PRG – PROGRAMMING ESSENTIALS

Lecture 10 – Classes & Objects II

Milan Nemy

Czech Technical University in Prague,
Faculty of Electrical Engineering, Dept. of Cybernetics

https://beat.ciirc.cvut.cz/people/milan-nemy/

milan.nemy@cvut.cz

RECAP: OOP PERSPECTIVE

OOP is about changing the perspective

- Syntax for a function call: **function_name(variable)**
  function is the one who executes on the variable
- Syntax in OOP: **object_name.function_name()**
  object is the one who executes its method on given data / attribute

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html

RECAP: CLASS vs. TUPLE

```python
class Point:
    """ Create a new Point, at coordinates x, y """

    def __init__(self, x=0, y=0):
        """ Create a new point at x, y """
        self.x = x
        self.y = y

    def distance_from_origin(self):
        """ Compute my distance from the origin """
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5
```

- Advantage of using a class (*e.g. Point*) rather than a tuple is that **class methods are sensible operations** for points, but may not be appropriate for other tuples (*e.g. calculate the distance from the origin*)
- Class allows to **group together sensible operations** as well as data to apply the methods on
- Each instance of the class has its **own state**
- Method **behaves like a function** but it is invoked on a specific instance

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html

RECAP: CLASS METHODS

@classmethod

- In the same way class attributes are defined, which are shared between all instances of a class, class methods are defined using @classmethod **decorator** for ordinary method

- Class method still has its calling object as the first parameter, but by convention it is **cls** instead of **self**
- If class method is called from an instance, this parameter will contain the instance object, but if it is called from the class it will contain the class object
- Naming the parameter **cls** serves as reminder that it is not guaranteed to have any instance attributes

RECAP: CLASS METHODS

What are class methods good for?

- For tasks associated with a class utilizing constants and other class attributes without the need to create any class instances
- **EXAMPLE**: *when we write classes to group related constants together with functions which act on them – no need to instantiate these classes at all*

RECAP: EXAMPLE – INSTANCE METHODS

```
class Inst:

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

    def introduce(self):
        print("Hello, I am %s, and my name is " %(self, self.name))
```

```
myinst = Inst("Test Instance")
otherinst = Inst("An other instance")
myinst.introduce()
# outputs: Hello, I am <Inst object at x>, and my name is Test Instance
otherinst.introduce()
# outputs: Hello, I am <Inst object at y>, and my name is An other instance
```

https://stackoverflow.com/questions/17134653/difference-between-class-and-instance-methods

RECAP: EXAMPLE – CLASS METHODS

```
class Cls:

    @classmethod
    def introduce(cls):
        print("Hello, I am %s!" %cls)
```

```
Cls.introduce() # same as Cls.introduce(Cls)
# outputs: Hello, I am <class 'Cls'>
```

Notice that again `Cls` is passed hiddenly, so we could also say `Cls.introduce(Inst)` and get output `"Hello, I am <class 'Inst'>`. This is particularly useful when we're inheriting a class from `Cls`:

```
class SubCls(Cls):
    pass

SubCls.introduce()
# outputs: Hello, I am <class 'SubCls'>
```

https://stackoverflow.com/questions/17134653/difference-between-class-and-instance-methods

RECAP: STATICS METHODS

@staticmethod

- Static method does not have the calling object passed into it as the first parameter
- Static method does not have access to the rest of the class or instance
- Static method is most commonly called from class objects (*like class methods*)

