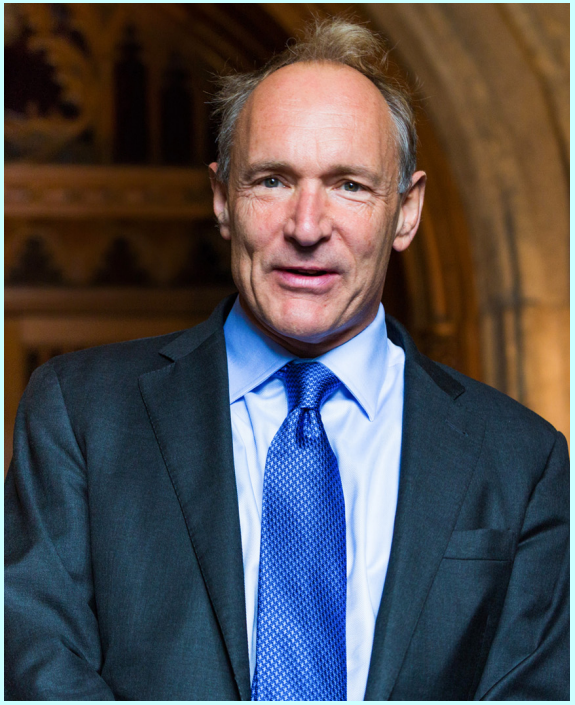
# JavaScript Programs

Jerry Cain
CS 106AJ
October 3, 2018
*slides courtesy of Eric Roberts*

*Once upon a time . . .*

# The World Wide Web

- One of the strengths of JavaScript is its integration with the World Wide Web, which was invented by Tim Berners-Lee at the CERN laboratory in Switzerland in 1989.
- In honor of his contributions to the web, Berners-Lee received the Turing Award in 2016. Named after computer scientist Alan Turing, the Turing Award is the highest honor in the computing field.



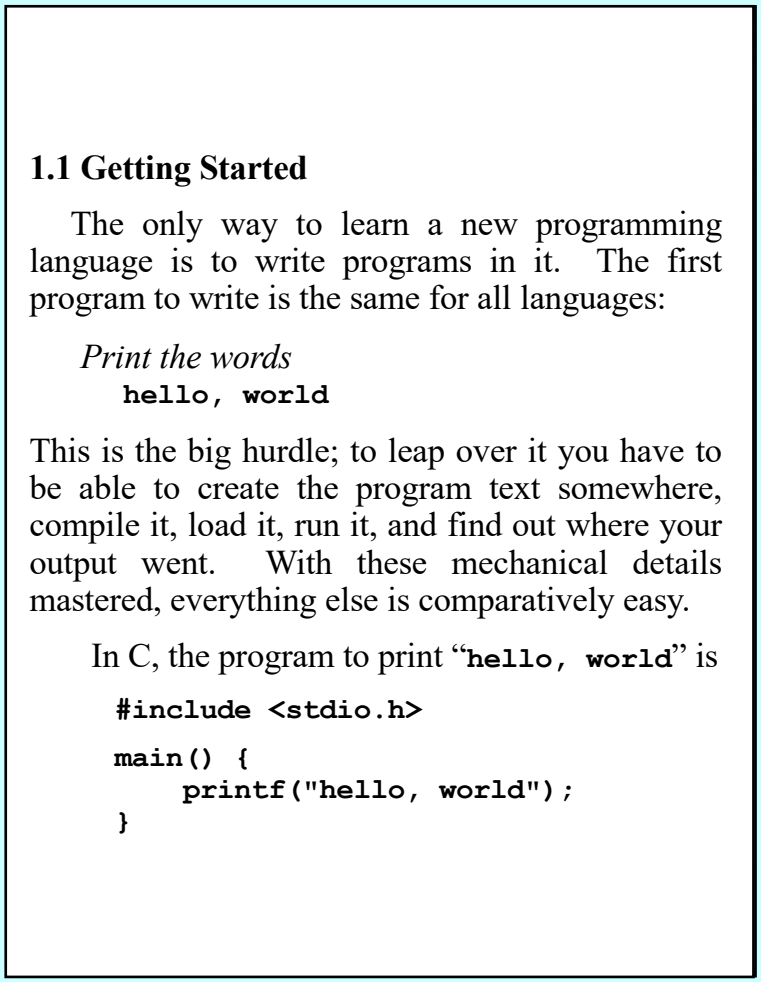
**Tim Berners-Lee (1955–)**

- The ideas behind the the World Wide Web have a long history that begins before the computing age. Contributors to the idea of a world-wide interconnected repository of data include the Belgian bibliographer Paul Otlet, the MIT-based engineer and scientist Vannevar Bush, and computer visionary Ted Nelson.

