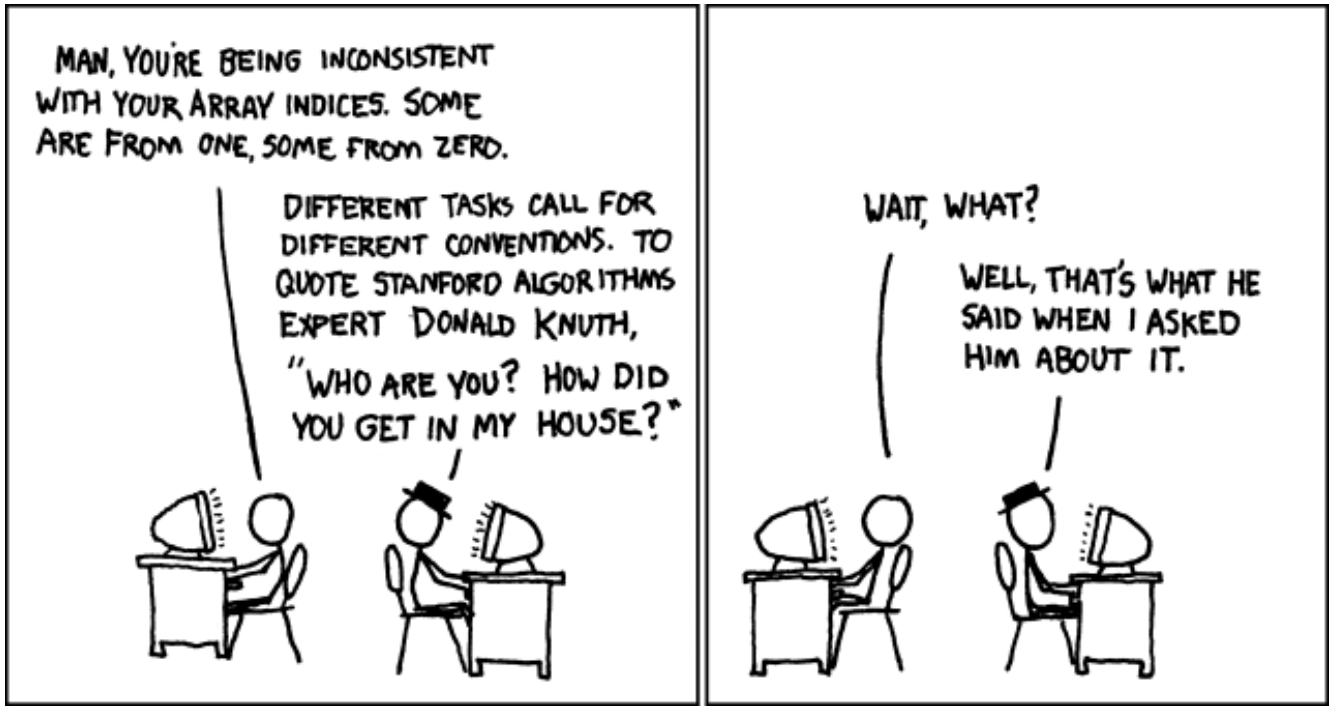
CSc 110, Spring 2017

Lecture 33: Methods

Adapted from slides by Marty Stepp and Stuart Reges



Questions

```
class Point:
  def __init__(self):
      self.x = 0
      self.y = 0

def draw(self, panel, color):
      panel.canvas.create_oval(self.x, self.y,
        self.x + 3, self.y + 3, outline=color)
      panel.canvas.create_text(self.x, self.y,
        text  =  "(" + str(self.x) + ", " + str(self.y) + ")")
```

What is the name of the class?
What is this class definition used for?
`x` is an _____ of the class `Point`.
`draw` is a _____ of the class `Point`.
What is the purpose of `__init__` ?

Initializing objects

- Currently it takes 3 lines to create a `Point` and initialize it:

```
p = Point()
p.x = 3
p.y = 8
```

- Here's an alternative approach:

```
p = Point(3, 8)    # not implemented yet
```

We will modify the `Point` class constructor to take parameters.

