CS 1124 Media Computation

Lecture 11.1 Nov 3, 2008

Steve Harrison

TODAY

- Objects
- Other Languages
  - Javascript

TODAY

- Objects
- Other Languages
  - Javascript

Object-oriented programming

- Define and describe the *objects* of the world
  - **Noun-oriented**
  - **Focus on the domain of the program**
  - **The object-oriented analyst asks herself: "The program I'm trying to write relates to the real world in some way. What are the things in the real world that this program relates to?"**

Example to motivate objects: SlideShow

- Let’s build a program to show a slide show.
  - **It shows a picture.**
  - **Then plays a corresponding sound.**
    - We’ll use the introduced-but-never-used **blockingPlay**() to make the execution wait until the sound is done.

Slideshow

```
def playslideshow():
    pic = makePicture(getMediaPath("barbara.jpg"))
    snd = makeSound(getMediaPath("bassoon-c4.wav"))
    show(pic)
    blockingPlay(snd)
    pic = makePicture(getMediaPath("beach.jpg"))
    snd = makeSound(getMediaPath("bassoon-e4.wav"))
    show(pic)
    blockingPlay(snd)
    pic = makePicture(getMediaPath("santa.jpg"))
    snd = makeSound(getMediaPath("bassoon-g4.wav"))
    show(pic)
    blockingPlay(snd)
    pic = makePicture(getMediaPath("jungle2.jpg"))
    snd = makeSound(getMediaPath("bassoon-c4.wav"))
    show(pic)
    blockingPlay(snd)
```

What’s wrong with this?

- From Procedural Abstraction:
  - **We have duplicated code.**
  - **We should get rid of it.**
- From Object-Oriented Programming:
  - **We have an object: A slide (with sound).**

Defining an object

- Objects *know* things.
  - **Data that is internal to the object.**
  - **We often call those instance variables.**
- Objects can *do* things.
  - **Behavior that is internal to the object.**
  - **We call functions that are specific to an object methods.**
    - But you knew that one already.
- We access both of these using *dot notation*
  - **object.variable**
  - **object.method()**

The Slide Object

- What does a slide know?
  - **It has a picture.**
  - **It has a sound**
- What can a slide do?
  - **Show itself.**
    - Show its picture.
    - (Blocking) Play its sound.

Classes

- Objects are *instances* of *classes* in many object-oriented languages.
  - **Including Smalltalk, Java, JavaScript, and Python.**
- A class defines the data and behavior of an object.
  - **A class defines what all instances of that class know and can do.**

We need to define a slide class

- Easy enough:

```
class slide:
 def someMethod(self):
   print "The slide class has one method."
```

- Methods are defined *like* functions, but *indented* within the class definition.
  - **We'll explain self in just a few minutes**
- What comes next?
  - **Some method for creating new slides.**
  - **Some method for playing slides.**

Creating new instances

- We are going to create new instances by calling the class name as if it were a function.
  - **That will automatically create a new instance of the class.**

Creating a slide

- Let's create a slide and give it a picture and sound *instance variables.*

```
>>> slide1=slide()
>>> slide1.someMethod()
```

The slide class has one method.

```
>>> slide1.picture =
    makePicture(getMediaPath("barbara.jpg"))
>>> slide1.sound = makeSound(getMediaPath("bassoon-
    c4.wav"))
```

Defining a show() method

- To show a slide, we want to **show()** the picture and **blockingPlay**() the sound.
- We define the function as part of the class *block*.
  - **So this is a def that gets indented.**

Defining the method show()

- Why self?
  - **When we say object.method(),**
  - **Python finds the method in the object's class, then calls it with the object as an input.**
  - **Python style is to call that self.**
    - It's the object it*self*.

```
class slide:
  def show(self):
    show(self.picture)
    blockingPlay(self.sound)
```

Now we can show our slide

```
>>> slide1.show()
```

- We execute the method using the same dot notation we've seen previously.
- Does just what you'd expect it to do.
  - **Shows the picture.**
  - **Plays the sound.**

Making it simpler

- Can we get rid of those picture and sound assignments?
- What if we could call **slide** as if it were a real function, with inputs?
  - **Then we could pass in the picture and sound filenames as inputs.**
- We can do this, by defining what Java calls a *constructor.*
  - **A method that builds your object for you.**

Making instances more flexibly

- To create new instances with inputs, we must define a function named __init__
  - **That's underscore-underscore-i-n-i-t-underscore-underscore.**
    - DOUBLE UNDERSCORE BEFORE AND AFTER
  - **It's the predefined name for a method that initializes new objects.**
- Our __init__ function will take three inputs:
  - **self, because all methods take that.**
  - **And a picture and sound filename.**
    - We'll create the pictures and sounds in the method.

Our whole slide class

```
class slide:
  def __init__(self, pictureFile,soundFile):
    self.picture = makePicture(pictureFile)
    self.sound = makeSound(soundFile)

  def show(self):
    show(self.picture)
    blockingPlay(self.sound)
```

The playslideshow()

```
def playslideshow():
    slide1 = slide(getMediaPath("barbara.jpg"), getMediaPath("bassoon-c4.wav"))
    slide2 = slide(getMediaPath("beach.jpg"),getMediaPath("bassoon-e4.wav"))
    slide3 = slide(getMediaPath("santa.jpg"),getMediaPath("bassoon-g4.wav"))
    slide4 = slide(getMediaPath("jungle2.jpg"),getMediaPath("bassoon-c4.wav"))
    slide1.show()
    slide2.show()
    slide3.show()
    slide4.show()
```

The value of objects

- Is this program easier to write?
  - **It certainly has less replication of code.**
  - **It does combine the data and behavior of slides in one place.**
    - If we want to change how slides work, we change them in the definition of slides.
    - We call that *encapsulation:* Combining data and behavior related to that data.
  - **Being able to use other objects with our objects is powerful.**
    - Being able to make lists of objects, to be able to use objects (like picture and sound) in our objects.
    - We call that *aggregation*: Combining objects, so that there are objects in other objects.

