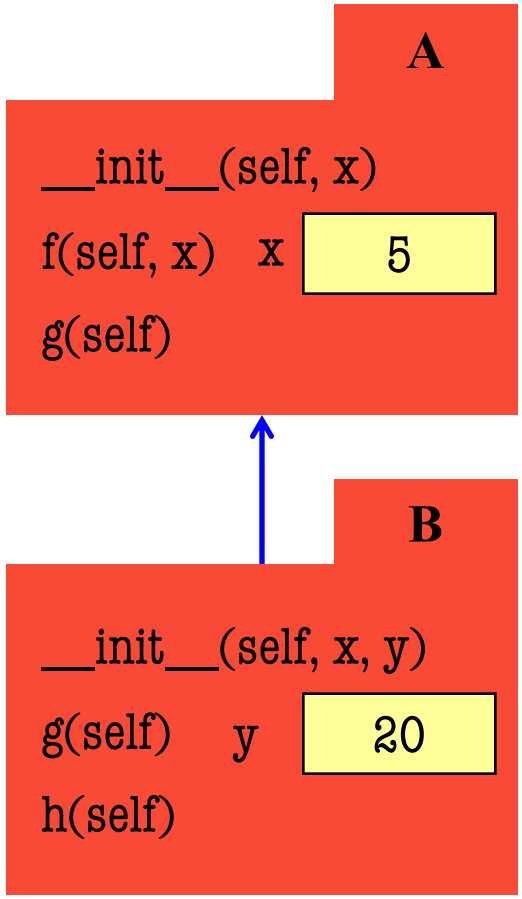
(1) a.y 1

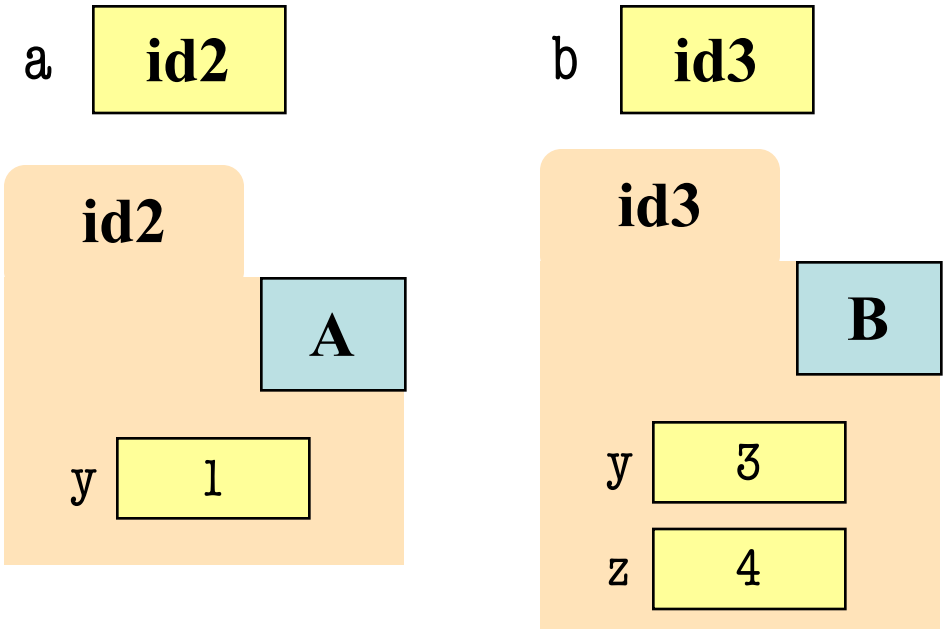
(2) b.x 5

(3) b.y 3

(4) B.y 20

Modified Fall 2015





What is…

(1) a.g()

(2) b.g()

