

JavaScript Graphics Library Reference

Written by Ben Yan, from materials by Jerry Cain, Eric Roberts, and Jonathan Kula.

This handout records most of the widely used and scintillating methods found in our course's JavaScript graphics library. Note that this album isn't fully comprehensive, so feel free to ask me for rarer methods!

As terminology, **getter methods** only access information from the object and do not modify its data internally, whereas **setter methods** do modify the object, e.g., changing the width, color, position, etc.

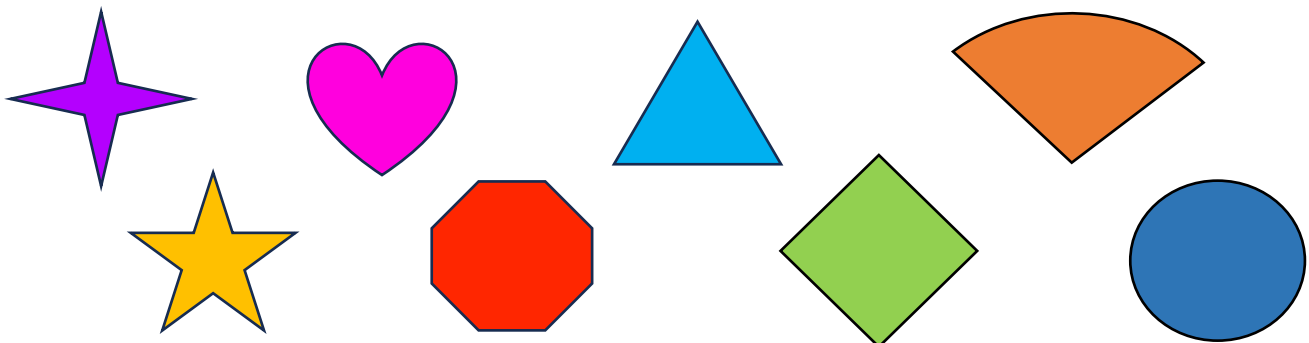
To summon the graphics library for use, add the following line to the **<head>** section of the HTML file.

```
<script type='text/javascript'
  src='https://cs106ax.stanford.edu/jslib/JSGraphics.js'></script>
```

Here's a table of contents for each graphical class and their corresponding documentation ☺.

Class	Pages
GWindow	2 – 3
GObject	3 – 4
GRect	5
GOval	6
GLine	7
GPoint	8
GPolygon	8 – 9
GArc	9 – 10
GImage	11
GLabel	12 – 13
GCompound	14 – 15
GMath	16

"I would like to make a film to tell children, it's good to be alive!" – Hayao Miyazaki



GWindow Class

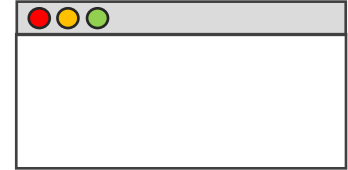
GWindow: Constructor Functions

function GWindow(width, height)

Creates a new GWindow object with a given width and height, displaying it on the HTML page, e.g.,

```
const GWINDOW_WIDTH = 1000;
const GWINDOW_HEIGHT = 500;

let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
console.log(gw.getHeight()); // 500
```



GWindow: Getter Methods

getWidth()

Returns the width of the window in pixels.

getHeight()

Returns the height of the window in pixels.

getElementAt(x,y)

Returns the topmost (i.e. most recently added) GObject that overlaps with the specified coordinate. If no objects overlap, this **returns** the JavaScript value **null**.

GWindow: Setter Methods

add(gObject)

Adds the specified graphics object to the window without changing its location.

add(gObject, x, y)

Adds the specified graphics object to the window, **and sets its location to the point (x, y)**.

remove(gObject)

Removes the specified graphics object from this window.

clear()

Removes all graphics objects from this window.

addEventListener(eventType, listener)

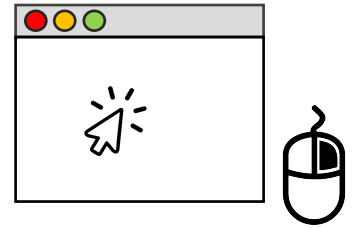
Stores a mouse listener in this window, so that when the window receives an event of the given **eventType**, e.g., "**click**", it executes the **listener** function and passes in an event object **e**.

The general template for installing a listener function is:

```
let listener = function(e) {
    code that executes each and every time this event occurs
}
gw.addEventListener(eventType, listener);
```

Here is a track list of mouse event types that our graphics library supports.

"click"	The user clicks the mouse in the window.
"dblclick"	The user double-clicks the mouse.
"mousedown"	The user presses the mouse button.
"mouseup"	The user releases the mouse button.
"mousemove"	The user glides the mouse with the button up.
"drag"	The user drags the mouse with the button down.



Note: Certain user actions can spark more than one mouse event, e.g., clicking the mouse generates a "mousedown" event, a "mouseup" event, and a "click" event, in that order. Furthermore, the listener function accepts a parameter **e**, which is the event object, and supports 2 key methods to know.