If slides are objects, what about a slide player?

```
class slidePlayer:
  ?????
```

We’ve been doing this already, of course.

- You’ve been using objects already, everywhere.
- Pictures, sounds, samples, colors—these are all objects.
- We’ve been doing aggregation.
    - **We’ve worked with or talked about lists of pictures, sounds, pixels, and samples**
- The functions that we’ve been providing merely cover up the underlying objects.

Using picture as an object

```
>>> pic=makePicture(getMediaPath("barbara.jpg"))
>>> pic.show()
```

Slides and pictures both show()

- Did you notice that we can say **slide1.show()** and **pic.show()**?
- Show() generally means, in both contexts, “show the object.”
- But what’s really *happening* is *different* in each context!
  - **Slides show pictures and play sounds.**
  - **Pictures just show themselves.**

Another powerful aspect of objects: Polymorphism

- When the same method *name* can be applied to more than one object, we call that method *polymorphic*
  - **From the Greek "many shaped"**
- A polymorphic method is very powerful for the programmer.
  - **You don't need to know exactly what method is being executed.**
  - **You don't even need to know exactly what object it is that you're telling to show()**
  - **You just know your goal: Show this object!**

Uncovering the objects

- This is how the show() function is defined in JES:

```
def show(picture):
    if not picture.__class__ == Picture:
        print "show(picture): Input is not a picture"
        raise ValueError
    picture.show()
```

- You can ignore the **raise** and **if**
  - **The key point is that the function is simply executing the method.**

Pictures and Colors have polymorphic methods, too

```
>>> pic=makePicture(getMediaPath("barbara.jpg"))
>>> pic.show()
>>> pixel = getPixel(pic,100,200)
>>> print pixel.getRed()
73
>>> color = pixel.getColor()
>>> print color.getRed()
73
```

We can get/set components at either level

- getRed, getBlue, getGreen, setRed, setBlue, setGreen
  - **Are all defined for both colors and pixels**
- Why didn’t we define the functions to work with either?
  - **It’s somewhat confusing to have a globally-available function take two kinds of things as input: Colors or pixels.**
  - **But it’s completely reasonable to have a method of the same name in more than one object.**

More methods than functions

- In general, there are many more *methods* defined in JES than there are *functions*.
- Most specifically, there are a whole bunch of methods for drawing onto a picture that aren't defined as functions.
  - **It's easier to deal with the complexity at the level of methods than functions.**
  - **The names for the functions get more and more complicated, where polymorphism lets them be simple and contextualized.**

Overview of graphics methods

- pic.addRect(color,x,y,width,height)
- pic.addRectFilled(color,x,y,width,height)
- pic.addOval(color,x,y,width,height)
- pic.addOvalFilled(color,x,y,width,height)

Arcs

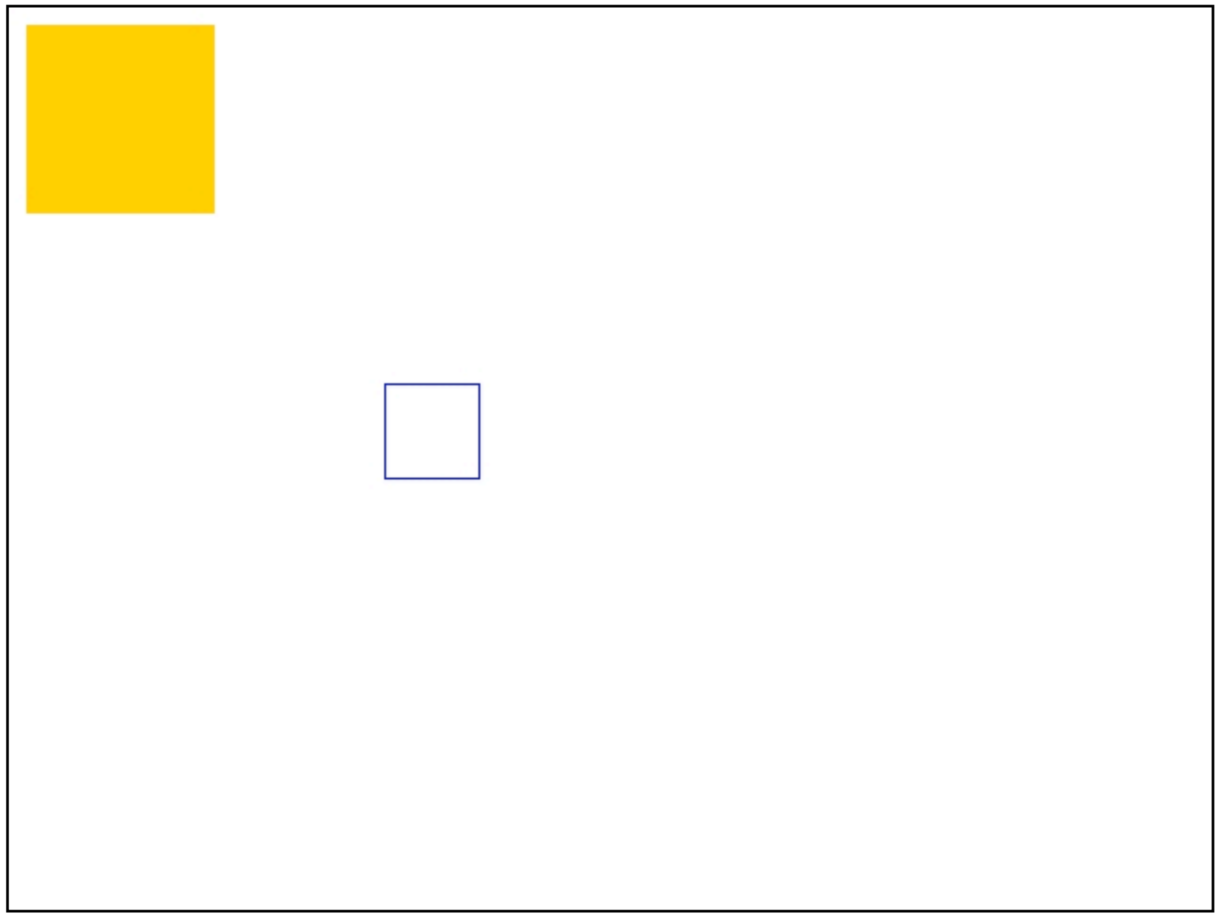
- pic.addArc(color,x,y,width,height,startangle,arcangle)
- pic.addArcFilled(color,x,y,width,height,startangle,arcangle)
  - **Make an arc for arcangle degrees, where startangle is the starting point. 0 = 3 o'clock.**
    - Positive arc is counter-clockwise, negative is clockwise
  - **Center of the circle is middle of the rectangle (x,y) with given height and width**