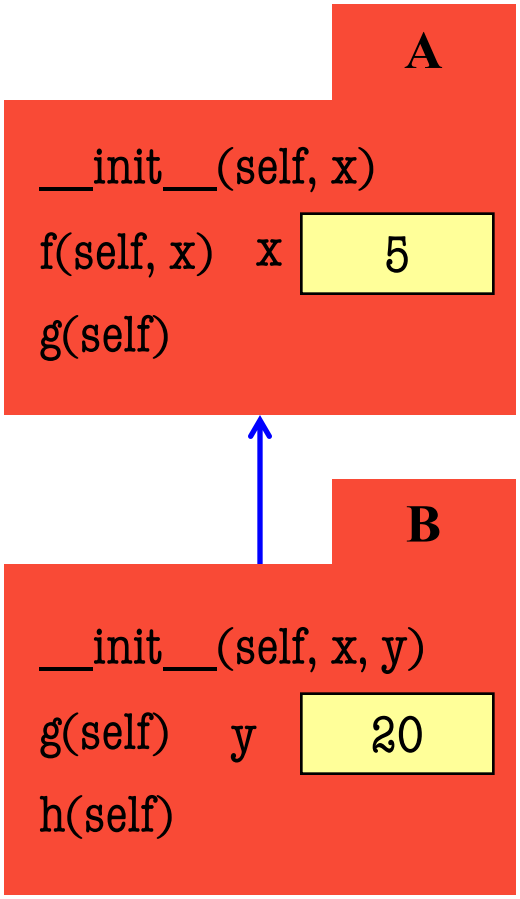
(3) A.g(b)

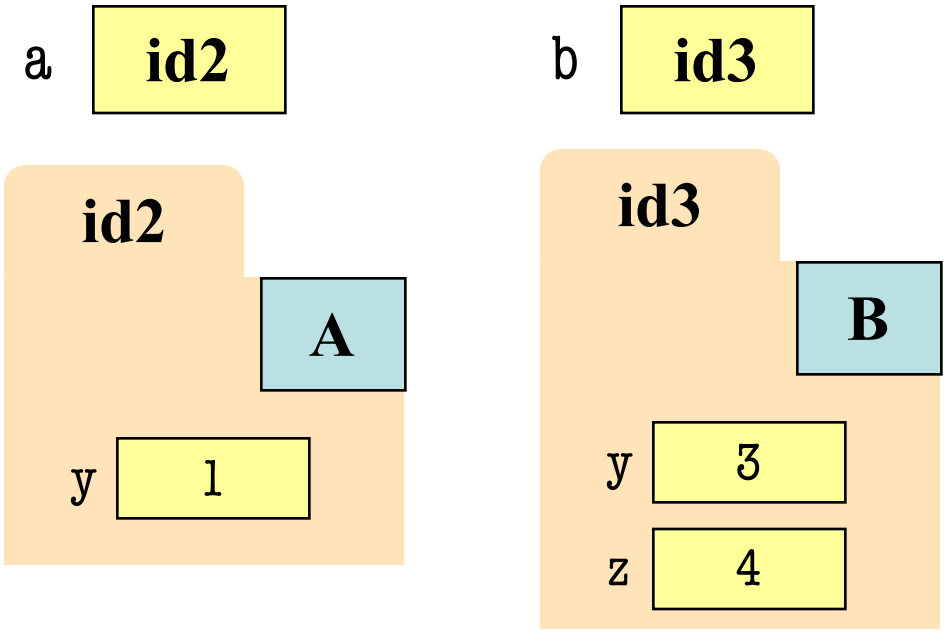
(4) b.h()