`Point` class, version 3

```
class Point:
  def __init__(self, x, y):
      self.x = x
      self.y = y

  def draw(self, panel):
      panel.canvas.create_rectangle(
        self.x, self.y, self.x + 3, self.y + 3)
      panel.canvas.create_text(self.x, self.y,
        text = "(" + str(self.x) + ", " + str(self.y) + ")")
```

- Each `Point` object is now initialized to the `x` and `y` passed in.

Class method question

Write a method `distance_from_origin` that returns the distance between a `Point` and the origin, (0, 0). Usage is shown below.

```
>>> p = Point(3,10)
>>> p.distance_from_origin()
10.440306508910551
>>>
```

Use the Pythagorean theorem.

Modify the `Point` class.

Class method answer

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

     def distance_from_origin(self):
        return sqrt(self.x ** 2 + self.y **2)
      ...
```

Understanding the implicit variable `self`

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return sqrt(self.x **2 + self.y  **2)
    ...(other methods here)

p1 = Point(7,2)
p2 = Point(4,3)
p1.distance_from_origin()
p2.distance_from_origin()
```

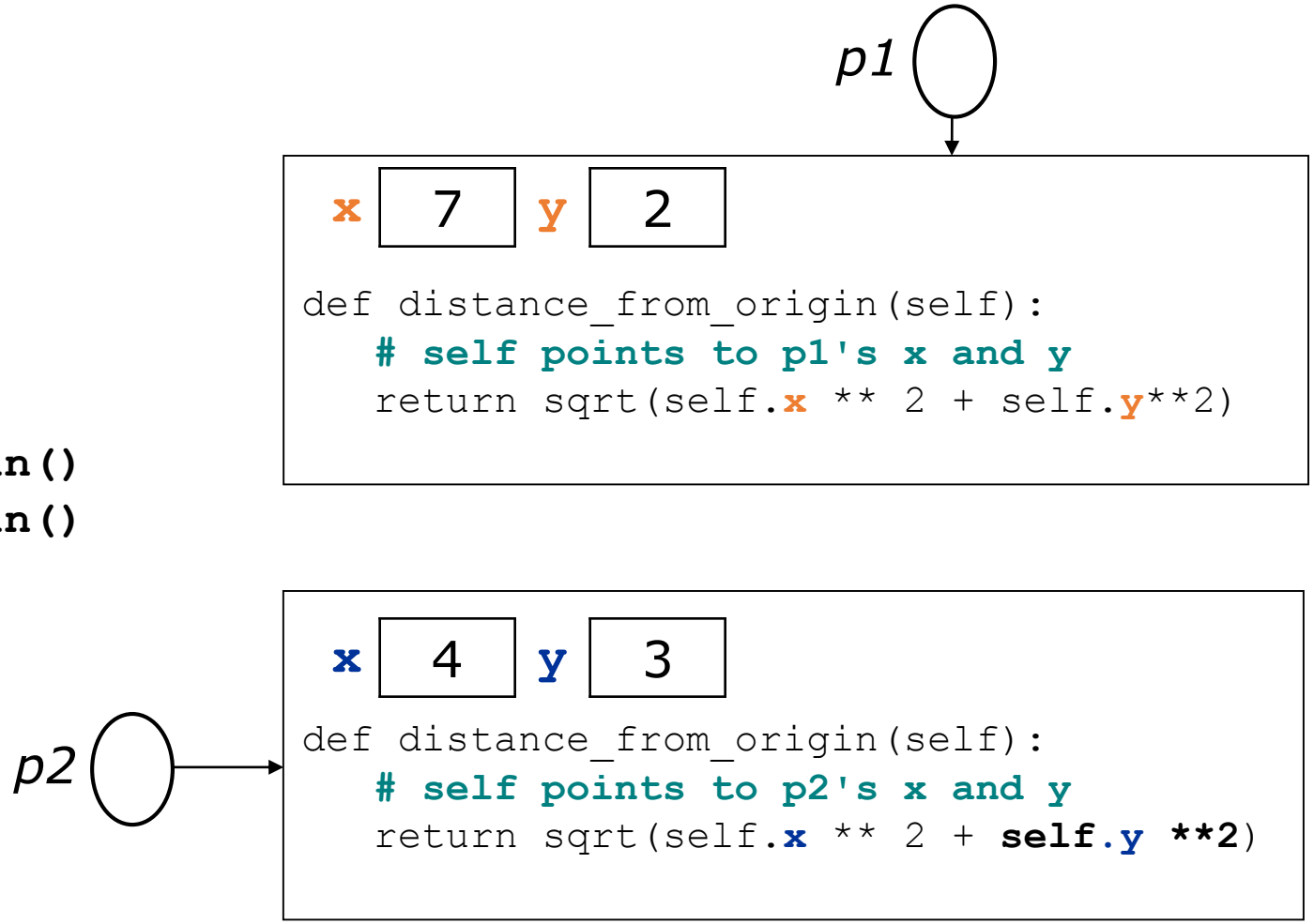
Understanding the implicit variable `self`