Text

- Text can have style, but only limited.
  - **Java limits it for cross-platform compatibility.**
- pic.addText(color,x,y,string)
- pic.addTextWithStyle(color,x,y,string,style)
  - **Style is made by makeStyle(font,emph,size)**
  - **Font is sansSerif, serf, or mono**
  - **Emph is italic, bold, or plain.**
    - You can get italic, bold by *italic+bold*
  - **Size is a point size**

Rectangles: Coloring lines and fills

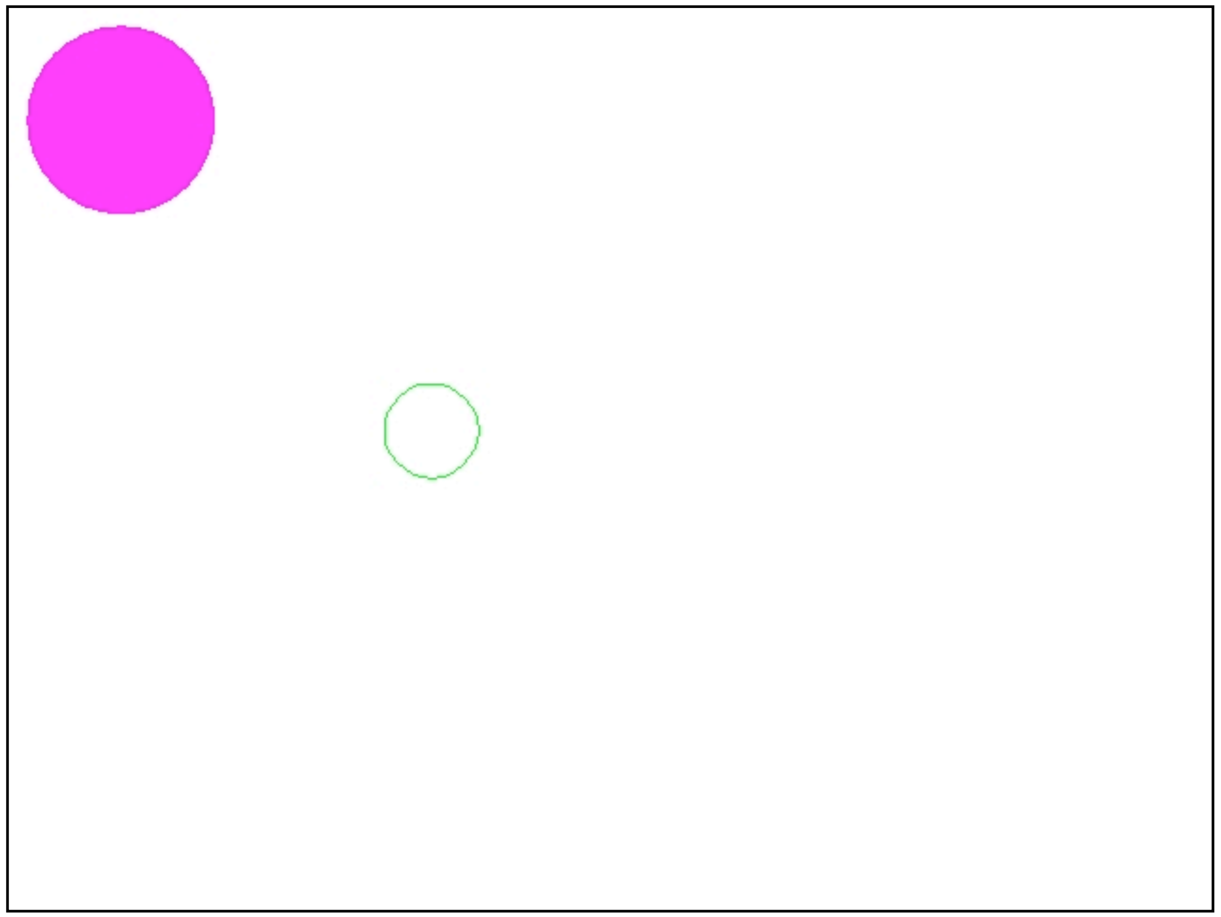
```
>>> pic=makePicture
    (getMediaPath("640x480.jpg"))
>>> pic.addRectFilled (orange,
    10,10,100,100)
>>> pic.addRect (blue,
    200,200,50,50)
>>> pic.show()
>>> pic.writeTo("newrects.jpg")
```



writeTo() is polymorphic for both sounds and pictures.