RECAP: EXAMPLE – STATIC METHODS

```python
class Person:
    TITLES = ('Dr', 'Mr', 'Mrs', 'Ms')

    def __init__(self, name, surname):
        self.name = name
        self.surname = surname

    def fullname(self):  # instance method
        # instance object accessible through self
        return "%s %s" % (self.name, self.surname)

    @classmethod
    def allowed_titles_starting_with(cls, startswith):  # class method
        # class or instance object accessible through cls
        return [t for t in cls.TITLES if t.startswith(startswith)]

    @staticmethod
    def allowed_titles_ending_with(endswith):  # static method
        # no parameter for class or instance object
        # we have to use Person directly
        return [t for t in Person.TITLES if t.endswith(endswith)]
```

```
In[3]: jane = Person("Jane", "Smith")
In[4]: print(jane.fullname())
Jane Smith
In[5]: print(jane.allowed_titles_starting_with("M"))
['Mr', 'Mrs', 'Ms']
In[6]: print(Person.allowed_titles_starting_with("M"))
['Mr', 'Mrs', 'Ms']
In[7]: print(jane.allowed_titles_ending_with("s"))
['Mrs', 'Ms']
In[8]: print(Person.allowed_titles_ending_with("s"))
['Mrs', 'Ms']
```

EXAMPLE – CLASSES, OBJECTS

```
1   class Rectangle:
2       """ A class to manufacture rectangle objects """
3
4       def __init__(self, posn, w, h):
5           """ Initialize rectangle at posn, with width w, height h """
6           self.corner = posn
7           self.width = w
8           self.height = h
9
10      def __str__(self):
11          return  "({0}, {1}, {2})"
12                      .format(self.corner, self.width, self.height)
13
```

- Assume a **rectangle** that is oriented either vertically or horizontally, never at an angle;
- Specify the **upper-left corner** of the rectangle, and its **size**

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

EXAMPLE – CLASSES, OBJECTS

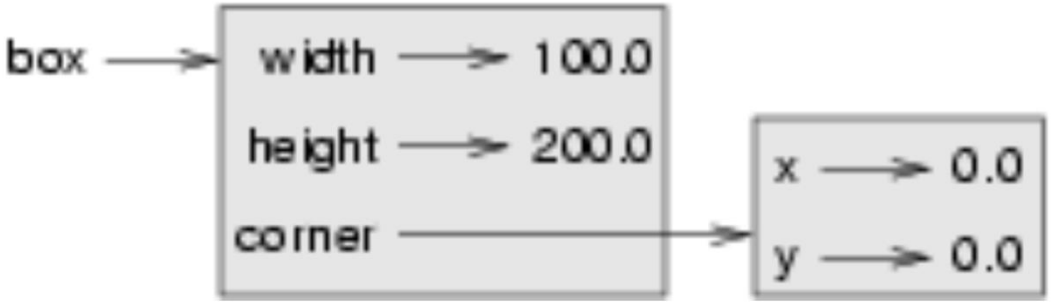
```
1   class Rectangle:
2       """ A class to manufacture rectangle objects """
3
4       def __init__(self, posn, w, h):
5           """ Initialize rectangle at posn, with width w, height h """
6           self.corner = posn
7           self.width = w
8           self.height = h
9
10      def __str__(self):
11          return  "({0}, {1}, {2})"
12                      .format(self.corner, self.width, self.height)
13
14  box = Rectangle(Point(0, 0), 100, 200)
15  bomb = Rectangle(Point(100, 80), 5, 10)    # In my video game
16  print("box: ", box)
17  print("bomb: ", bomb)
```

```
box: ((0, 0), 100, 200)
bomb: ((100, 80), 5, 10)
```

- To specify the upper-left corner embed a **Point object** within the new **Rectangle object**
- Create two new Rectangle objects, and then print them

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

DOT OPERATOR COMPOSITION



- The dot operator composes.
- The expression **box.corner.x** means:

"Go to the object that box refers to and select its attribute named corner, then go to that object and select its attribute named x"

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

OBJECTS ARE MUTABLE

```
1   class Rectangle:
2       # ...
3
4       def grow(self, delta_width, delta_height):
5           """ Grow (or shrink) this object by the deltas """
6           self.width += delta_width
7           self.height += delta_height
8
9       def move(self, dx, dy):
10          """ Move this object by the deltas """
11          self.corner.x += dx
12          self.corner.y += dy
```

```
>>> r = Rectangle(Point(10,5), 100, 50)
>>> print(r)
((10, 5), 100, 50)
>>> r.grow(25, -10)
>>> print(r)
((10, 5), 125, 40)
>>> r.move(-10, 10)
print(r)
((0, 15), 125, 40)
```