# JavaScript Programs

# The "Hello World" Program

- In Monday's class, you learned how to execute JavaScript functions in the console window. Today, your goal is to learn how to create and execute a complete JavaScript program.
- In 1978, Brian Kernighan and Turing Award winner Dennis Ritchie wrote the reference manual for the C programming language, one of the forerunners of JavaScript.
- On the first page of their book, the authors suggest that the first step in learning any language is to write a simple program that prints the words "hello, world" on the display. That advice remains sound today.



**1.1 Getting Started**

The only way to learn a new programming language is to write programs in it. The first program to write is the same for all languages:

*Print the words*

```
hello, world
```

This is the big hurdle; to leap over it you have to be able to create the program text somewhere, compile it, load it, run it, and find out where your output went. With these mechanical details mastered, everything else is comparatively easy.

In C, the program to print "`hello, world`" is

```
#include <stdio.h>

main() {
    printf("hello, world");
}
```

# HelloWorld in JavaScript

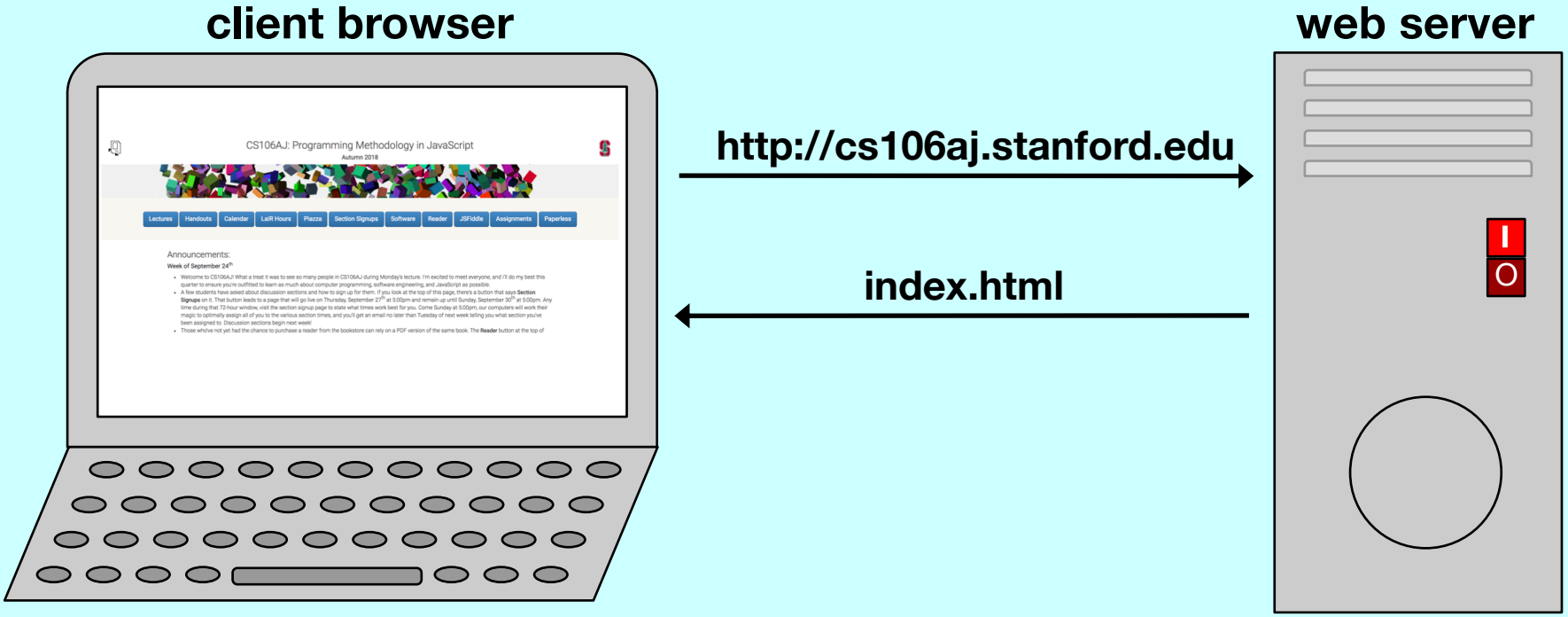
- The code for the **`HelloWorld`** program in JavaScript is similar to the C version from 1978 and looks like this:

```
function HelloWorld() {
   console.log("hello, world");
}
```

  The **`HelloWorld`** function asks JavaScript to display the string **`"hello, world"`** on the console log. So far, so good.