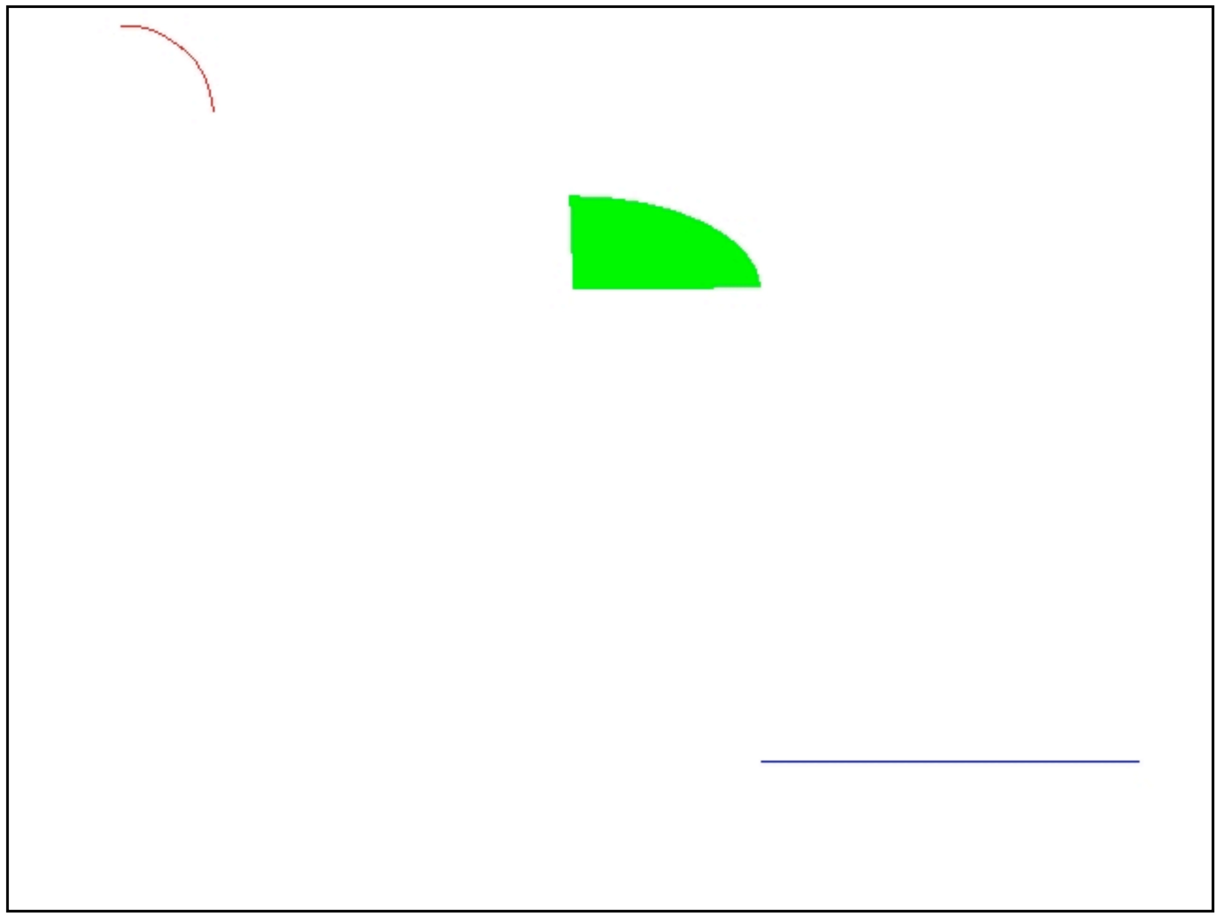
Ovals

```
>>> pic=makePicture
    (getMediaPath("640x480.jpg"))
>>> pic.addOval (green,
    200,200,50,50)
>>> pic.addOvalFilled (magenta,
    10,10,100,100)
>>> pic.show()
>>> pic.writeTo("ovals.jpg")
```



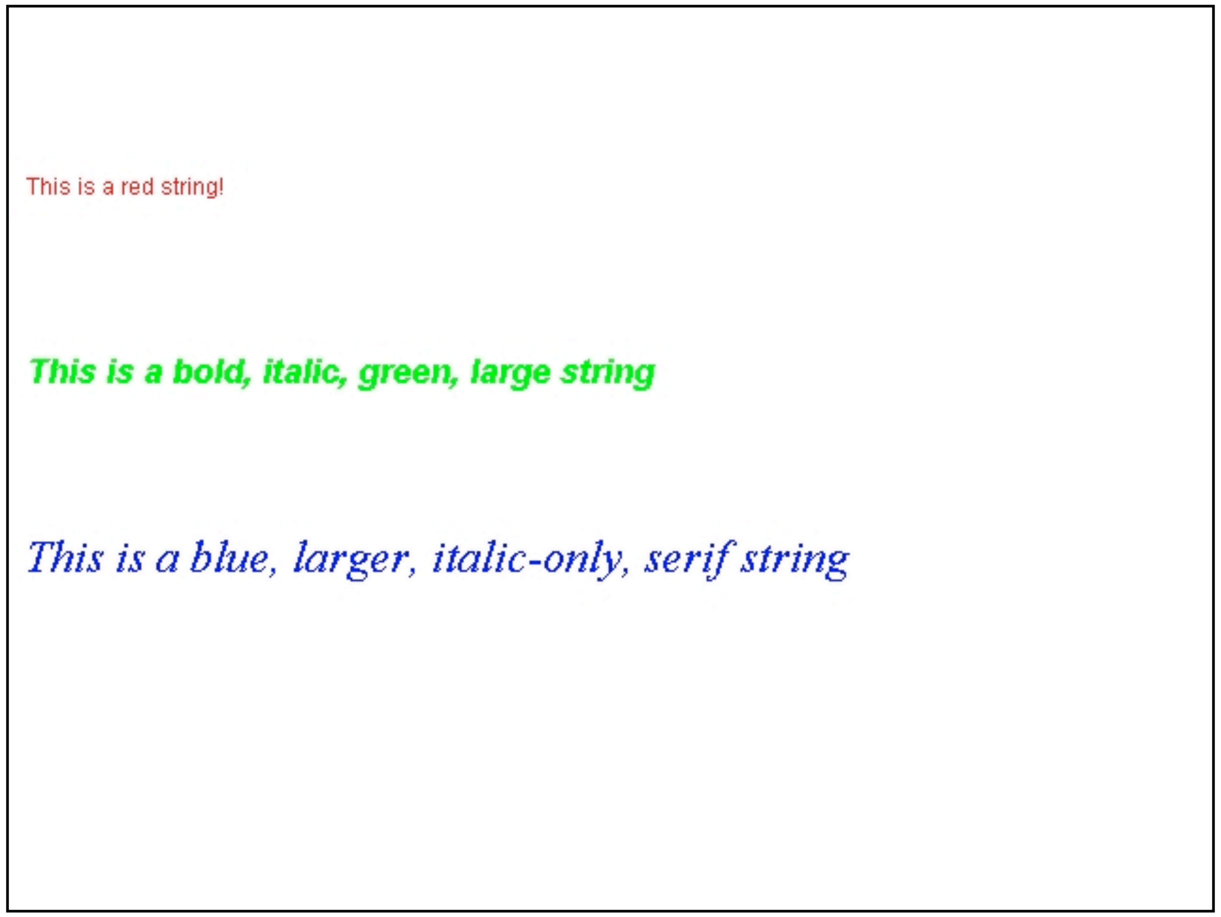
Arcs and colored lines

```
>>> pic=makePicture
    (getMediaPath("640x480.jpg"))
>>> pic.addArc(red,10,10,100,100,5,45)
>>> pic.show()
>>> pic.addArcFilled (green,
    200,100,200,100,1,90)
>>> pic.repaint()
>>> pic.addLine(blue,400,400,600,400)
>>> pic.repaint()
>>> pic.writeTo("arcs-lines.jpg")
```



