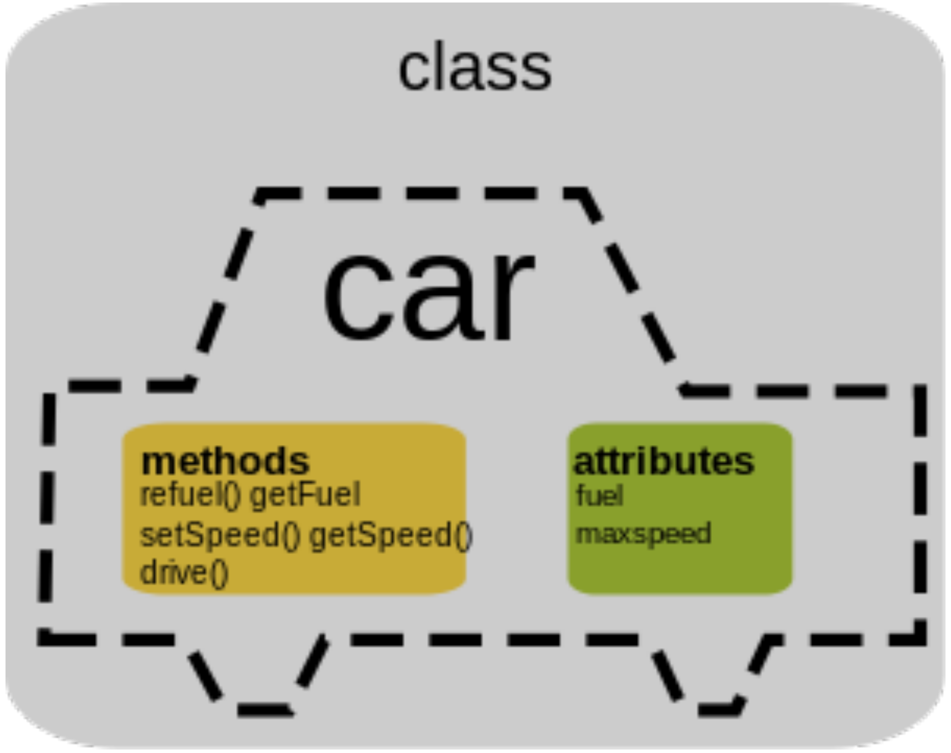
15-112
Fundamentals of Programming

Week 9 - Lecture 1b:
Intro to Object Oriented Programming (OOP)



March 15, 2016

What is object oriented programming (OOP)?

1. The ability to create your own data types.

```
s = “hello”
print(s.capitalize())

s = set()
s.add(5)
```

These are built-in data types.

2. Designing your programs around the data types you create.

Important terminology

data	data type (type)
object	class
instance	

```
s = set()
```

Create an object/instance of type/class set.

s is then a reference to that object/instance.

Motivating example

Suppose you want to keep track of the books in your library.

For each book, you want to store:
title, author, year published

How can we do it?

Motivating example

Option 1:

```
book1Title = “The Catcher in the Rye”
book1Author = “J. D. Sallinger”
book1Year = 1951

book2Title = “The Brothers Karamazov”
book2Author = “F. Dostoevsky”
book2Year = 1880;
```

Would be better to use one variable for each book.

One variable to hold logically connected data together.
(like lists)

Motivating example

Option 2:

```
book1 = [“The Catcher in the Rye”, “J.D. Sallinger”, 1951]

book2 = list()
book2.append(“The Brothers Karamazov”)
book2.append(“F. Dostoevsky”)
book2.append(1880)
```

Can forget which index corresponds to what.

Hurts readability.

Motivating example

Option 3:

```
book1 = {“title”: “The Catcher in the Rye”,
         “author”: “J.D. Sallinger”,
         “year”: 1951}

book2 = dict()
book2[“title”] = “The Brothers Karamazov”,
book2[“author”] = “F. Dostoevsky”
book2[“year”] = 1880
```

Doesn’t really tell us what type of object book1 and book2 are.

They are just dictionaries.

Motivating example

Option 3:

```
book1 = {“title”: “The Catcher in the Rye”,
         “author”: “J.D. Sallinger”,
         “year”: 1951}

book2 = {“title”: “The Brothers Karamazov”,
         “author”: “F. Dostoevsky”,
         “year”: 1880}

article1 = {“title”: “On the Electrodynamics of Moving Bodies”,
            “author”: “A. Einstein”,
            “year”: 1905}
```

Better to define a new data type.

