More Object-Oriented Programming

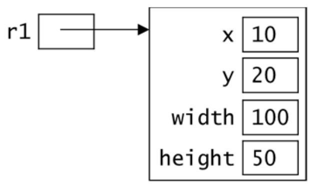
Computer Science 111
Boston University

Vahid Azadeh Ranjbar , Ph.D.

based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College

Recall: Our `Rectangle` Class

```
# rectangle.py

class Rectangle:
    def __init__(self, init_width, init_height):
        self.x = 0
        self.y = 0
        self.width = init_width
        self.height = init_height

    def grow(self, dwidth, dheight):
        self.width += dwidth
        self.height += dheight

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2*self.width + 2*self.height

    def scale(self, factor):
        self.width *= factor
        self.height *= factor
```



Original Client Program...

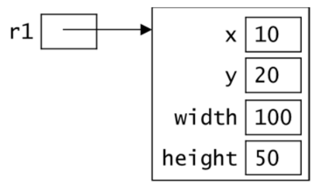
```
from rectangle import *

# construct two Rectangle objects
r1 = Rectangle(100, 50)
r2 = Rectangle(75, 350)

# print dimensions and area of each
print('r1:', r1.width, 'x', r1.height)
area1 = r1.width * r1.height
print('area =', area1)

print('r2:', r2.width, 'x', r2.height)
area2 = r2.width * r2.height
print('area =', area2)

# grow both Rectangles
r1.width += 50
r1.height += 10
r2.width += 5
r2.height += 30

# print new dimensions
print('r1:', r1.width, 'x', r1.height)
print('r2:', r2.width, 'x', r2.height)
```

Simplified Client Program

```
from rectangle import *

# construct two Rectangle objects
r1 = Rectangle(100, 50)
r2 = Rectangle(75, 350)

# print dimensions and area of each
print('r1:', r1.width, 'x', r1.height)
print('area =', r1.area())

print('r2:', r2.width, 'x', r2.height)
print('area =', r2.area())

# grow both Rectangles
r1.grow(50, 10)
r2.grow(5, 30)

# print new dimensions
print('r1:', r1.width, 'x', r1.height)
print('r2:', r2.width, 'x', r2.height)
```

Recall: Our `Rectangle` Class

```
# rectangle.py

class Rectangle:
    def __init__(self, init_width, init_height):
        self.x = 0
        self.y = 0
        self.width = init_width
        self.height = init_height

    def grow(self, dwidth, dheight):
        self.width += dwidth
        self.height += dheight

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2*self.width + 2*self.height

    def scale(self, factor):
        self.width *= factor
        self.height *= factor
```



What is the output of this program?

```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

r1.scale(2)
print(r1.width, r2.width, r3.width)
```

A. 40 40 40

B. 80 40 40

C. 80 40 80

D. 80 80 80

E. none of these

What is the output of this program?

```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

r1.scale(2)
print(r1.width, r2.width, r3.width)
```

A. `40 40 40`

B. `80 40 40`

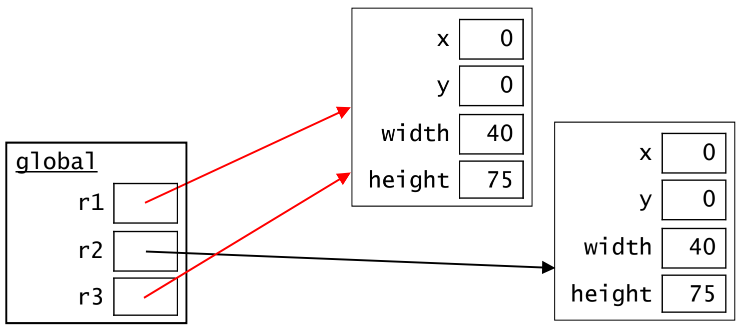
C. `80 40 80`

D. `80 80 80`

E. none of these

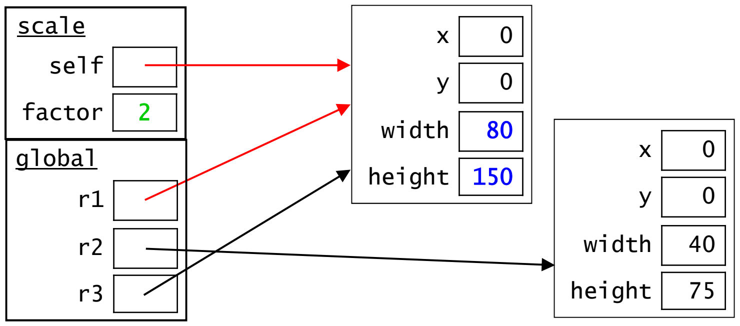
What is the output of this program?

```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

r1.scale(2)
print(r1.width, r2.width, r3.width)
```



