Review 2

Classes and Subclasses

Class Definition

```
class <name>(<optional superclass>):
    """Class specification"""
    class variables (format: Class.variable)
    initializer (__init__)
    special method definitions
    other method definitions
```

Class type to extend

- Every class must extend *something*
- Most classes extend `object` implicitly

Attribute Invariants

- Attribute invariants are important for programmer
  - Should look at them while writing methods
  - Anyone reading the code will understand how the class works
- Constructors initialize the attributes to satisfy invariants
  - Can use assert statements to enforce invariants

```
class Point(object):
    """An instance is a 3D point in space
        x: the x value of the point [float]
        y: the y value of the point [float]
        z: the z value of the point [float] """
```

Constructors

- Function that creates new *instances* of a class
- Constructor and class share the same name
- Creates object folder, initializes attributes, returns ID

```
class Point(object):
    ...
    def __init__(self, x, y, z):
        """Initializer: makes a Point object with x, y, z values"""
        self.x = x
        self.y = y
        self.z = z
```

Special Methods

- Start/end with underscores
  - __init__ for initializer
  - __str__ for str()
  - __repr__ for repr()
- Predefined by Python
  - You are overriding them
  - Defines a new behavior

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

    def __init__(self, x, y, z):
        """Initializer: makes new Point"""
        ...

    def __str__(self):
        """Returns: string with contents"""
        ...

    def __repr__(self):
        """Returns: unambiguous string"""
        ...
```

Operator Overloading

- Methods for operators
  - __add__ for +
  - __mul__ for *
  - __mod__ for %
  - __eq__ for ==
  - __lt__ for <
- Can then directly use the operators on objects
  - p1 == p2
  - Difference between == and is?

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

    def __add__(self, p):
        """Adds two points together"""
        …
    def __mul__(self, p):
        """Multiplies two points together"""
        …
    def __eq__(self, p):
        """Returns: whether two points are
        equivalent"""
```

Writing and Calling Methods

- Must include the keyword self to reference each individual instance
- Call the method with the object in front
  - <object>.<method>(<args>)
  - p1.quadrant()
  - dist = p1.distance(p2)
  - Object **is** the argument for the parameter self

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

    def __init__(self, x, y, z):
        """Initializer: makes new Point"""
        …

    def quadrant(self):
        """Returns: the quadrant occupied
            by the point"""

    def distance(self, p):
        """Returns: the distance between
            two points"""
```

Optional Arguments

- Can assign default values for method's parameters
  - Instead of just writing the parameter, put an assignment
  - Calling method without an argument for that
- Examples using first init
  - p = Point() #(0, 0, 0)
  - p = Point(1, 2) #(1, 2, 0)
  - p = Point(y=3, z=4) #(0, 3, 4)

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

    def __init__(self, x=0, y=0, z=0):
        """Initializer: makes new Point"""
        …
```

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

    def __init__(self, x, y, z=0):
        """Initializer: makes new Point"""
        …
```

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, also 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`.
  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 without duplicates
                         [nonempty list of string]
        roster: student net-ids
                   [list of string, can be 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 non-empty 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 non-empty 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; 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__
```

Subclasses

- Subclass conceptually is a subgroup of its parent class
  - Cat and Dog are both Animals, but are distinct
- Inherits **attributes** and **methods** of parent class
  - Can include additional ones that are unique to subclass
  - Overrides methods such as __init__ to add functionality
  - When looking for an attribute/method, will resolve in the name in the following order (`object` is built-in class):

  object → class → parent class → parent of parent → `object`
- `isinstance(<obj>, <class>)`
  - True if <obj>'s class is <class> or is a subclass of <class>
  - `isinstance(p, Point)`