- From here, you need to do the following things:
  - Learn how to store this function in a file.
  - Understand how to get JavaScript to execute the function.
  - Figure out where the output goes.
- These steps are different in JavaScript than they are in most languages because JavaScript relies on a web-based model.

# The Web's Client-Server Model




1. The user starts a web browser.
2. The user requests a web page.
3. The browser sends a network request for the page.
4. The server sends back a text file containing the HTML.
5. The browser interprets the HTML and renders the page image.

# The Three Central Web Technologies

- Modern web pages depend on three technological tools: ***HTML*** (Hypertext Markup Language), ***CSS*** (Cascading Style Sheets), and JavaScript.
- These tools are used to control different aspects of the page:
  - HTML is used to specify content and structure.
  - CSS is used to control appearance and formatting.
  - JavaScript is used to animate the page.
- You will have a chance to learn even more about HTML and CSS later in the quarter. For now, all you need to know is how to write a simple **`index.html`** file that will read and execute your JavaScript code.

# The Structure of an HTML File

- An HTML file consists of the text to be displayed on the page, interspersed with various commands enclosed in angle brackets, which are known as ***tags***.
- HTML tags usually occur in pairs. The opening tag begins with the name of the tag. The corresponding closing tag has the same name preceded by a slash. The effect of the tag applies to everything between the opening and closing tag.
- The only HTML tags you will need to use for most of the course appear in the template on the next page, which describes the structure of the HTML index file, which is conventionally called **`index.html`**.

# Standard `index.html` Pattern

- The following components of **`index.html`** are standardized:
  - Every file begins with a **`<!DOCTYPE html>`** tag
  - The entire content is enclosed in **`<html>`** and **`</html>`** tags.
  - The file begins with a **`<head>`** section that specifies the title and JavaScript files to load.
  - The file includes a **`<body>`** section that specifies the page.

```
<!DOCTYPE html>
<html>
  <head>
    <title>title of the page</title>
    One or more script tags to load JavaScript code.
  </head>
  <body onload="function()">
    Contents of the page, if any.
  </body>
</html>
```

# Creating the JavaScript Program File

- The first step in running a JavaScript program is creating a file that contains the definitions of the functions, along with comments that give human readers a better understanding of what the program does.
- Here, for example, is the complete **`HelloWorld.js`** file:

```
/*
 * File: HelloWorld.js
 * -------------------
 * This program displays "hello, world" on the console.  It
 * is inspired by the first program in Brian Kernighan and
 * Dennis Ritchie's classic book, The C Programming Language.
 */

function HelloWorld() {
   console.log("hello, world");
}
```

# Creating the HTML File (Version 1)

- A simple HTML file that loads the **`HelloWorld.js`** program looks like this:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Hello World</title>
  <script type="text/javascript" src="HelloWorld.js"></script>
 </head>
 <body onload="HelloWorld()"></body>
</html>
```

- This file asks the browser to load the file **`HelloWorld.js`** and then call the function **`HelloWorld`** once the page is loaded.
- The problem with this strategy is that it is hard "to find out where your output went" as Kernighan and Ritchie advise.

# Creating the HTML File (Version 2)

- The output from the console log appears in different places in different browsers and usually requires the user to take some explicit action before it is visible.
- To make the console log easier to find, we have provided a library called **`JSConsole.js`** that redirects the console log to a much more visible area of the web page.