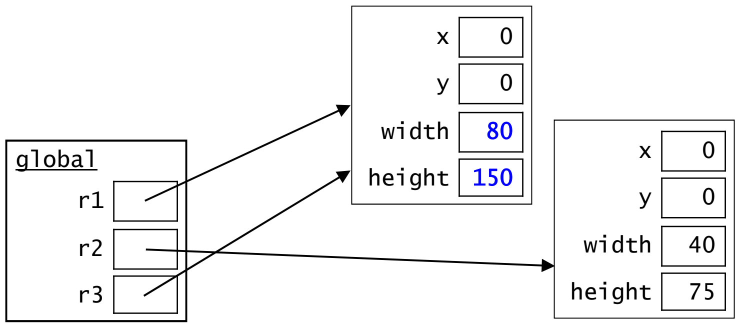
What is the output of this program?

```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

r1.scale(2)
print(r1.width, r2.width, r3.width)
```



What is the output of this program?

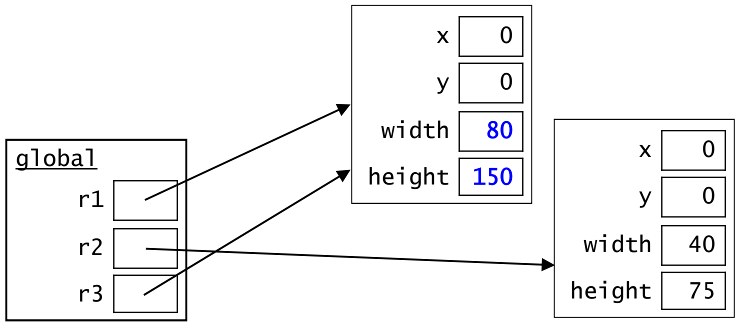
```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

r1.scale(2)   # changes are still inside the object!
print(r1.width, r2.width, r3.width)
```



What is the output of this program?

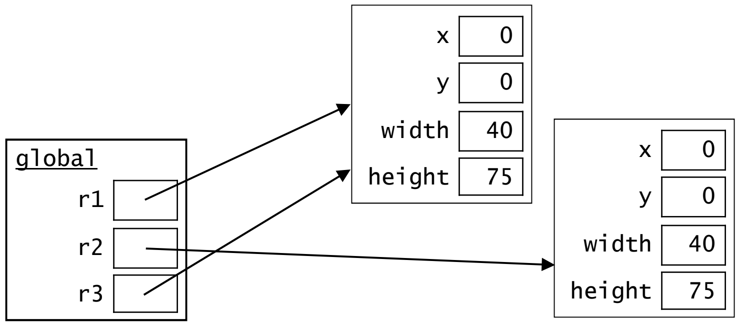
```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

r1.scale(2)
print(r1.width, r2.width, r3.width)
```

output: 80 40 80



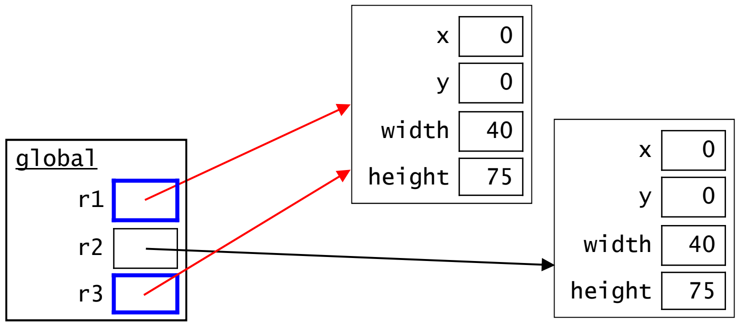
What about this program?

```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

print(r1 == r2)
print(r1 == r3)
```



What is the output of this client program?

```
from rectangle import *

r1 = Rectangle(40, 75)
r2 = Rectangle(40, 75)
r3 = r1

print(r1 == r2)        # outputs False
print(r1 == r3)        # outputs True
```



__eq__ (Implementing Our Own ==)

- The __eq__ method of a class allows us to implement our own version of the == operator.
- If we don't write a __eq__ method for a class, we get a default version that compares the object's memory addresses
    - see the previous example!