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.
- 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: name is a string, ls is a nonempty list"""
        super().__init__(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: name 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: name a string, ls list,
    lec a Lecture"""
    super().__init__(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
    super().add(self, n)
    # Add to lecture roster
    self.mainlecture.add(n)
```

Two Example Classes

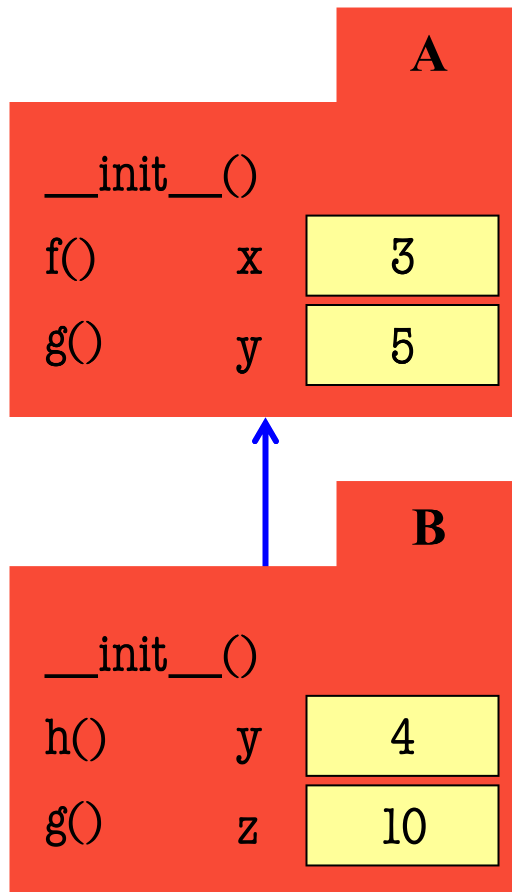
```
class A(object):
    x=3
    y=5
    def __init__(self,y):
        self.y = y

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

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

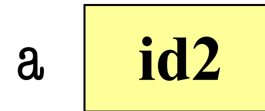
```
class B(A):
    y=4
    z=10
    def __init__(self,x,y):
        self.x = x
        self.y = y

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

    def h(self):
        return 42
```

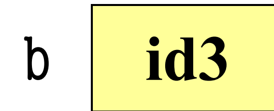
Execute:

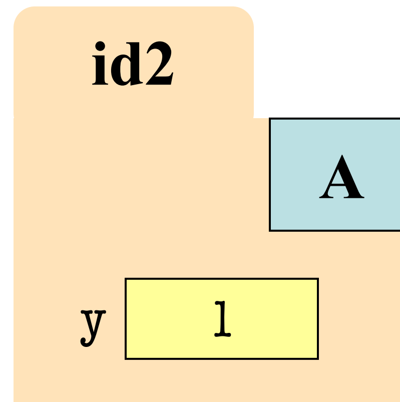
```
>>> a = A(1)
>>> b = B(7,3)
```

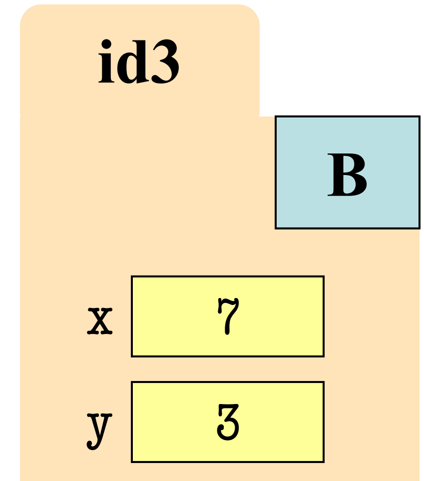
Example from Fall 2013







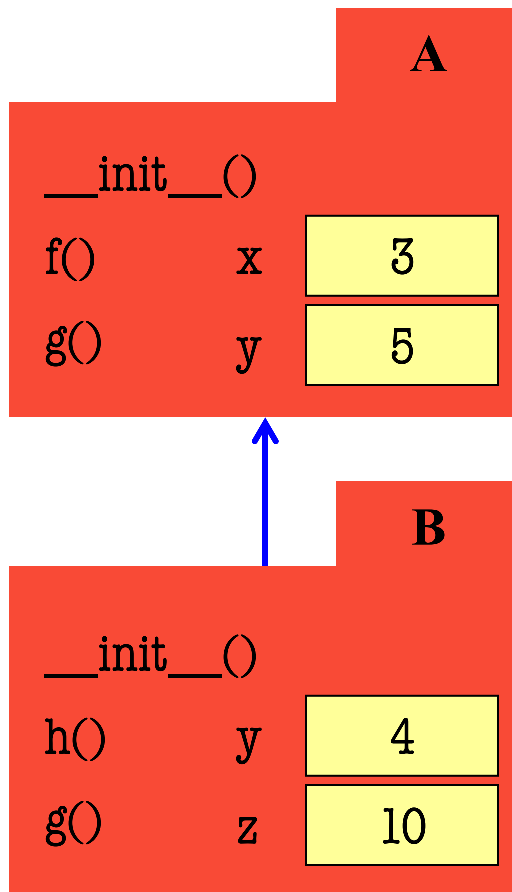




Execute:

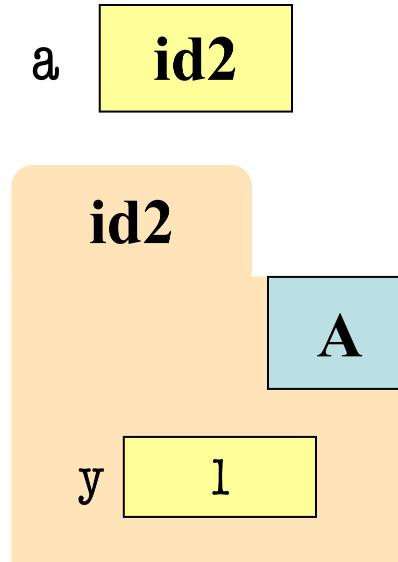
```
>>> a = A(1)
>>> b = B(7,3)
```

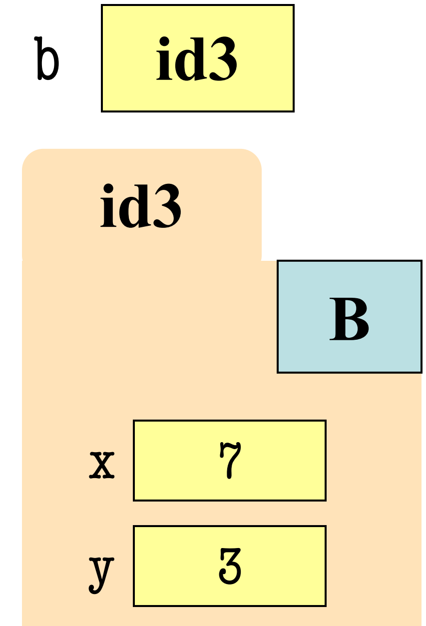
Example from Fall 2013



a id2

b id3





What is…

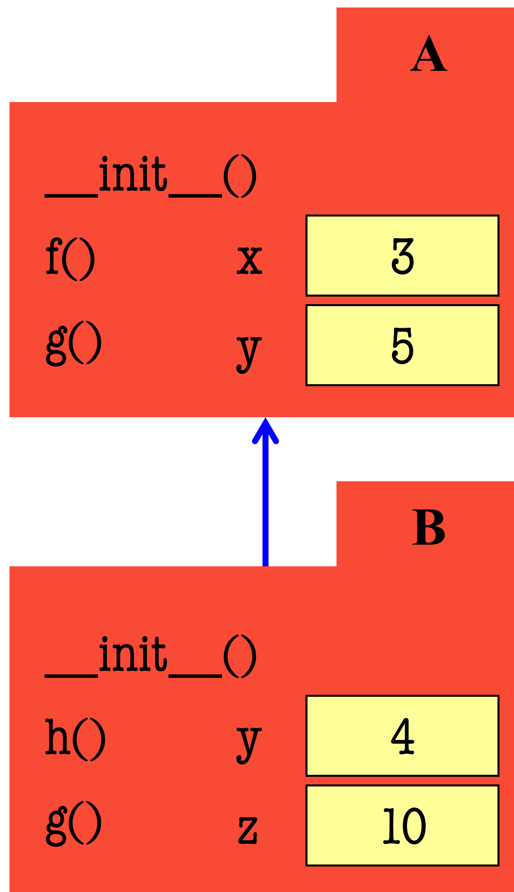
(1) a.y

(2) a.z

(3) b.x

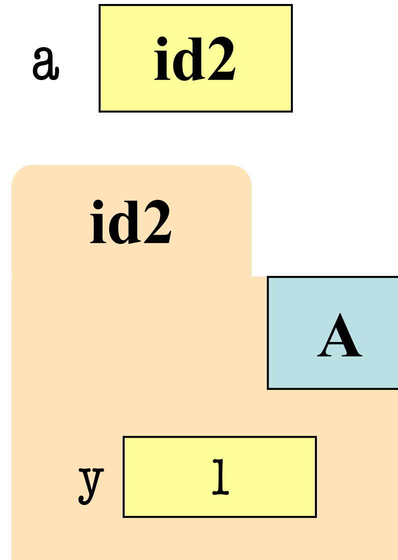
(4) B.x

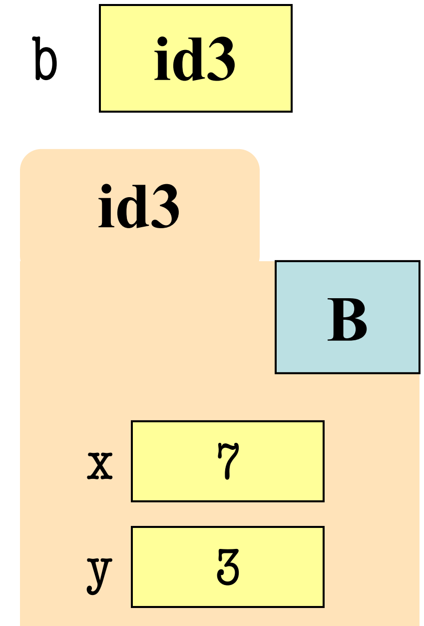