Text examples

```
>>> pic=makePicture
    (getMediaPath("640x480.jpg"))
>>> pic.addText(red,10,100,"This is a
    red  string!")
>>> pic.addTextWithStyle (green,
    10,200,"This is a bold, italic,
    green, large string",
    makeStyle(sansSerif,bold+italic,
    18))
>>> pic.addTextWithStyle (blue,
    10,300,"This is a blue, larger,
    italic-only, serif string",
    makeStyle(serif,italic,24))
>>> pic.writeTo("text.jpg")
```


This is a red string!
This is a bold, italic, green, large string
This is a blue, larger, italic-only, serif string

Yes, you may use any of these in your homework, if you want.

Sunset using methods

- Any of our older functions will work just fine with methods.

```
def makeSunset(picture):
  for p in getPixels(picture):
    p.setBlue(p.getBlue()*0.7)
    p.setGreen(p.getGreen()*0.7)
```

Backwards using methods

```
def backwards(filename):
  source = makeSound(filename)
  target = makeSound(filename)

  sourceIndex = source.getLength()
  for targetIndex in range(1,target.getLength()+1):
    # The method is getSampleValue, not getSampleValueAt
    sourceValue =source.getSampleValue(sourceIndex)
    # The method is setSampleValue, not setSampleValueAt
    target.setSampleValue(targetIndex,sourceValue)
    sourceIndex = sourceIndex - 1

  return target
```

To get the sample object, snd.getSampleObjectAt(index)

Why objects?

- An important role for objects is to reduce the number of names that you have to remember.
  - **writeSoundTo() and writePictureTo() vs. sound.writeTo() and picture.writeTo()**
- They also make it easier to change data and behavior *together*.
  - **Think about changing the name of an instance variable. What functions do you need to change? Odds are good that they're the ones right next to where you're changing the variable.**
- Most significant power is in *aggregation:* Combining objects

Python objects vs. other objects

- One of the key ideas for objects was “not messing with the innards.”
- Not true in Python.
  - **We can always get at instance variables of objects.**
- It is true in other object-oriented languages.
  - **In Java or Smalltalk, instance variables are only accessible through methods (getPixel) or through special declarations (“This variable is public!”)**

Inheritance

- We can declare one class to be *inherited* by another class.
- It provides instant polymorphism.
  - **The child class immediately gets all the data and behavior of the parent class.**
- The child can then add *more* than the parent class had.
  - **This is called making the child a specialization of the parent.**
  - **A 3-D rectangle might know/do all that a rectangle does, plus some more:**

```
class rectangle3D(rectangle):
```

Inheritance is a tradeoff

- Inheritance is talked about a lot in the object-oriented world.
  - **It does reduce even further duplication of code.**
  - **If you have two classes that will have many the same methods, then set up inheritance.**
- But in actual practice, inheritance doesn’t get used all that much, and can be confusing.

When should you use objects?

- Define your own objects when you have:
  - **Data in groups, like both pictures and sounds.**
  - **Behavior that you want to define over that group.**
- Use existing objects:
  - **Always—they're very powerful!**
  - **Unless you're not comfortable with dot notation and the idea of methods.**
    - Then functions work just fine.

Using map with slides

- Slides are now just objects, like any other kind of object in Python.
- They can be in lists, for example.
- Which means that we can use map.
- We need a function:

```
def showSlide(aslide):
  aslide.show()
```

PlaySlideShow with Map

```
def playslideshow():
    slide1 = slide(getMediaPath("barbara.jpg"), getMediaPath("bassoon-
     c4.wav"))
    slide2 = slide(getMediaPath("beach.jpg"),getMediaPath("bassoon-e4.wav"))
    slide3 = slide(getMediaPath("santa.jpg"),getMediaPath("bassoon-g4.wav"))
    slide4 = slide(getMediaPath("jungle2.jpg"),getMediaPath("bassoon-c4.wav"))
    map(showSlide,[slide1,slide2,slide3,slide4])
```

TODAY

- Objects
- Other Languages
  - Javascript

What do other languages look like?