- For a given `Point` object, the `distance_from_origin` method operates on that object's state.

```
p1 = Point(7,2)

p2 = Point(4,3)

p1.distance_from_origin()
p2.distance_from_origin()
```



Printing objects

- By default, Python doesn't know how to print objects:

```
p = Point()
p.x = 10
p.y = 7
print("p is ", p)  # p is <p.Point object at 0x000001BA6AE0BF28>


# better, but cumbersome;              p is (10, 7)
print("p is (" + str(p.x) + ", " + str(p.y) + ")")


# desired behavior
print("p is ", p))  # p is (10, 7)
```

Class method question

- Write a method `show()` that returns a string consisting of the `x` and `y` attributes of a point surrounded by parenthesis.

The following code provides an example of using the `show()` method:

```
>>> p = Point(30, 45)
>>> p.show()
'(30,45)'
>>>
>>> print(p.show())
(30,45)
>>>
```

Class method question

- Write a method `translate` that changes a `Point`'s location by a given *dx*, *dy* amount.

The following code provides an example of using the `translate` method:

```
>>> p = Point(8, 20)
>>> p.show()
'(8,20)'
>>> p.translate(2, 10)
>>> p.show()
'(10,30)'
>>>
```

The `__str__` method

tells Python how to convert an object into a `string`

```
p1 = Point(7, 2)
print("p1: " + str(p1))
```

By default you get this output:

```
<point.Point object at 0x000001BA6AE0BF28>
```

Every class has a `__str__`, even if it isn't in your code.

You can write your own code for the `__str__` method

`__str__` syntax

```
def __str__(self):
    code that returns a String representing this object
```

- Method name, return, and parameters must match exactly.
- Example:

```
# Returns a String representing this Point.
def __str__(self):
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```

Class method answers

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return sqrt(self.x ** 2 + self.y **2)

    def show(self):
        return "(" + str(self.x) + "," + str(self.y + ")"

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy

    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"
…
```

Kinds of methods

- **accessor**: A method that examines an object's state.
  - Example: `show`, `distance_from_origin`
  - often returns something
  - also called a getter method

- **mutator**: A method that modifies an object's state.
  - Example: `translate`
  - also called a setter method

```
class Review:
    def __init__(self, title, author, rating):
        self.__title = title
        self.__author = author
        self.__rating = int(rating)

    def get_title(self):
        return self.__title

    def get_author(self):
        return self.__author

    def get_rating(self):
        return self.__rating

    def __str__(self):
        return ("Title: " + self.__title + " by " + self.__author +
                ", rating = " + str(self.__rating))
```

Accessing objects in a set

How do you access an object that is in a set in a dictionary?

Regardless of what the set contains, how do you access the elements of a set?

Suppose you have a set called `set_of_reviews`:

```
for r in set_of_reviews:
    <process r>
```

Accessing attributes of a Review object

- If you loop over a set and each set element `r` is a `Review` object, how do you access the attributes of `r`?

- Looking at the Review class the methods are:

```
get_title()
get_author()
get_rating()
```

- If you have a `Review` object `r`, then

`r.get_title()` is the title
`r.get_author()` is the author
`r.get_rating()` is the rating