Example from Fall 2013



a id2

b id3





What is…

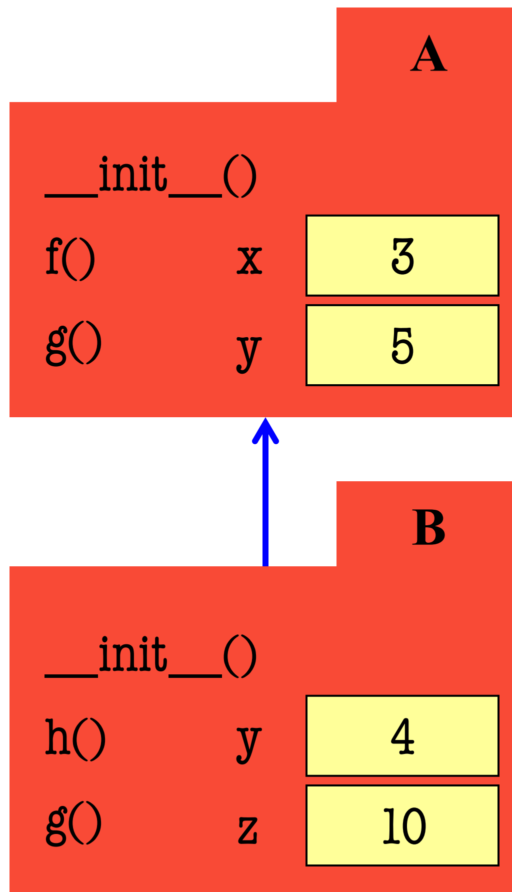
(1) a.y 1

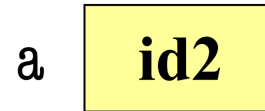
(2) a.z ERROR

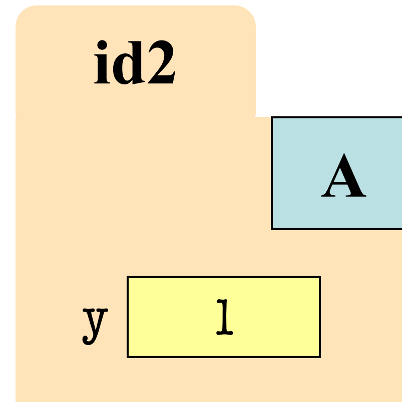
(3) b.x 7

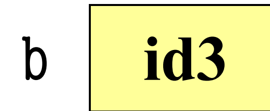
(4) B.x 3

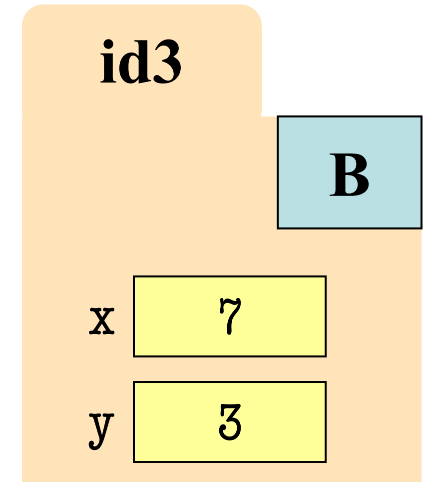