- Change the state of an object by making an assignment to one of its attributes

```
box.width += 50
box.height += 100
```

- Provide a method to encapsulate this inside the class
- Provide another method to *move the position of the rectangle elsewhere*

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

OBJECT EQUALITY

```
>>> p1 = Point(3, 4)
>>> p2 = Point(3, 4)
>>> p1 is p2
False
```

```
>>> p3 = p1
>>> p1 is p3
True
```

- *If two objects are the same, does it mean they contain the **same data** or that they are the **same object**?*
- The **is** operator was used in previous examples on the lists when explaining aliases: it allows us to find out if two references refer to the same object

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

OBJECT EQUALITY

- **Shallow copy**: is defined as constructing a new collection object and then populating it with references to the child objects found in the original, i.e. a shallow copy is only one level deep. The copying process does not recurse and therefore will not create copies of the child objects.

- **Deep copy**: is defined as recursive copying process, i.e. first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. *Copying an object this way walks the whole object tree to create a fully independent clone of the original object and all of its children*.

source https://realpython.com/copying-python-objects/

OBJECT EQUALITY

```
1  def same_coordinates(p1, p2):
2      return (p1.x == p2.x) and (p1.y == p2.y)
```

```
>>> p1 = Point(3, 4)
>>> p2 = Point(3, 4)
>>> same_coordinates(p1, p2)
True
```

- **Shallow equality**: When **is** is **True**, this type of equality is *shallow equality* because it compares only the **references** and not the *contents* of the objects
- **Deep equality**: To compare the **contents** of the objects a function like *same_coordinates* needs to be created
- **IMPORTANT**: **If two variables refer to the same object, they have both shallow and deep equality**

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

OBJECT EQUALITY

```
1  p = Point(4, 2)
2  s = Point(4, 2)
3  print("== on Points returns", p == s)
4  # By default, == on Point objects does a shallow equality test
5
6  a = [2,3]
7  b = [2,3]
8  print("== on lists returns",  a == b)
9  # But by default, == does a deep equality test on lists
```

```
== on Points returns False
== on lists returns True
```

- Think about **shallow** & **deep** copy when designing classes!
- Even though the two lists (or tuples, etc.) are *distinct objects with different memory addresses*, for **lists** the **==** operator tests for ***deep equality***, while in the case of **objects** (points) it makes a ***shallow test***

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

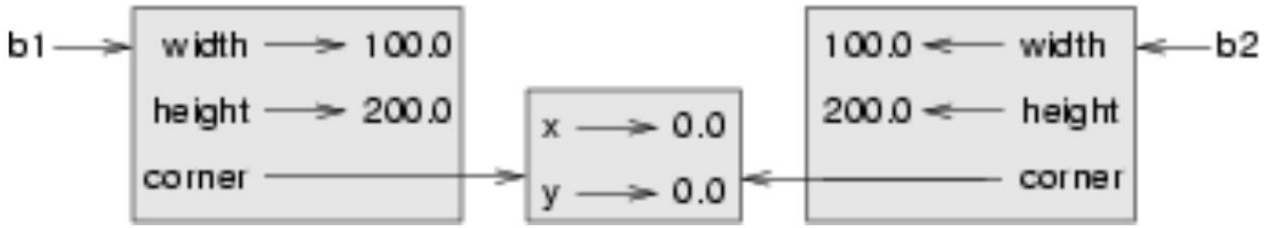
OBJECT COPY

```
>>> import copy
>>> p1 = Point(3, 4)
>>> p2 = copy.copy(p1)
>>> p1 is p2
False
>>> same_coordinates(p1, p2)
True
```

