Lecture 11
More Object-Oriented Programming

Unrelated but do you see how happy dogs make Milla???

*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
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)
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?

```python
from rectangle import *

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

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

```
height 75
width 40
y 0
x 0
global
r1
r2
r3
```
What is the output of this program?

```python
from rectangle import *

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

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

```
width  80
height 150
```

```
x  0
y  0
width 40
height 75
```
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?

```python
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?

```python
from rectangle import *

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

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

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

```
What is the output of this client program?

```python
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:
    ```python
    >> r1 = Rectangle(100, 80)
    >> r1                    # calls __repr__
    ```
  • apply str():
    ```python
    >> r1string = str(r1)    # also calls __repr__
    ```
  • print an object:
    ```python
    >> 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!

```python
generate code if needed
```

```python
>>> r2 = Rectangle(50, 20)
>>> r2
<__main__.Rectangle object at 0x03247C30>
```
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
class Rectangle:
    def __init__(self, init_width, 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
        return False

    def __repr__(self):
        return str(self.width) + ' x ' + str(self.height)
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)
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 is, 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:  \texttt{r.move\_right(30)}

```python
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 is, and False otherwise
  • sample call:  \texttt{r1.is\_square()}

```python
def is\_square(self):
    if self.width == self.height:
        return True
    else:
        return False
```
The 4 Pillars of OOP

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

This is a common interview question for software developers!
1. **Encapsulation**

Grouping related functions and variables together into objects

---

#### String

<table>
<thead>
<tr>
<th>contents</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

#### Rectangle

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>width</th>
<th>height</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

#### Methods

- `upper()`
- `lower()`
- `find()`
- `count()`
- `replace()`
- `split()`
- `area()`
- `perimeter()`
The 4 Pillars of OOP

2. Abstraction

Hiding code complexity from an object’s interface

Before OOP
area1 = r1.width * r1.height
area2 = r2.width * r2.height
area3 = r3.width * r3.height

After OOP
area1 = r1.area()
area2 = r2.area()
area3 = r3.area()

What if we wanted to change how area is calculated?

Users of Rectangle don’t care how the area is calculated, only that it is
3. Inheritance

Reducing redundant code by combining common features

Rectangle
+ area()
+ perimeter()

Square
+ area()
+ perimeter()

Where do we define the area & perimeter of a Square?

We will cover inheritance next class
4. **Polymorphism**

   Enabling a common interface for many different data types

**Before OOP**

<table>
<thead>
<tr>
<th></th>
<th>Triangle</th>
<th>Rectangle</th>
<th>Pentagon</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1 =</td>
<td>area_triangle(b,h)</td>
<td>area_triangle(b,h)</td>
<td>area_triangle(a)</td>
</tr>
<tr>
<td>a2 =</td>
<td>area_rectangle(w,h)</td>
<td>area_rectangle(w,h)</td>
<td>area_rectangle(a)</td>
</tr>
<tr>
<td>a3 =</td>
<td>area_pentagon(a)</td>
<td>area_pentagon(a)</td>
<td>area_pentagon(a)</td>
</tr>
</tbody>
</table>

**After OOP**

<table>
<thead>
<tr>
<th></th>
<th>tri.area()</th>
<th>rec.area()</th>
<th>pen.area()</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1 =</td>
<td>tri.area()</td>
<td>rec.area()</td>
<td>pen.area()</td>
</tr>
<tr>
<td>a2 =</td>
<td>tri.area()</td>
<td>rec.area()</td>
<td>pen.area()</td>
</tr>
<tr>
<td>a3 =</td>
<td>tri.area()</td>
<td>rec.area()</td>
<td>pen.area()</td>
</tr>
</tbody>
</table>
The 4 Pillars of OOP

What are the main benefits of these OOP concepts?

1. **Encapsulation**
   Reduces complexity and improves reusability

2. **Abstraction**
   Reduces complexity and minimizes impact of changes

3. **Inheritance**
   Eliminates redundant code

4. **Polymorphism**
   Simplifies object interfaces
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Hiding code complexity from an object’s interface

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Hiding code complexity from an object’s interface

**Before OOP**
- area1 = r1.width * r1.height
- area2 = r2.width * r2.height
- area3 = r3.width * r3.height

**After OOP**
- area1 = r1.area()
- area2 = r2.area()
- area3 = r3.area()

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Enabling a common interface for many different data types

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Enabling a common interface for many different data types

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Reducing redundant code by combining common features

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Reducing redundant code by combining common features

A. Encapsulation  
B. Abstraction  
C. Inheritance  
D. Polymorphism  
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Grouping related functions and variables together into objects

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
The 4 Pillars of OOP

Which of the pillars matches the following definition?

Grouping related functions and variables together into objects

A. Encapsulation
B. Abstraction
C. Inheritance
D. Polymorphism
E. None of the above
class Date:
    def __init__(self, new_month, new_day, new_year):
        """Constructor""
        self.month = new_month
        self.day = new_day
        self.year = new_year

    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:

```python
>>> d = Date(12, 31, 2014)
>>> print(d) # calls __repr__
12/31/2014
>>> d.tomorrow() # a method you will write
01/01/2015 # a new date is returned!
>>> print(d) # d has not been changed
12/31/2015
```
Methods Calling Other Methods

class Date:
...

def incrementDay(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

    self.day += 1

    # advance month and year as needed
    if self.day ... 

    • 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

A. d1.is_before(d2)        C. d3.is_before(d1)
B. d2.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

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

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."""
        
        # buggy version!
        if self.year < other.year:
            return True
        elif self.month < other.month and...
        
        # buggy version!
        elif self.day < other.day and...
        
        # buggy version!
        else:
            return False

A. d1.is_before(d2)  C. d3.is_before(d1)
B. d2.is_before(d1)  D. more than one
I feel so objectified!