- We call the language “look” its *syntax*
- Python is a fairly traditional language in terms of syntax.
  - **Languages like Scheme and Squeak are significantly different.**
- Some points of difference:
  - **Whether or not variables have to be declared before first use.**
  - **Details of how individual lines are written.**
  - **Details of how blocks are defined.**

TODAY

- Objects
- Other Languages
  - Javascript

JavaScript

- JavaScript is meant to be a *scripting* language, like Python.
  - **Scripting languages are meant for non-professional programmers to solve simple tasks.**
  - **It's designed to look like Java to ease the transition in either way.**
- JavaScript can be used by the web server (used on the computer accessed via the Internet), or it can be used within an HTML page.
  - **If it's within the HTML page, it's actually executed by the user's browser.**
  - **We call that client side JavaScript.**

JavaScript syntax: Variables

- Variables must be declared before use.
  - **You can't just say:**

    ```
    a = 12
    ```
  - **You can either say:**

    ```
    var a = 12;
    ```
  - **Or:**

    ```
    var a;
    a = 12;
    ```
- In other languages, you might also declare the variable's *type*

  ```
  int a=12;
  ```

JavaScript syntax: Blocks

- Blocks are delimited with curly braces.

```
function test()
{
   document.writeln("This is a test");
}
```

JavaScript syntax: Individual statements

- Lots of differences:
  - **function instead of def**
  - **End lines with semicolons “;”**
    - (But lines can have returns in the middle of them.)
  - **The for statement is numeric (mostly) and has different parts to it.**
  - **You use write or writeln instead of print**
- But they’re mostly detail changes.
  - **The basic operation of JavaScript is not unlike Python.**

JavaScript is all about objects

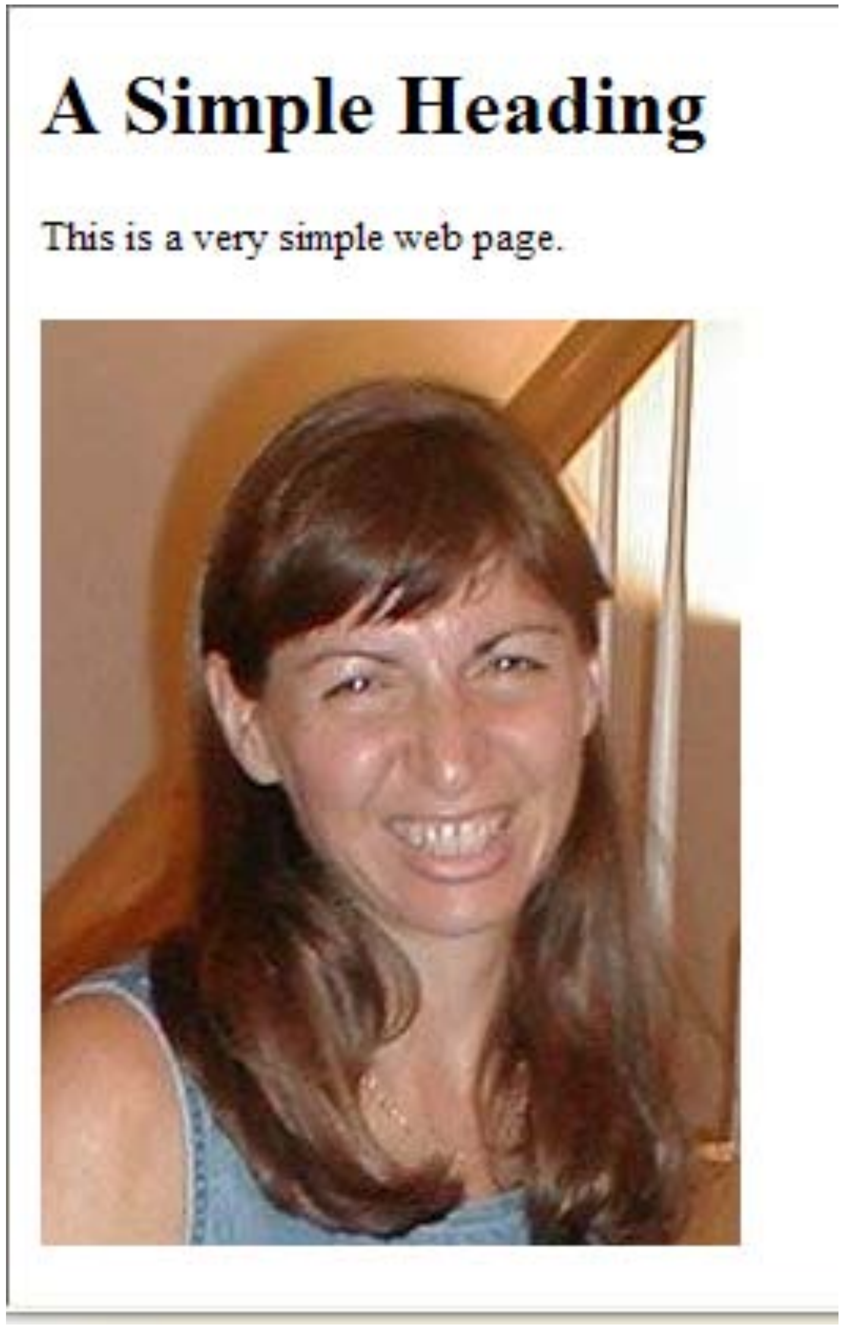
- Just about every function is actually a method.
- For example, there is no global **print**.
- There is a function **write** or **writeln**
  - **Writeln adds a new line (‘\n’) at the end.**
- But these aren’t global functions.
  - **To write into the document, you use document.write()**
  - **document.write() is a method on the HTML document itself.**

Embedding JavaScript inside HTML

- JavaScript sits inside of HTML pages.
  - **You wrap <script> </script> tags around the JavaScript.**
- You can have <script> tags in two kinds of places.
  - **Inside the <head></head> tags to define functions used elsewhere.**
  - **Inside the body, where the scripts are actually executed.**

Our Simple Web Page

```
<!DOCTYPE HTML PUBLIC "-//W3C//
    DTD HTML 4.01 Transition//EN"
    "http://wwww.w3.org/TR/html4/
    loose.dtd">
<html>
<head>
<title>The Simplest Possible Web Page</
    title>
</head>
<body>
<h1>A Simple Heading</h1>
<p>This is a very simple web page.</p>
<p><image src="mediasources/
    barbara.jpg" />
</body>
</html>
```