- Aliasing makes code difficult to read – *changes made in one place might have unexpected effects in another place*
- Copying object is an **alternative to aliasing:** the **copy module** contains a function copy that can duplicate any object
- **EXAMPLE**: *To copy objects import the copy module and use the copy function to make a new Point: p1 and p2 are not the same point, but they contain the same data (shallow copy)*

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

OBJECT COPY



- Assume **Rectangle**, which contains a reference to a **Point**: copy copies the reference to the **Point** object, so both the old **Rectangle** and the new one refer to the same **Point**

- **Invoking** *grow* on one of the Rectangle objects would not affect the other,
- **Invoking** *move* on either would affect both since **shallow copy has created an alias to the Point that represents the corner**
- Copy module contains a function named **deepcopy** that copies not only the object but also any embedded objects

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

OBJECT COPY

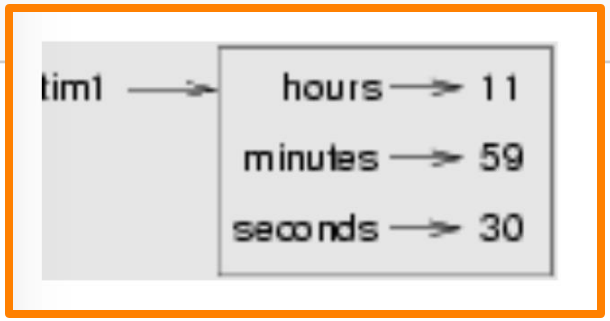
- **Deep copy**: To copy the contents of an object as well as any embedded objects, and any objects embedded in them, etc. (*implemented as deepcopy function in copy module*)
- **Deep equality**: Equality of values, or two references that point to objects that have the same value
- **Shallow copy**: To copy the contents of an object, including any references to embedded objects (one level copy).
  (*implemented by the copy function in the copy module*)
- **Shallow equality**: Equality of references, or two references that point to the same object

source http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

EXAMPLE MYTIME – OBJECT DEFINITION

```
1  class MyTime:
2
3      def __init__(self, hrs=0, mins=0, secs=0):
4          """ Create a MyTime object initialized to hrs, mins, secs """
5          self.hours = hrs
6          self.minutes = mins
7          self.seconds = secs
```

```
1  tim1 = MyTime(11, 59, 30)
```



- **EXAMPLE**: *User-defined type called **MyTime** that records the time of day*
- Initializer using an **__init__** method to ensure that every instance is created with appropriate attributes

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – PURE FUNCTIONS

```
1  def add_time(t1, t2):
2      h = t1.hours + t2.hours
3      m = t1.minutes + t2.minutes
4      s = t1.seconds + t2.seconds
5      sum_t = MyTime(h, m, s)
6      return sum_t
```

```
>>> current_time = MyTime(9, 14, 30)
>>> bread_time = MyTime(3, 35, 0)
>>> done_time = add_time(current_time, bread_time)
>>> print(done_time)
12:49:30
```

EXAMPLE: *Create two **MyTime** objects*

- **current_time**, which contains the current time
- **bread_time**, which contains the amount of time it takes for a breadmaker to make bread
- use **add_time** to figure out when the bread will be done

PROBLEM: ??

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – PURE FUNCTIONS

```
1  def add_time(t1, t2):
2      h = t1.hours + t2.hours
3      m = t1.minutes + t2.minutes
4      s = t1.seconds + t2.seconds
5      sum_t = MyTime(h, m, s)
6      return sum_t
```

```
>>> current_time = MyTime(9, 14, 30)
>>> bread_time = MyTime(3, 35, 0)
>>> done_time = add_time(current_time, bread_time)
>>> print(done_time)
12:49:30
```

EXAMPLE: *Create two **MyTime** objects:*

- **current_time**, which contains the current time
- **bread_time**, which contains the amount of time it takes for a breadmaker to make bread
- use **add_time** to figure out when the bread will be done

PROBLEM: *We do not deal with cases where the number of seconds or minutes adds up to more than sixty!*