```
<!DOCTYPE html>
<html>
 <head>
  <title>Hello World</title>
   <script type="text/javascript" src="JSConsole.js"></script>
   <script type="text/javascript" src="HelloWorld.js"></script>
 </head>
 <body onload="HelloWorld()"></body>
</html>
```

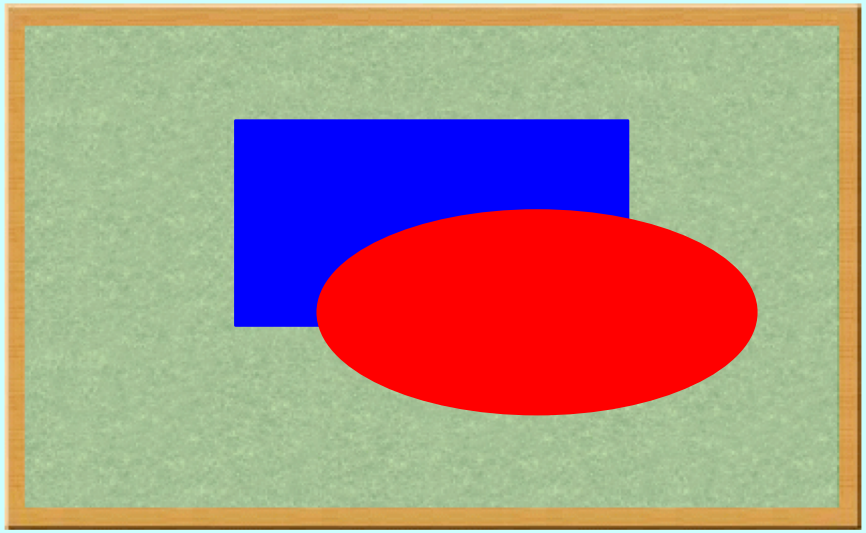
# Simple Graphics

- In addition to **`JSConsole.js`**, CS 106AJ also supports a library called **`JSGraphics.js`** that makes it easy to write graphical programs.
- The structure of the **`index.html`** file for graphics programs is similar to the one used for **`HelloWorld`**. The **`BlueRectangle`** program introduced later uses the following **`index.html`**:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Blue Rectangle</title>
  <script type="text/javascript" src="JSGraphics.js"></script>
  <script type="text/javascript" src="BlueRectangle.js"></script>
 </head>
 <body onload="BlueRectangle()"></body>
</html>
```

# The Graphics Model

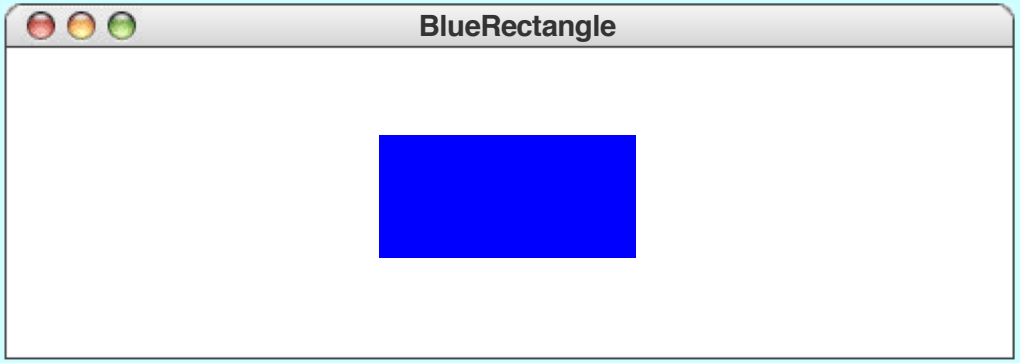
- The `JSGraphics.js` library uses a graphics model based on the metaphor of a ***collage***.
- A collage is similar to a child's felt board that serves as a backdrop for colored shapes that stick to the felt surface. As an example, the following diagram illustrates the process of adding a blue rectangle and a red oval to a felt board:



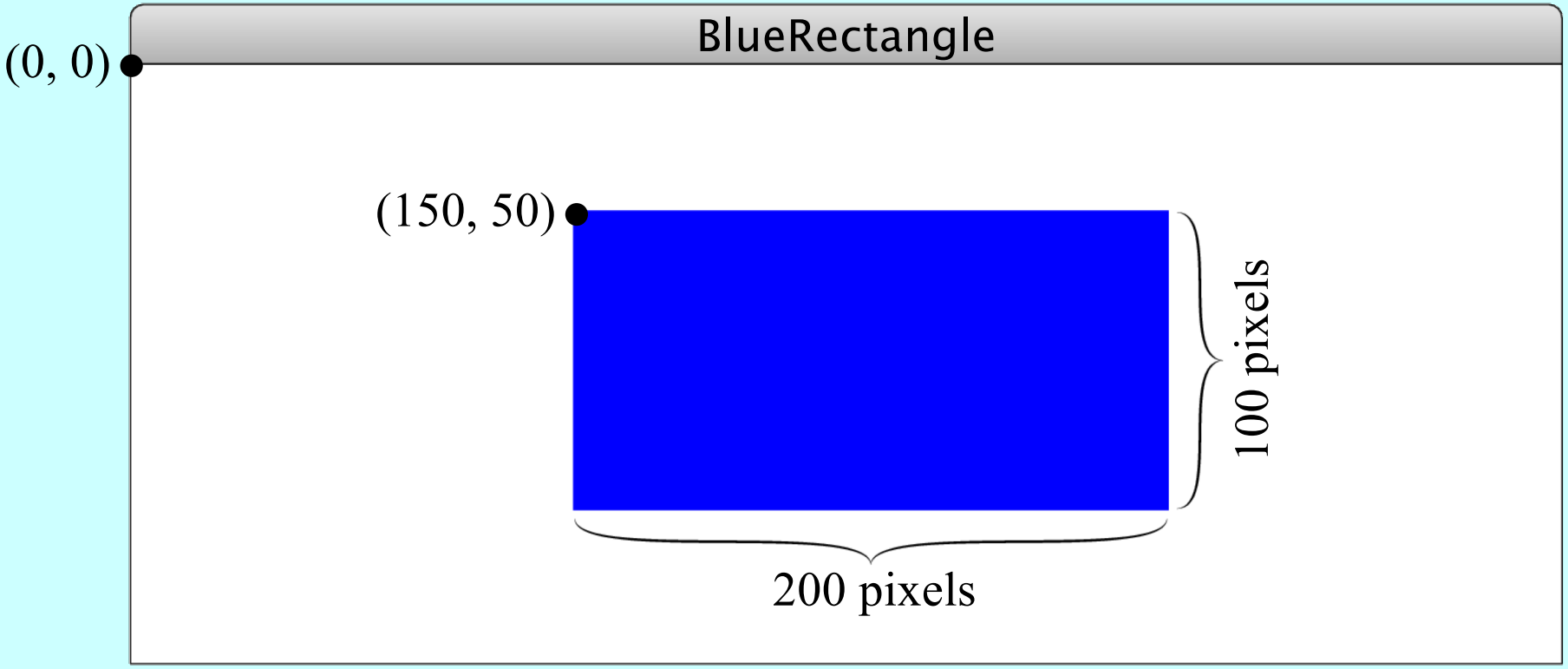
- Note that newer objects can obscure those added earlier. This layering arrangement is called the ***stacking order***.

# The `BlueRectangle` Program

```
function BlueRectangle() {
   let gw = GWindow(500, 200);
   let rect = GRect(150, 50, 200, 100);
   rect.setColor("Blue");
   rect.setFilled(true);
   gw.add(rect);
}
```

