Strings and Graphics

Strings and Graphics

Abraham Gebrekidan
EIP 01
July 31, 2024

The String Type

- One of the most important data types in any programming language is the ***string type***.
- The domain of the string type is all sequences of characters. In JavaScript, you create a string simply by including that sequence of characters inside quotation marks, as in `"Abraham"`.
- The set of operations that can be applied to strings is large, but you don't need to know the entire set. In fact, for the first few weeks in this course, the only string operation you need to know is concatenation, as described on the next slide. You will learn about other operations in next sessions.
- All values—including numbers, strings, graphical objects, and values of many other types—can be assigned to variables, passed as arguments to functions, and returned as results.

Nonnumeric Data

- The arithmetic expressions in the previous session enable you to perform numeric computation. Much of the excitement of modern computing, however, lies in the ability of computers to work with other types of data, such as characters, images, sounds, and video.
- As you will learn in future sessions, all of these data types are represented inside the machine as sequences of binary digits, or ***bits***. When you are getting started with programming, it is more important to think about data in a more abstract way in which you focus on the conceptual values rather than the underlying representation.

Concatenation

- One of the most useful operations available for strings is ***concatenation***, which consists of combining two strings end to end with no intervening characters.
- Concatenation is built into JavaScript using the `+` operator. For example, the expression `"ABC" + "DEF"` returns the string `"ABCDEF"`.
- If you use `+` with numeric operands, it signifies addition. If at least one of its operands is a string, JavaScript interprets `+` as concatenation. It automatically converts the other operand to a string and concatenates the two strings, so that

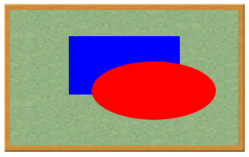
  `"Catch" + -22` ⇒ `"Catch-22"`

Data Types

- The notion that data values come in many different forms gives rise to the notion of a ***data type***, which defines the common characteristics of data values that have a particular form or purpose.
- In computer science, each data type is defined by a ***domain***, which is the set of values that belong to that type, and a ***set of operations***, which shows how the values in that domain can be manipulated.
- For example, the JavaScript type for numbers has a domain that consists of numeric values like 1.414213 or 42. The set of operations includes addition, subtraction, multiplication, division, remainder, and a few more that you haven't learned yet.

The Graphics Model

- SJS uses the same graphics model that we have used for the last decade, which is 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:



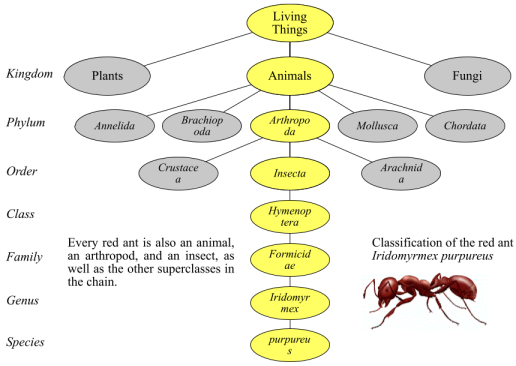
- Note that newer objects can obscure those added earlier. This layering arrangement is called the ***stacking order***.

The `BlueRectangle` Program

```
import "graphics";

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

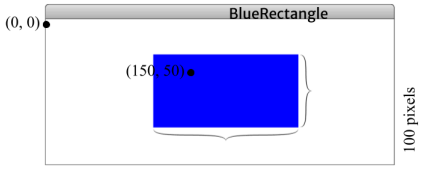
Biological Class Hierarchy



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*

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.

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.

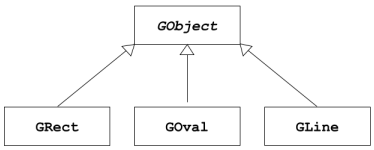
Systems of Classification

- 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)

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:

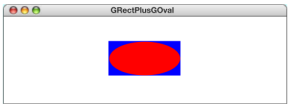
```
var gw = GWindow(width, height);
```

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

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

The `GRectPlusGOval` Program

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



Operations on the `GObject` Class

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

object`.getX()` Returns the *x* coordinate of this object.
object`.getY()` Returns the *y* coordinate of this object.
object`.getWidth()` Returns the width of this object.
object`.getHeight()` Returns the height of this object.
object`.setColor(`*color*`)` 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"`.

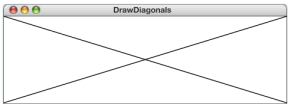
The `DrawDiagonals` Program

```
import "graphics";

/* Constants */

const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 200;

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



Drawing Geometrical Objects

Functions to create geometrical objects:

`GRect(`*x***`, `***y***`, `*width*`, `*height*`)`** Creates a rectangle whose upper left corner is at (*x*, *y*) of the specified size.
`GOval(`*x***`, `***y***`, `*width*`, `*height*`)`** Creates an oval that fits inside the rectangle with the same dimensions.
`GLine(`x_0`, `y_0`, `x_1`, `y_1`)` 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*`)` If *fill* is `true`, fills in the interior of the object; if `false`, shows only the outline.
object`.setFillColor(`*color*`)` Sets the color used to fill the interior, which can be different from the border.

The End