Defining a data type (class) called Book

```
class Book(object):

    def __init__(self):
        self.title = None
        self.author = None
        self.year = None
```

name of the new data type

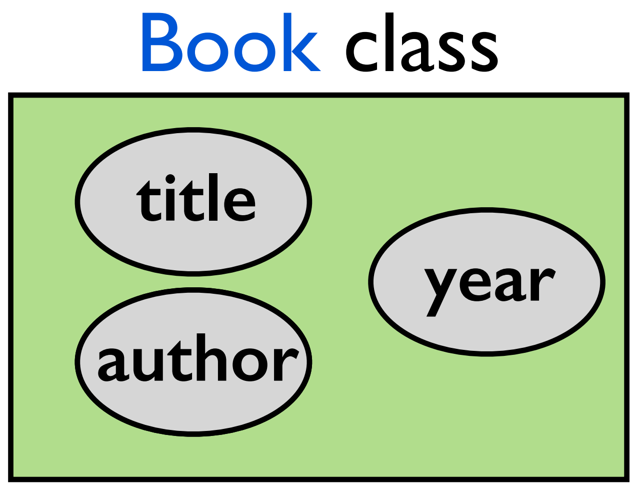
fields or properties or data members or attributes

This **defines** a new data type named Book.

__init__ is called a constructor.

Defining a data type (class) called Book

```
class Book(object):

    def __init__(self):
        self.title = None
        self.author = None
        self.year = None
```



Defining a data type (class) called Book

```
class Book(object):
    def __init__(self):
        self.title = None
        self.author = None
        self.year = None

b = Book()
b.title = “Hamlet”
b.author = “Shakespeare”
b.year = 1602
```

call __init__ with self = b

Creates an object of type Book

b refers to that object.

Compare to:

```
b = dict()
b[“title”] = “Hamlet”
b[“author”] = “Shakespeare”
b[“year”] = 1602
```

Creating 2 books

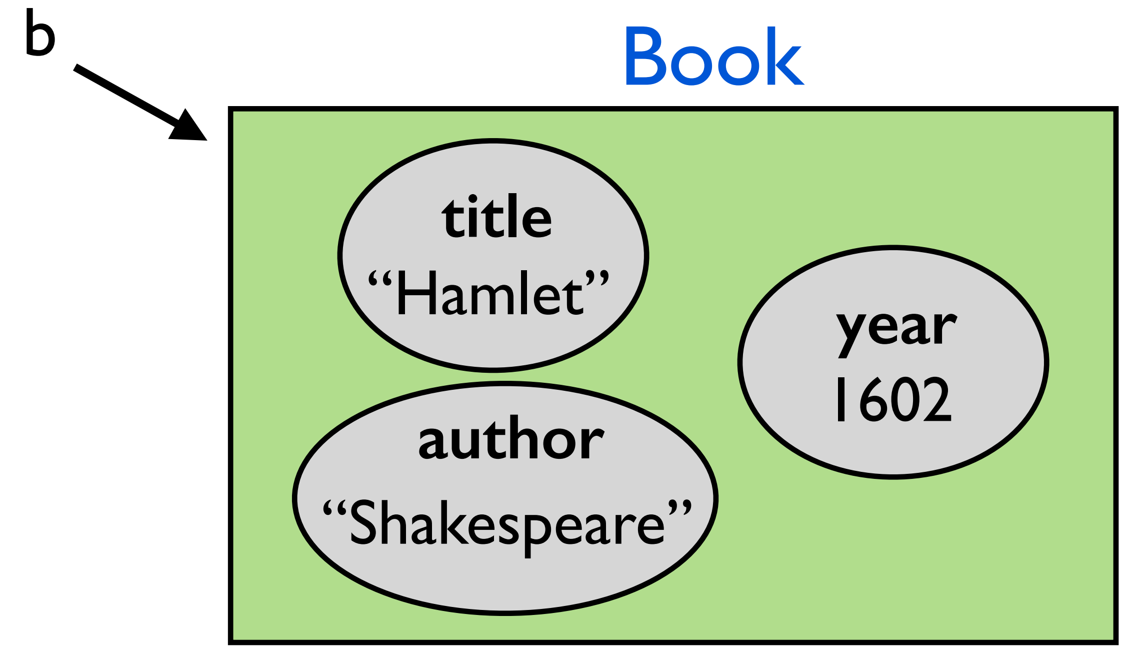
```
class Book(object):
    def __init__(self):
        self.title = None
        self.author = None
        self.year = None

b = Book()
b.title = “Hamlet”
b.author = “Shakespeare”
b.year = 1602

b2 = Book()
b2.title = “It”
b2.author = “S. King”
b2.year = 1987
```

b refers to an object of type Book.