# The JavaScript Coordinate System



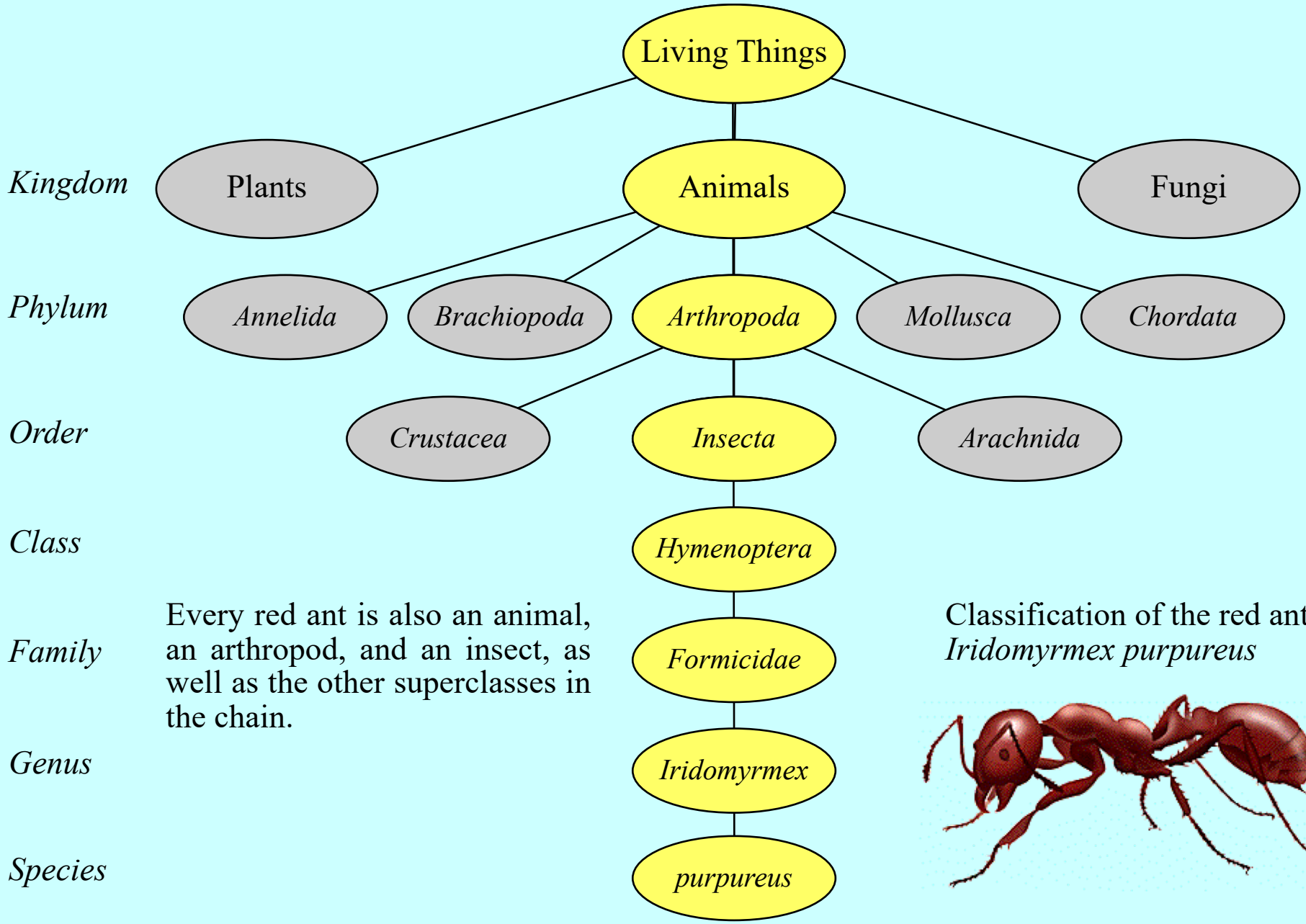

- Positions and distances on the screen are measured in terms of ***pixels***, which are the small dots that cover the screen.
- Unlike traditional mathematics, JavaScript defines the ***origin*** of the coordinate system to be in the upper left corner. Values for the *y* coordinate increase as you move downward.

# Systems of Classification

- In the mid-18th century, the Scandinavian botanist Carl Linnaeus revolutionized the study of biology by developing a new system for classifying plants and animals in a way that revealed their structural relationships and paved the way for Darwin's theory of evolution a century later.
- Linnaeus's contribution was to recognize that organisms fit into a hierarchy in which the placement of individual species reflects their anatomical similarities.

**Carl Linnaeus (1707–1778)**

# Biological Class Hierarchy




Every red ant is also an animal, an arthropod, and an insect, as well as the other superclasses in the chain.

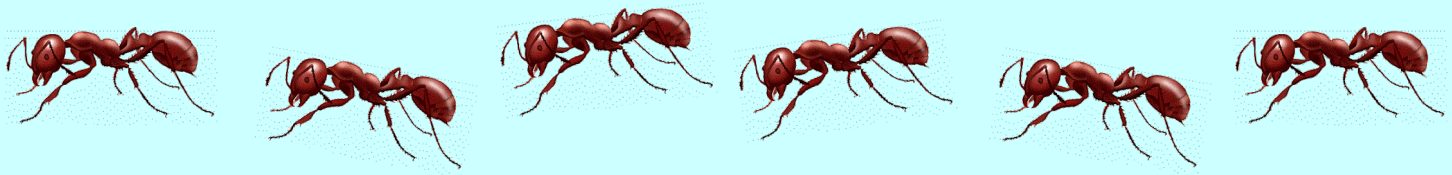
Classification of the red ant *Iridomyrmex purpureus*