__eq__ Method for Our Rectangle Class

```
class Rectangle:
    ...         r1     r2
    def __eq__(self, other):
        if self.width == other.width and
           self.height == other.height:
            return True
        else:
            return False
```

```
>>> r1 = Rectangle(40, 75)
>>> r2 = Rectangle(40, 75)

>>> print(r1 == r2)
True
```

__repr__ (Printing/Evaluating an Object)

- The __repr__ method of a class returns a string representation of objects of that class.
- It gets called when you:
  - evaluate an object in the Shell:
    ```
    >> r1 = Rectangle(100, 80)
    >> r1                        # calls __repr__
    ```
  - apply str():
    ```
    >> r1string = str(r1)        # also calls __repr__
    ```
  - print an object:
    ```
    >> print(r1)                 # also calls __repr__
    ```

`__repr__` (Printing/Evaluating an Object)

- If we don't write a `__repr__` method for a class, we get a default version that isn't very helpful!

```
>>> r2 = Rectangle(50, 20)
>>> r2
<__main__.Rectangle object at 0x03247C30>
```

`__repr__` Method for Our `Rectangle` Class

```
class Rectangle:
    ...
    def __repr__(self):
        return str(self.width) + ' x ' + str(self.height)
```

- Note: the method does *not* do any printing.
- It returns a string that can then be printed or used when evaluating the object:

```
>>> r2 = Rectangle(50, 20)

>>> print(r2)
50 x 20

>>> r2
50 x 20
```

Updated Rectangle Class

```
class Rectangle:
    def __init__(self, init_width, init_height):
        self.x = 0
        self.y = 0
        self.width = init_width
        self.height = init_height

    def grow(self, dwidth, dheight):
        self.width += dwidth
        self.height += dheight

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2*self.width + 2*self.height

    def scale(self, factor):
        self.width *= factor
        self.height *= factor

    def __eq__(self, other):
        if self.width == other.width and self.height == other.height:
            return True
        else:
            return False

    def __repr__(self):
        return str(self.width) + ' x ' + str(self.height)
```

Simplifying the Client Program Again...

```
from rectangle import *

# Construct two Rectangle objects
r1 = Rectangle(100, 50)
r2 = Rectangle(75, 350)

# Print dimensions and area of each
print('r1:', r1.width, 'x', r1.height)
print('area =', r1.area())

print('r2:', r2.width, 'x', r2.height)
print('area =', r2.area())

# grow both Rectangles
r1.grow(50, 10)
r2.grow(5, 30)

# Print new dimensions
print('r1:', r1.width, 'x', r1.height)
print('r2:', r2.width, 'x', r2.height)
```

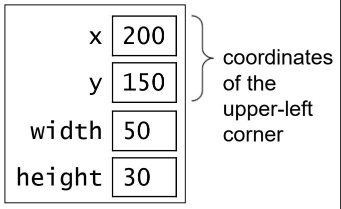
Simplifying the Client Program Again...

```
from rectangle import *

# Construct two Rectangle objects
r1 = Rectangle(100, 50)
r2 = Rectangle(75, 350)

# Print dimensions and area of each
print('r1:', r1)
print('area =', r1.area())

print('r2:', r2)
print('area =', r2.area())

# grow both Rectangles
r1.grow(50, 10)
r2.grow(5, 30)

# Print new dimensions
print('r1:', r1)
print('r2:', r2)
```

More Practice Defining Methods

- Write a method that moves the rectangle to the right by some amount.
  - sample call: `r.move_right(30)`

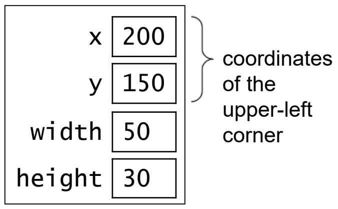
```
def move_right(self, ________):
```



- Write a method that determines if the rectangle is a square.
  - return `True` if it does, and `False` otherwise
  - sample call: `r1.is_square()`

More Practice Defining Methods

- Write a method that moves the rectangle to the right by some amount.
  - sample call: `r.move_right(30)`

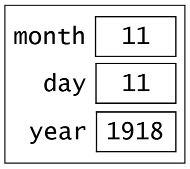


```
def move_right(self, amount):
    self.x += amount

    # do we need to return something?
    # no! the changes will still be in the object
    # after the method returns!
```

- Write a method that determines if the rectangle is a square.
  - return `True` if it does, and `False` otherwise
  - sample call: `r1.is_square()`

```
def is_square(self):
    if self.width == self.height:
        return True
    else:
        return False
```