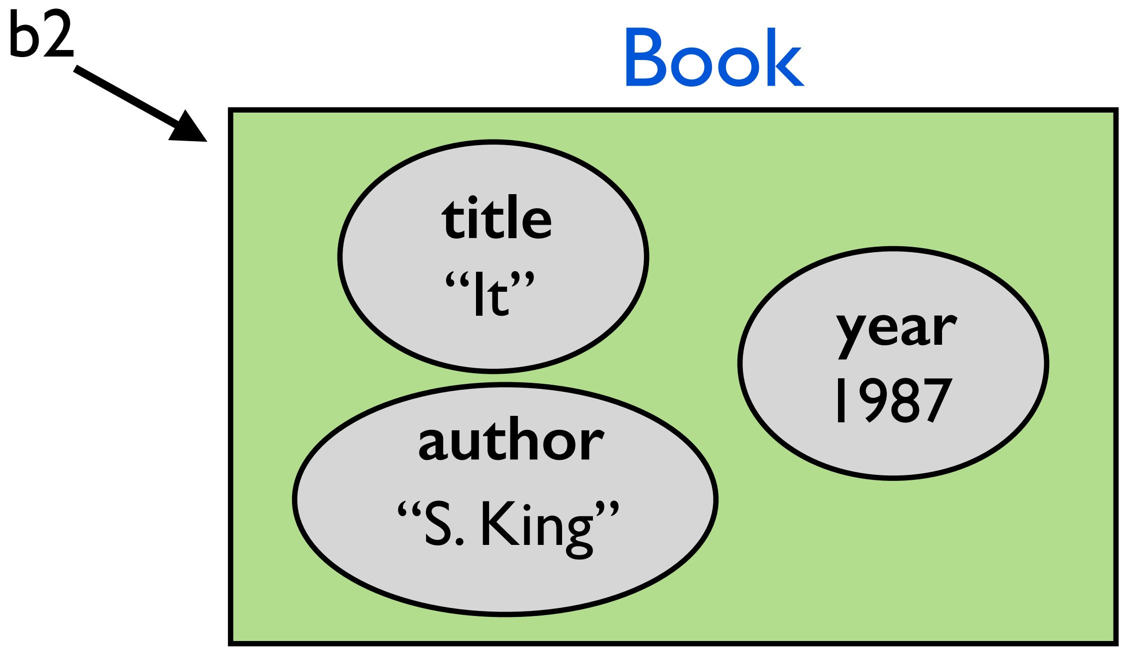
b2 refers to another object of type Book.

Creating 2 books

```
b = Book()
b.title = "Hamlet"
b.author = "Shakespeare"
b.year = 1602
```



```
b2 = Book()
b2.title = "It"
b2.author = "S. King"
b2.year = 1987
```



Initializing fields at object creation

```
class Book(object):
    def __init__(self, t, a, y):
        self.title = t
        self.author = a
        self.year = y
```

```
b.title = “Hamlet”
b.author = “Shakespeare”
b.year = 1602
```

```
b = Book(“Hamlet”, “Shakespeare”, 1602)
```

Initializing fields at object creation

```
class Book(object):
    def __init__(self, title, author, year):
        self.title = title
        self.author = author
        self.year = year
```

```
b.title = “Hamlet”
b.author = “Shakespeare”
b.year = 1602
```

```
b = Book(“Hamlet”, “Shakespeare”, 1602)
```

Initializing fields at object creation

```
class Book(object):
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.year = None
```

```
b.title = “Hamlet”
b.author = “Shakespeare”
```

```
b = Book(“Hamlet”, “Shakespeare”)
```

Initializing fields at object creation

```
class Book(object):
    def __init__(foo, title, author):
        foo.title = title
        foo.author = author
        foo.year = None
```

```
b.title = “Hamlet”
b.author = “Shakespeare”
```

```
b = Book(“Hamlet”, “Shakespeare”)
```

An object has 2 parts

1. instance variables: a collection of related data

2. methods: functions that act on that data

```
s = "hello"
s.capitalize()
```

Recall this is like having
a function called capitalize:

```
capitalize(s)
```

How can you define methods?

Example: Rectangle

```
class Rectangle(object):
    def __init__(self, width, height):
        self.width = width
        self.height = height


def getArea(rec):
    return rec.width*rec.height


r = Rectangle(3, 5)
print (“The area is”,  getArea(r))
```

Defining a **function**
that acts on a rectangle object

Example: Rectangle

```
class Rectangle(object):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def getArea(self):
        return self.width*self.height
```

Defining a **method**

that acts on a rectangle object

```
r = Rectangle(3, 5)
print ("The area is",  r.getArea())
```

Example: Rectangle

```
class Rectangle(object):
    def __init__(self, width, height):
        self.width = width
        self.height = height

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

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

    def doubleDimensions(self):
        self.width *= 2
        self.height *= 2

    def rotate90Degrees(self):
        (self.width, self.height) = (self.height, self.width)
```

Example: Dot

```
class Dot(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.r = random.randint(20,50)
        self.fill = random.choice(["pink","orange","yellow","green",
                                   "cyan","purple"])
        self.clickCount = 0

    def containsPoint(self, x, y):
        d = ((self.x - x)**2 + (self.y - y)**2)**0.5
        return (d <= self.r)


    def draw(self, canvas):
        canvas.create_oval(self.x-self.r, self.y-self.r,
                           self.x+self.r, self.y+self.r,
                           fill=self.fill)
        canvas.create_text(self.x, self.y, text=str(self.clickCount))
```