# Instances *vs.* Patterns

*Drawn after you, you pattern of all those.*

—William Shakespeare, Sonnet 98

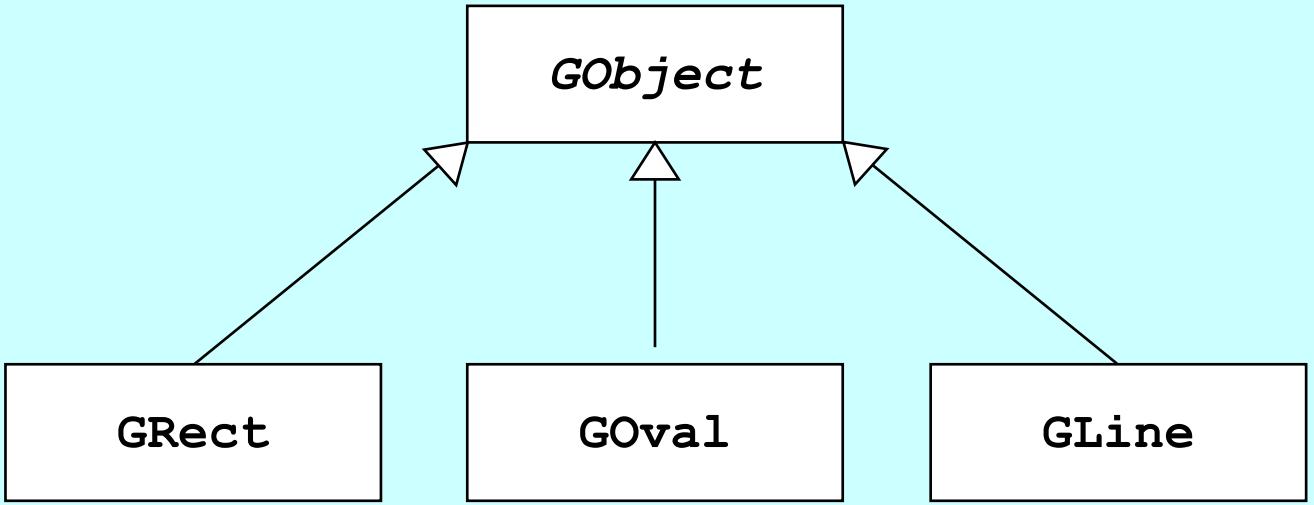
- In thinking about any classification scheme—biological or otherwise—it is important to draw a distinction between a class and specific instances of that class. In the most recent example, the designation *Iridomyrmex purpureus* is not itself an ant, but rather a ***class*** of ant. There can be (and usually are) many ants, each of which is an individual of that class.



- Each of these fire ants is an ***instance*** of a particular class of ants. Each instance is of the species *purpureus,* the genus *Iridomyrmex,* the family *Formicidae* (which makes it an ant), and so on. Thus, each ant is not only an ant, but also an insect, an arthropod, and an animal.

# The `GObject` Hierarchy

- The classes that represent graphical objects form a hierarchy, part of which looks like this:




- The **`GObject`** class represents the collection of all graphical objects.
- The three subclasses shown in this diagram correspond to particular types of objects: rectangles, ovals, and lines. Any **`GRect`**, **`GOval`**, or **`GLine`** is also a **`GObject`**.

# Creating a `GWindow` Object

- The first step in writing a graphical program is to create a window using the following function declaration, where *width* and *height* indicate the size of the window:

```
let gw = GWindow(width, height);
```

- The following operations apply to a `GWindow` object:

| |
|---|
| **`gw.add(`***object***`)`**<br>Adds an object to the window. |
| **`gw.add(`***object***`, `***x***`, `***y***`)`**<br>Adds an object to the window after first moving it to (*x*, *y*). |
| **`gw.remove(`***object***`)`**<br>Removes the object from the window. |
| **`gw.getWidth()`**<br>Returns the width of the graphics window in pixels. |
| **`gw.getHeight()`**<br>Returns the height of the graphics window in pixels. |

# Operations on the `GObject` Class

- The following operations apply to all **`GObject`**s:

| |
|---|
| *object*`.getX()`<br>Returns the *x* coordinate of this object. |
| *object*`.getY()`<br>Returns the *y* coordinate of this object. |
| *object*`.getWidth()`<br>Returns the width of this object. |
| *object*`.getHeight()`<br>Returns the height of this object. |
| *object*`.setColor(`*color*`)`<br>Sets the color of the object to the specified color. |

- All coordinates and distances are measured in pixels.
- Each color is a string, such as **`"Red"`** or **`"White"`**. The names of the standard colors are defined in Figure 4-5 on page 125.