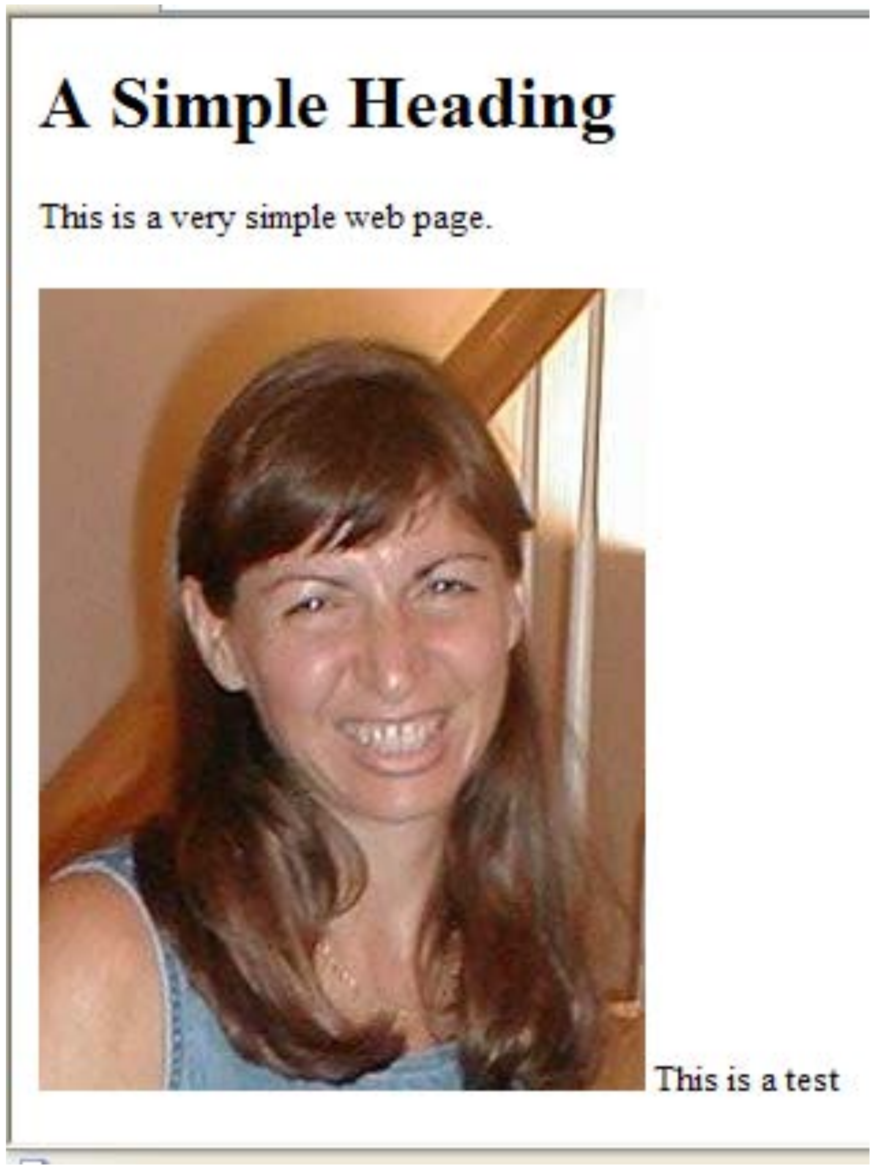


A Simple Heading

This is a very simple web page.

Adding some simple JavaScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
      HTML 4.01 Transition//EN" "http://
      wwww.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>The Simplest Possible Web Page</title>
<script>
function test()
{
   document.writeln("This is a test");
}
</script>
</head>
<body>
<h1>A Simple Heading</h1>
<p>This is a very simple web page.</p>
<p><image src="mediasources/barbara.jpg" />
<script> test() </script></p>
</body>
</html>
```



A Simple Heading

This is a very simple web page.

This is a test

Going into detail on the function

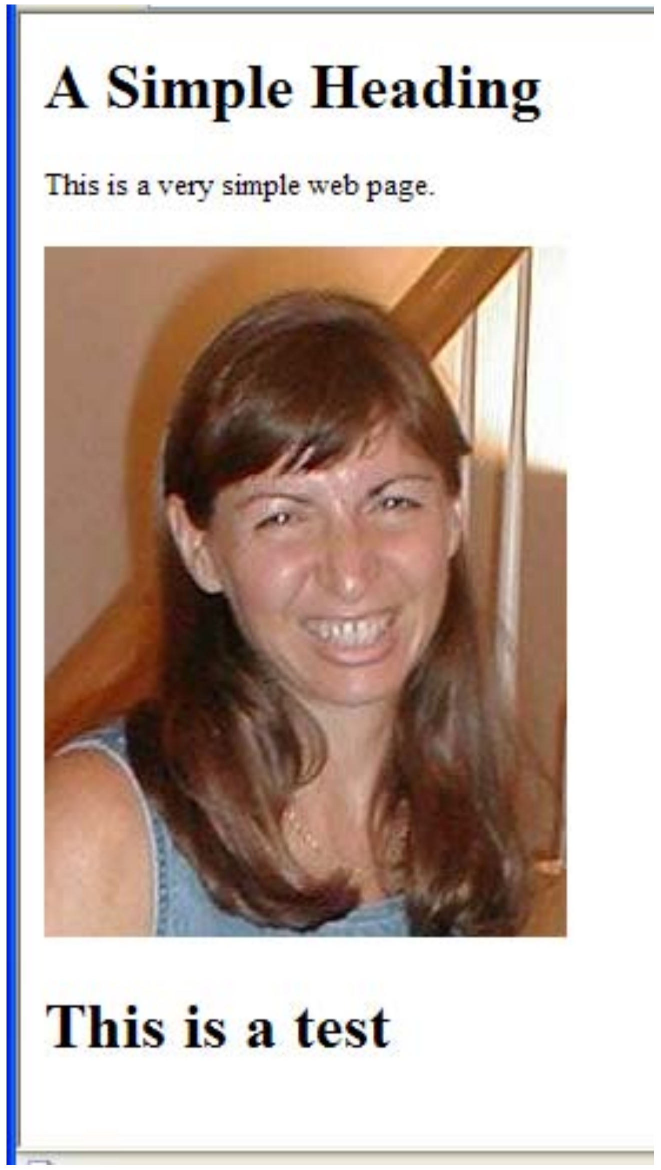
```
<script>
function test()
{
   document.writeln("This is a test");
}
</script>
</head>
<body>
<h1>A Simple Heading</h1>
<p>This is a very simple web page.</p>
<p><image src="mediasources/barbara.jpg" />
<script> test() </script></p>
```

Here's a function named "test" with no inputs, that only writes out a string.

Here we execute the function.

Can also insert HTML