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – PURE FUNCTIONS

```
1  def add_time(t1, t2):
2      h = t1.hours + t2.hours
3      m = t1.minutes + t2.minutes
4      s = t1.seconds + t2.seconds
5      sum_t = MyTime(h, m, s)
6      return sum_t
```

```
>>> current_time = MyTime(9, 14, 30)
>>> bread_time = MyTime(3, 35, 0)
>>> done_time = add_time(current_time, bread_time)
>>> print(done_time)
12:49:30
```

- There can be two versions of a function **add_time**, pure function or the modifier, which both calculate the sum of two MyTime objects
- Function that creates a new MyTime object and returns a reference to the new object is a **pure function** because it does not modify any of the objects passed to it as parameters and it has no side effects

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – PURE FUNCTIONS

```
1   def add_time(t1, t2):
2
3       h = t1.hours + t2.hours
4       m = t1.minutes + t2.minutes
5       s = t1.seconds + t2.seconds
6
7       if s >= 60:
8           s -= 60
9           m += 1
10
11      if m >= 60:
12          m -= 60
13          h += 1
14
15      sum_t = MyTime(h, m, s)
16      return sum_t
```

PROBLEM: *Do we now deal with cases where the number of seconds or minutes adds up to more than sixty?*

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – MODIFIERS

```
1   def increment(t, secs):
2       t.seconds += secs
3
4       if t.seconds >= 60:
5           t.seconds -= 60
6           t.minutes += 1
7
8       if t.minutes >= 60:
9           t.minutes -= 60
10          t.hours += 1
```

```
1   def increment(t, seconds):
2       t.seconds += seconds
3
4       while t.seconds >= 60:
5           t.seconds -= 60
6           t.minutes += 1
7
8       while t.minutes >= 60:
9           t.minutes -= 60
10          t.hours += 1
```

- It can be useful for a function to **modify** one or more of the objects it gets as parameters
- Usually, the **caller keeps a reference** to the objects it passes, so any changes the function are visible to the caller (*modifier*)
- Function **increment**, which adds a given number of seconds to a MyTime object, is a natural example of a **modifier**

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – MODIFIERS

```python
class MyTime:
    # Previous method definitions here...

    def increment(self, seconds):
        self.seconds += seconds

        while self.seconds >= 60:
            self.seconds -= 60
            self.minutes += 1

        while self.minutes >= 60:
            self.minutes -= 60
            self.hours += 1
```

```python
current_time.increment(500)
```

- **SOLUTION**: Include functions that work with MyTime objects into the MyTime class, i.e. conversion of the function **increment** to a method
- This conversion means moving the definition into the class and changing the name of the first parameter to **self**

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – INSIGHT

```python
class MyTime:
    # ...

    def to_seconds(self):
        """ Return the number of seconds represented
            by this instance
        """
        return self.hours * 3600 + self.minutes * 60 + self.seconds
```

```python
hrs = tsecs // 3600
leftoversecs = tsecs % 3600
mins = leftoversecs // 60
secs = leftoversecs % 60
```

- **INSIGHT**: **MyTime** object is actually a **three-digit number in base 60**!
- Another approach —convert the MyTime object into a single number instead, i.e. the method **to_seconds** can be added to the **MyTime** class to convert any instance into corresponding number of seconds

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – INSIGHT

```
1   class MyTime:
2       # ...
3
4       def __init__(self, hrs=0, mins=0, secs=0):
5           """ Create a new MyTime object initialized to hrs, mins, secs.
6               The values of mins and secs may be outside the range 0-59,
7               but the resulting MyTime object will be normalized.
8           """
9
10          # Calculate total seconds to represent
11          totalsecs = hrs*3600 + mins*60 + secs
12          self.hours = totalsecs // 3600        # Split in h, m, s
13          leftoversecs = totalsecs % 3600
14          self.minutes = leftoversecs // 60
15          self.seconds = leftoversecs % 60
```

- In OOP we wrap together the **data** and the **operations**
- Solution is to rewrite the class initializer so that it can cope with initial values of seconds or minutes that are outside the range of the normalized values