Example from Fall 2013











What is…

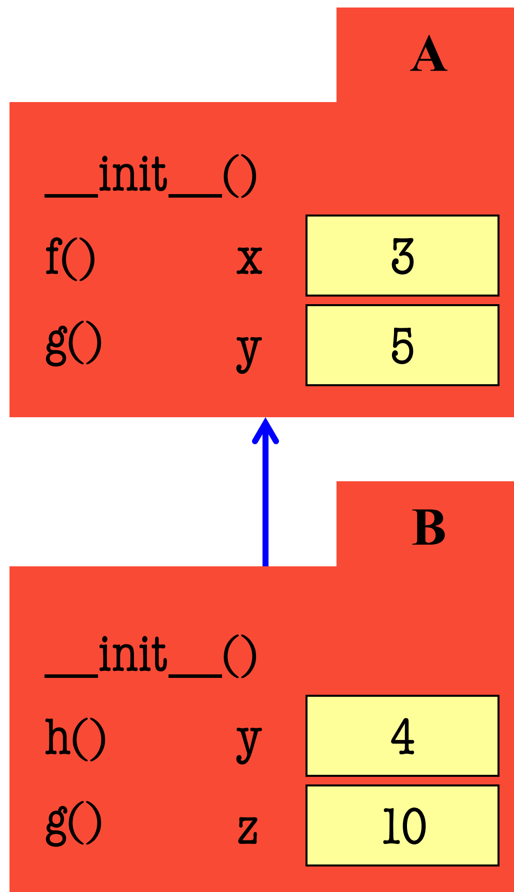
(1) a.f()

(2) a.h()

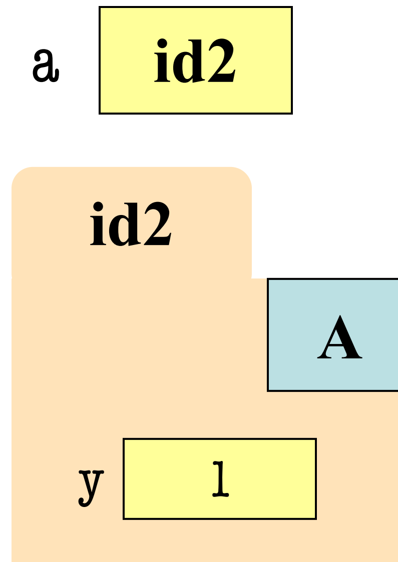
(3) b.f()

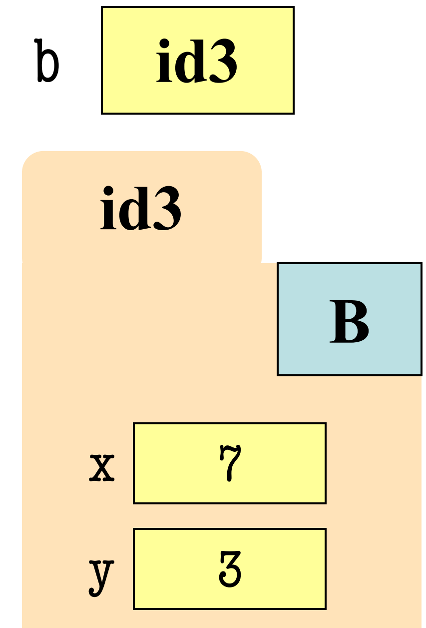
(4) b.g()

Example from Fall 2013



a id2 b id3





What is…

(1) a.f() 4 (2) a.h() ERROR

(3) b.f() 17 (4) b.g() 17