PS 8: `Date` Class



```
class Date:
    def __init__(self, new_month, new_day, new_year):
        """ constructor that initializes the
            three attributes
        """
        # you will write this!

    def __repr__(self):
        """This method returns a string representation for the
           object of type Date that calls it (named self).
        """
        s =  "%02d/%02d/%04d" % (self.month, self.day, self.year)
        return s

    def is_leap_year(self):
        """ Returns True if the calling object is
            in a leap year. Otherwise, returns False.
        """
        if self.year % 400 == 0:
            return True
        elif self.year % 100 == 0:
            return False
        elif self.year % 4 == 0:
            return True
        return False
```

Date Class (cont.)

- Example of how `Date` objects can be used:

```
>>> d = Date(12, 31, 2014)
>>> print(d)          # calls __repr__
12/31/2014
>>> d.advance_one()   # a method you will write
                      # nothing is returned!
>>> print(d)          # d has been changed!
01/01/2015
```

Methods Calling Other Methods

```
class Date:
    ...
    def advance_one(self):
        """ moves the date ahead 1 day """

        days_in_month = [0,31,28,31,30,31,30,31,31,30,31,30,31]
        if self.is_leap_year() == True:
            days_in_month[2] = 29

        # adjust the attributes
```

- The object calls `is_leap_year()` on itself!

Another Method You Will Add...

```
class Date:
    ...
    def is_before(self, other):    # buggy version!
        """ returns True if the called Date object (self)
            occurs before other, and False otherwise.
        """
        if self.year < other.year:
            return True
        elif self.month < other.month:
            return True
        elif self.day < other.day:
            return True
        else:
            return False
```

Which call(s) does the method get wrong?

```
class Date:
    ...
    def is_before(self, other):    # buggy version!
        """ returns True if the called Date object (self)
            occurs before other, and False otherwise.
        """
        if self.year < other.year:
            return True
        elif self.month < other.month:
            return True
        elif self.day < other.day:
            return True
        else:
            return False

d1 = Date(11, 10, 2014)
d2 = Date(1, 1, 2015)
d3 = Date(1, 15, 2014)
```

Extra: Can you think of any *other* cases that it would get wrong involving these dates?

A. `d1.is_before(d2)`

B. `d2.is_before(d1)`

C. `d3.is_before(d1)`

D. more than one

Which call(s) does the method get wrong?

```
class Date:
    ...
    def is_before(self, other):    # buggy version!
        """ returns True if the called Date object (self)
            occurs before other, and False otherwise.
        """
        if self.year < other.year:     2015 < 2014 (False)
            return True
        elif self.month < other.month: 1 < 11 (True)
            return True    # not the correct return value!
        elif self.day < other.day:
            return True
        else:
            return False

d1 = Date(11, 10, 2014)
d2 = Date(1, 1, 2015)
d3 = Date(1, 15, 2014)
```

A. `d1.is_before(d2)`

B. `d2.is_before(d1)`

C. `d3.is_before(d1)`

D. more than one

Which call(s) does the method get wrong?

```
class Date:
    ...
    def is_before(self, other):    # buggy version!
        """ returns True if the called Date object (self)
            occurs before other, and False otherwise.
        """
        if self.year < other.year:
            return True
        elif self.month < other.month and...:
            return True
        elif self.day < other.day and...:
            return True
        else:
            return False

d1 = Date(11, 10, 2014)
d2 = Date(1, 1, 2015)
d3 = Date(1, 15, 2014)
```

Extra: Can you think of any *other* cases that it would get wrong involving these dates?

A. `d1.is_before(d2)`

B. `d2.is_before(d1)`

C. `d3.is_before(d1)`

D. more than one

Which call(s) does the method get wrong?

```
class Date:
    ...
    def is_before(self, other):    # buggy version!
        """ returns True if the called Date object (self)
            occurs before other, and False otherwise.
        """
        if self.year < other.year:
            return True
        elif self.month < other.month and...:
            return True
        elif self.day < other.day and...:
            return True
        else:
            return False

d1 = Date(11, 10, 2014)
d2 = Date(1, 1, 2015)
d3 = Date(1, 15, 2014)
```

Extra: Can you think of any *other* cases that it would get wrong involving these dates?

`d1.is_before(d3)`
`d2.is_before(d3)`

A. `d1.is_before(d2)`

B. `d2.is_before(d1)`

C. `d3.is_before(d1)`

D. more than one