(*normalized time: 3 hours 12 minutes and 20 seconds; the same time but not normalized 2 hours 70 minutes and 140 seconds*)

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – SOLUTION

```
>>> t1 = MyTime(10, 55, 12)
>>> t2 = MyTime(10, 48, 22)
>>> after(t1, t2)              # Is t1 after t2?
True
```

- The function **after** can be defined to compare two times and specify whether the first time is strictly after the second
- This solution is a bit more complicated because it operates on **two MyTime objects** not just one

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – SOLUTION

```
1  class MyTime:
2      # Previous method definitions here...
3
4      def after(self, time2):
5          """ Return True if I am strictly greater than time2 """
6          if self.hours > time2.hours:
7              return True
8          if self.hours < time2.hours:
9              return False
10
11         if self.minutes > time2.minutes:
12             return True
13         if self.minutes < time2.minutes:
14             return False
15         if self.seconds > time2.seconds:
16             return True
17
18         return False
```

```
1  if current_time.after(done_time):
2      print("The bread will be done before it starts!")
```

- **Lines 11-18** will only be reached if the two hour fields are the same.
- The test at **line 16** is only executed if both times have the same hours and the same minutes.

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – SOLUTION

```
1  class MyTime:
2      # Previous method definitions here...
3
4      def after(self, time2):
5          """ Return True if I am strictly greater than time2 """
6          return self.to_seconds() > time2.to_seconds()
```

- The whole example can be made easier using the previously discovered insight of converting the time into single integer!
- This is a great way to code this:
  If we want to tell if the first time is after the second time, turn them both into integers and compare the integers.

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – OVERLOADING

```
1  class MyTime:
2      # Previously defined methods here...
3
4      def __add__(self, other):
5          return MyTime(0, 0, self.to_seconds() + other.to_seconds())
```

- **Operator overloading**: *opens the possibility to have different meanings for the same operator when applied to different types*

- **EXAMPLE**: *the + in Python means quite different things for integers (**addition**) and for strings (**concatenation**)!*
- To override the addition operator **+** provide a method named **__add__**

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

EXAMPLE MYTIME – OVERLOADING

```
1  class MyTime:
2      # Previously defined methods here...
3
4      def __add__(self, other):
5          return MyTime(0, 0, self.to_seconds() + other.to_seconds())
```

```
>>> t1 = MyTime(1, 15, 42)
>>> t2 = MyTime(3, 50, 30)
>>> t3 = t1 + t2
>>> print(t3)
05:06:12
```

- **First parameter** is the object on which the method is invoked
- **Second parameter** is named other to distinguish it from self
- To add two MyTime objects create and return a new MyTime object that contains their sum

- The expression **t1 + t2** is equivalent to **t1.__add__(t2)**

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

OPERATOR OVERLOADING

```
1  class Point:
2      # Previously defined methods here...
3
4      def __add__(self, other):
5          return Point(self.x + other.x,  self.y + other.y)
```

- **EXAMPLE**: *back to the Point class – adding two points adds their respective (x, y) coordinates*
- There are several ways to override the behavior of the multiplication operator by defining a method named **__mul__** or **__rmul__** or both

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

OPERATOR OVERLOADING

```
1  def __mul__(self, other):
2      return self.x * other.x + self.y * other.y
```

```
1  def __rmul__(self, other):
2      return Point(other * self.x,  other * self.y)
```

- If the **left** operand of * is a **Point**, Python invokes **__mul__**, which assumes that the other operand is also a Point
  (this computes the dot product of the two Points)
- If the **left** operand of * is a **primitive type** and the right operand is a **Point**, Python invokes **__rmul__**, which performs *scalar multiplication*
- The result is always a new **Point** whose coordinates are a multiple of the original coordinates

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

OPERATOR OVERLOADING

```
>>> p1 = Point(3, 4)
>>> p2 = Point(5, 7)
>>> print(p1 * p2)
43
>>> print(2 * p2)
(10, 14)
```

PROBLEM: How is **p2 * 2** evaluated?