```
<script>
function insertHead()
{
   document.writeln("<h1>This is a test</h1>");
}
</script>
</head>
<body>
<h1>A Simple Heading</h1>
<p>This is a very simple web page.</p>
<p><image src="mediasources/barbara.jpg" />
</p>
<script> insertHead() </script>
</body>
</html>
```



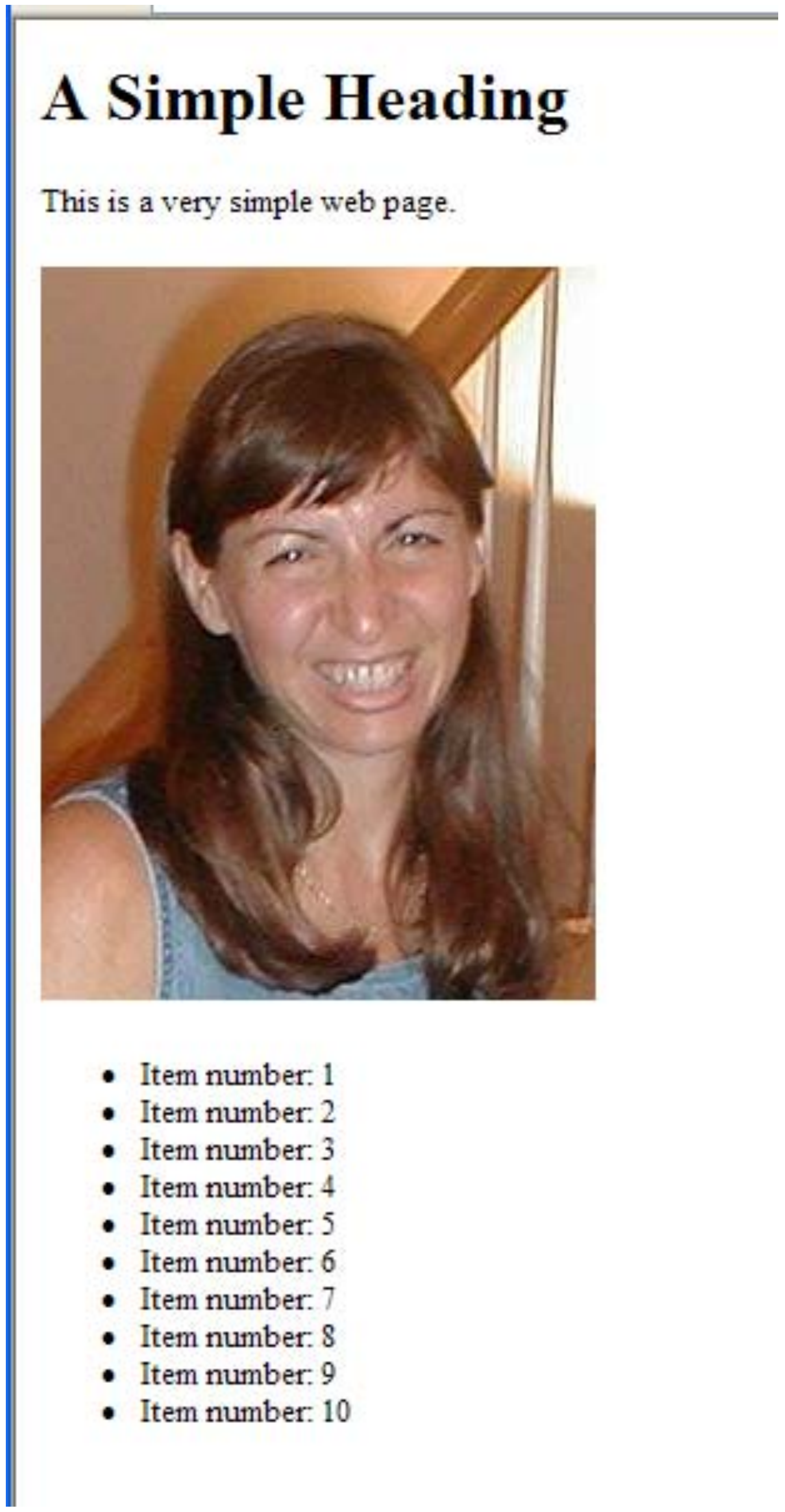
A Simple Heading

This is a very simple web page.

This is a test

Using loops

```
<html>
<head>
<title>The Simplest Possible Web Page</title>
<script>
function countToTen()
{
  document.write("<ul>");
  for (i=1; i<= 10; i++)
  {
    document.write("<li>Item number: ",i);
  }
  document.write("</ul>");
}
</script>
</head>
<body>
<h1>A Simple Heading</h1>
<p>This is a very simple web page.</p>
<p><image src="mediasources/barbara.jpg" />
</p>
<script> countToTen() </script>
</body>
</html>
```



A Simple Heading

This is a very simple web page.

- Item number: 1
- Item number: 2
- Item number: 3
- Item number: 4
- Item number: 5
- Item number: 6
- Item number: 7
- Item number: 8
- Item number: 9
- Item number: 10