# Drawing Geometrical Objects

Functions to create geometrical objects:

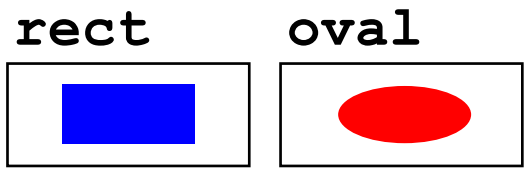
| |
|---|
| **`GRect(`** *x*`,` *y*`,` *width*`,` *height*`)`<br>Creates a rectangle whose upper left corner is at ($x$, $y$) of the specified size. |
| **`GOval(`** *x*`,` *y*`,` *width*`,` *height*`)`<br>Creates an oval that fits inside the rectangle with the same dimensions. |
| **`GLine(`** $x_0$`,` $y_0$`,` $x_1$`,` $y_1$`)`<br>Creates a line extending from ($x_0$, $y_0$) to ($x_1$, $y_1$). |

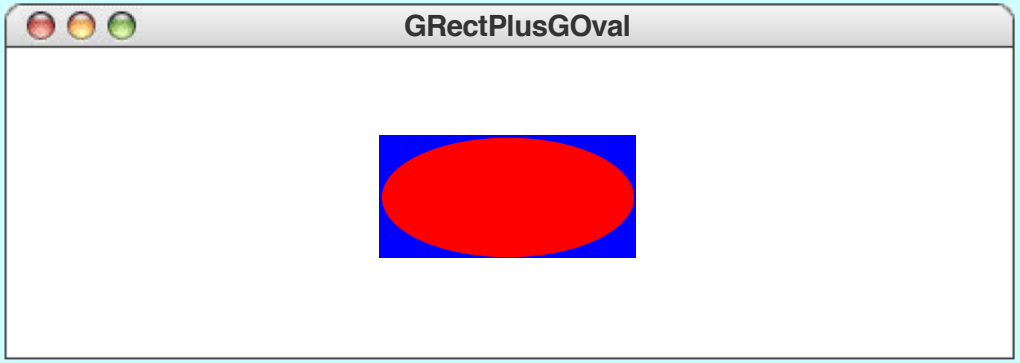
Methods shared by the **`GRect`** and **`GOval`** classes:

| |
|---|
| *object*`.`**`setFilled(`***fill***`)`<br>If *fill* is **`true`**, fills in the interior of the object; if **`false`**, shows only the outline. |
| *object*`.`**`setFillColor(`***color***`)`<br>Sets the color used to fill the interior, which can be different from the border. |

# The GRectPlusGOval Program

```
function GRectPlusGOval() {
   let gw = GWindow(500, 200);
   let rect = GRect(150, 50, 200, 100);
   rect.setFilled(true);
   rect.setColor("Blue");
   gw.add(rect);
   let oval = GOval(150, 50, 200, 100);
   oval.setFilled(true);
   oval.setColor("Red");
   gw.add(oval);
}
```



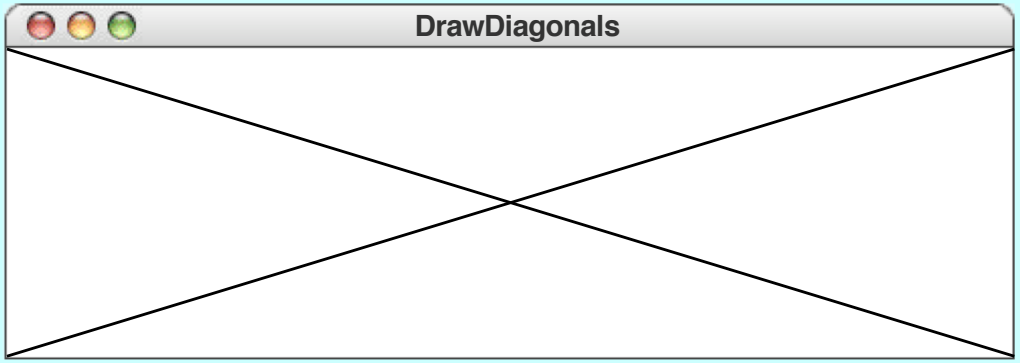



# The DrawDiagonals Program

```
/* Constants */

const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 200;

function DrawDiagonals() {
   let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   gw.add(GLine(0, 0, GWINDOW_WIDTH, GWINDOW_HEIGHT));
   gw.add(GLine(0, GWINDOW_HEIGHT, GWINDOW_WIDTH, 0));
}
```



The End