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

OPERATOR OVERLOADING

```
>>> p1 = Point(3, 4)
>>> p2 = Point(5, 7)
>>> print(p1 * p2)
43
>>> print(2 * p2)
(10, 14)
```

```
>>> print(p2 * 2)
AttributeError: 'int' object has no attribute 'x'
```

- **PROBLEM**: How is **p2 * 2** evaluated?
- Since the first parameter is a Point, Python invokes **__mul__** with 2 as the second argument
- Inside **__mul__**, the program tries to access the **x** coordinate of other, which fails because an integer has no attributes

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

POLYMORPHISM

```
1  def front_and_back(front):
2      import copy
3      back = copy.copy(front)
4      back.reverse()
5      print(str(front) + str(back))
```

```
>>> my_list = [1, 2, 3, 4]
>>> front_and_back(my_list)
[1, 2, 3, 4][4, 3, 2, 1]
```

- **EXAMPLE**: ***front_and_back*** *– consider a function which prints a list twice: forward and backward*
- The reverse method is a **modifier** therefore a copy needs to be made before applying it (this way we prevent to modify the list the function gets as a parameter!)
- Function that can take arguments with different types and handles them accordingly is called **polymorphic**

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

POLYMORPHISM

```
1  def multadd (x, y, z):
2      return x * y + z
```

```
>>> multadd (3, 2, 1)
7
```

```
>>> p1 = Point(3, 4)
>>> p2 = Point(5, 7)
>>> print(multadd (2, p1, p2))
(11, 15)
>>> print(multadd (p1, p2, 1))
44
```

- **Polymorphism ==** ability to process objects differently based on data type
- There are certain operations that can be applied to many types, such as the arithmetic operations …
- **EXAMPLE**: *The **multadd** operation takes three parameters: multiplies the first two and then adds the third*

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

POLYMORPHISM

```
1  def multadd (x, y, z):
2      return x * y + z
```

```
>>> multadd (3, 2, 1)
7
```

```
>>> p1 = Point(3, 4)
>>> p2 = Point(5, 7)
>>> print(multadd (2, p1, p2))
(11, 15)
>>> print(multadd (p1, p2, 1))
44
```

- **EXAMPLE**: *The **multadd** operation takes three parameters: multiplies the first two and then adds the third*

- **The first case**: the Point is multiplied by a scalar and then added to another Point
- **The second case:** the dot product yields a numeric value, so the third parameter also has to be a numeric value

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

POLYMORPHISM

```
1  def reverse(self):
2      (self.x , self.y) = (self.y, self.x)
```

```
>>> p = Point(3, 4)
>>> front_and_back(p)
(3, 4)(4, 3)
```

- **Python's fundamental rule of polymorphism** is called the **duck typing rule**: *If all of the operations inside the function can be applied to the type, the function can be applied to the type*.
- Operations in the **front_and_back** : *copy, reverse, print*
- **EXAMPLE**: What about our Point class?
  The copy method works on any object; already written a **__str__** method for Point objects for the str() conversion, only the reverse method for the Point class is needed!

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

REFERENCES

This lecture re-uses selected parts of the OPEN BOOK PROJECT
Learning with Python 3 (RLE)
http://openbookproject.net/thinkcs/python/english3e/index.html
available under **GNU Free Documentation License Version 1.3**)

- Version date: October 2012
- by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers (based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)
- Source repository is at https://code.launchpad.net/~thinkcspy-rle-team/thinkcspy/thinkcspy3-rle
- For offline use, download a zip file of the html or a pdf version from http://www.ict.ru.ac.za/Resources/cspw/thinkcspy3/

This lecture re-uses selected parts of the PYTHON TEXTBOOK
Object-Oriented Programming in Python
http://python-textbok.readthedocs.io/en/1.0/Classes.html#
(released under CC BY-SA 4.0 licence Revision 8e685e710775)