(5) a.f

(6) a.z

Modified Fall 2015





What is…

(1) a.g() 6

(2) b.g() 15

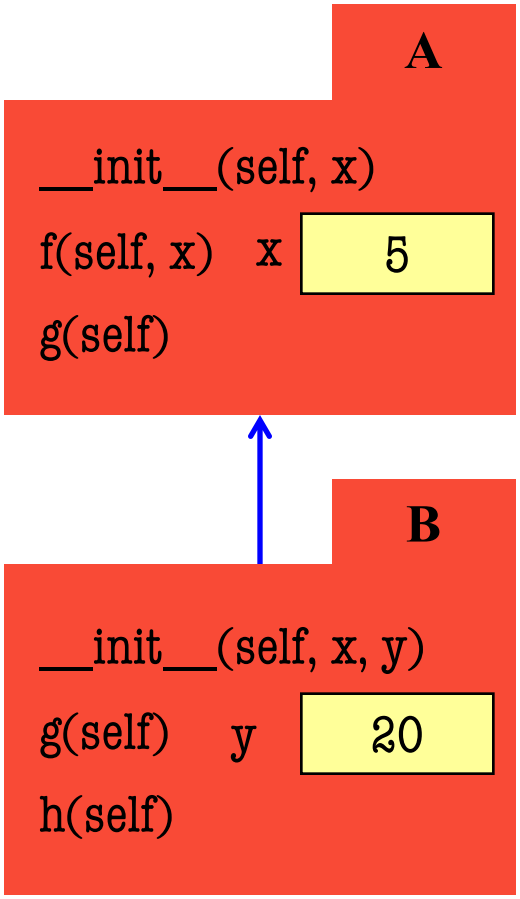
(3) A.g(b) 8

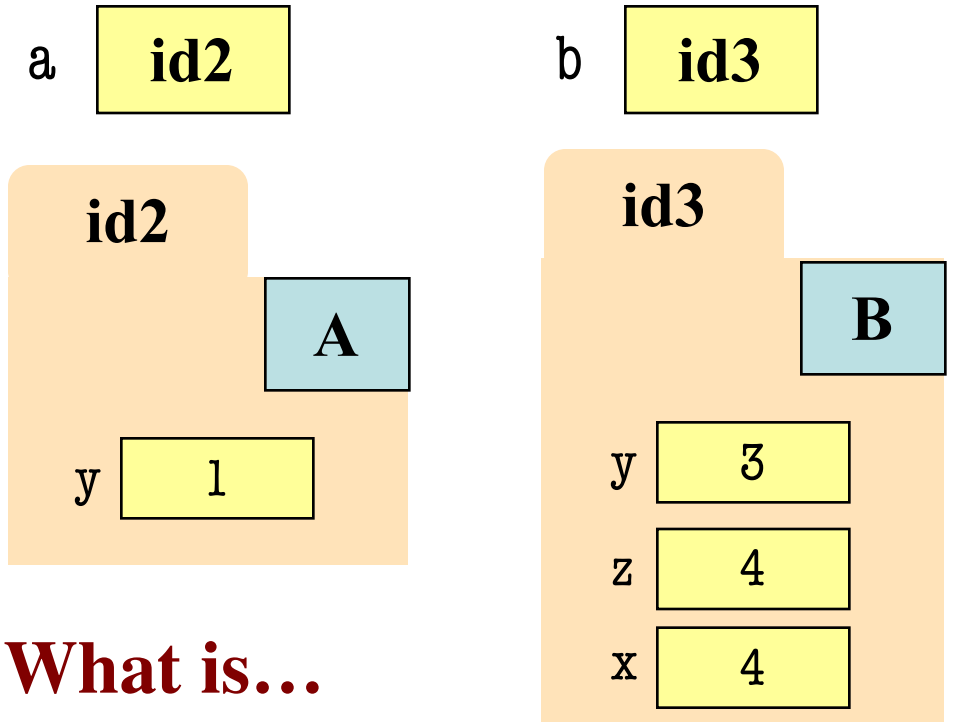
(4) b.h()

(5) a.f

(6) a.z

After executing b.h()





What is…

(1) a.g() 6

(2) b.g() 15

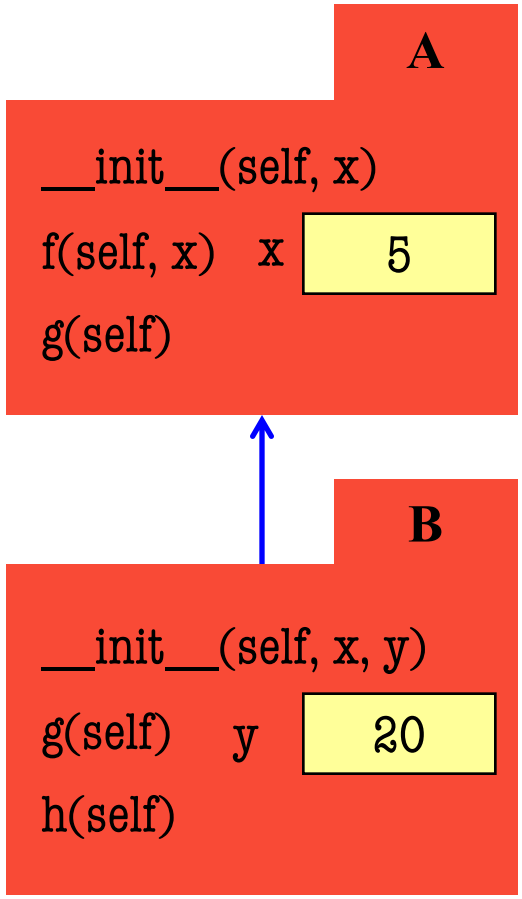
(3) A.g(b) 8

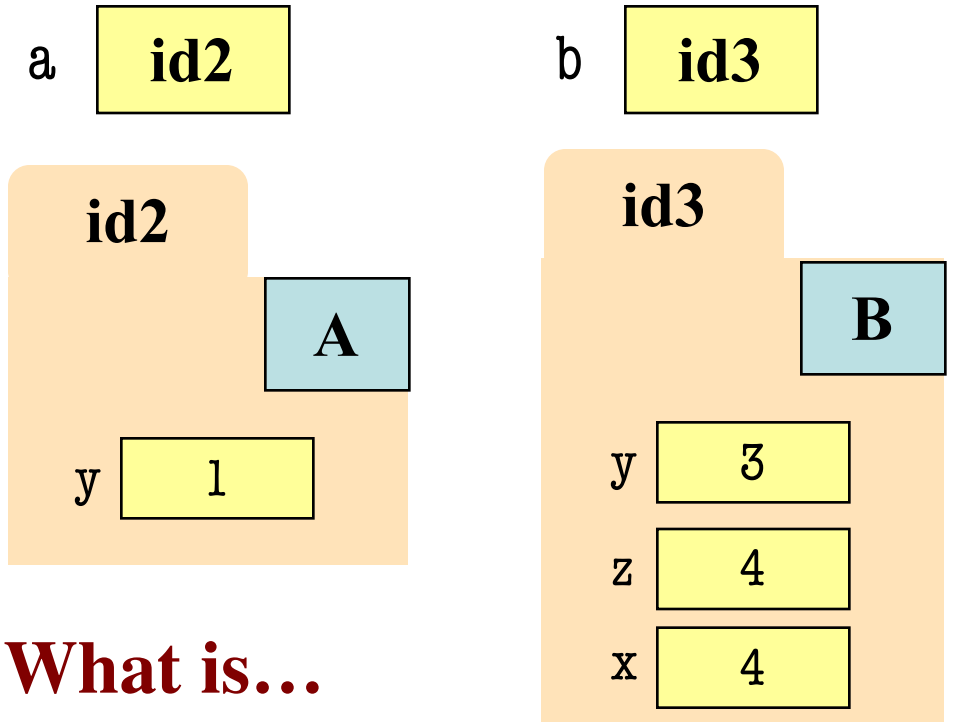
(4) b.h() 12

(5) a.f

(6) a.z

Modified Fall 2015





What is…

(1) a.g() 6

(2) b.g() 15

(3) A.g(b) 8

(4) b.h() 12

(5) a.f method

(6) a.z Error

Good Luck!