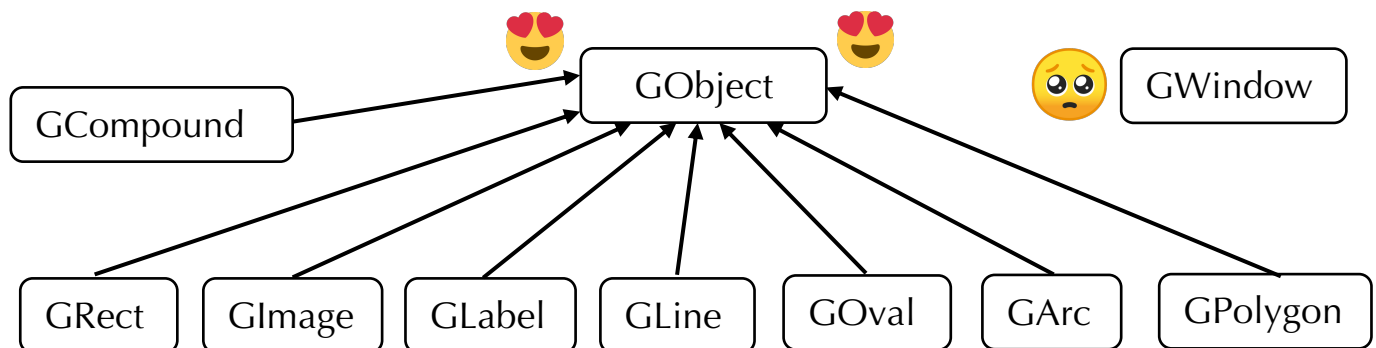
e: Event Object Methods	
getX()	
Returns the x coordinate of the mouse in the window when the event was triggered, via e.getX()	
getY()	
Returns the y coordinate of the mouse in the window when the event was triggered, via e.getY()	

The library also supports a "keydown" event, which is invoked whenever a user clicks a keyboard key. In this case, the event object **e** that's passed into the listener function has a string property **e.code** that expresses the key present, e.g., "Enter", "Backspace", "Return", letter keys "a", "A", "B", etc.

GObject Class

The **GObject** is the base template or common superclass daddy for nearly all other graphics objects. This means that you essentially never create a literal **GObject**, but rather a **GRect**, **GOval**, **GLine**, etc.

GObject: Constructor Functions	
function GObject()	
Creates an empty GObject. You probably shouldn't be doing this , and instead should be using the specific constructor method of the GObject type, e.g., GRect , that you want to create. But it exists...	



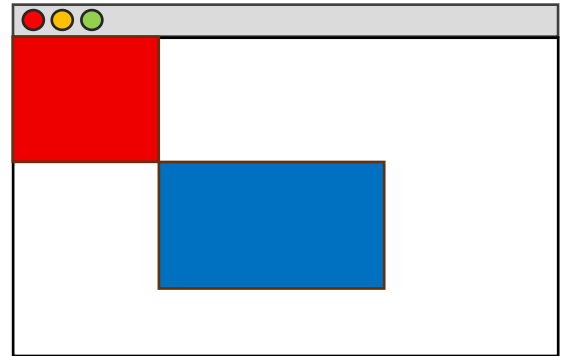
However, since the graphics objects are internally designed from this shared **GObject** blueprint, **they inherit the following methods, which thus can be used on GRect, GOval, GLine objects, and so on.**

GObject: Getter Methods	
getX()	
Returns the x coordinate of this object's origin.	
getY()	
Returns the y coordinate of this object's origin.	
getWidth()	
Returns the width of this object in pixels.	
getHeight()	
Returns the height of this object in pixels.	
getColor()	
Returns the color of this object as previously set. The color is either a predefined string, e.g. " red ", " green ", or a red-green-blue hex string of form " #RRGGBB ", e.g., " #40E0D0 " for turquoise.	
contains(x, y)	
Returns true if the space occupied by this object contains the given coordinate, otherwise false .	
isFilled()	
Returns true if this object is color-filled, and false otherwise.	

GObject : Setter Methods	
move(dx, dy)	
Moves the object on the screen using the displacements dx and dy, e.g., x += dx , and y += dy .	
movePolar(r, theta)	
Moves the object on screen using displacements given in polar coordinates . That is, the object moves a total distance of r pixels, at an angle of theta degrees counterclockwise from the positive x-axis.	
setLocation(x, y)	
Sets the location of this object to the point (x,y) .	
setColor(color)	
Sets the border color of this object. If the object is filled, but hasn't been assigned a specific fill color, this automatically sets the fill color as well. See getColor above for the format of the color string.	
setFillColor(color)	
Sets the interior fill color of this object specifically. See getColor above for the input color format.	
setFilled(flag)	
Sets this object's interior to be filled if flag is true , or not filled if flag is false . By default, an object is not filled, so make sure to call this if you do intend to pigment the object with its fill color.	

GRect Class

GRect: Constructor Functions
function GRect(width, height)
Creates a new GRect object with the specified width and height, positioned at the origin (0,0).
function GRect(width, height, x, y)
Creates a new GRect with the specified width and height, and its top-left corner positioned at (x,y).



This code sample creates the visage on the above screen, using the constructors and graphical methods.

```
let gw = GWindow(1000, 500); // width is 1000, height is 500 pixels

let redSquare = GRect(200, 200);
redSquare.setFillColor("red");
redSquare.setFilled(true);
gw.add(redSquare)

let blueRect = GRect(400, 200, 200, 200); // 400 by 200 rectangle
blueRect.setFillColor("blue");
blueRect.setFilled(true);
gw.add(blueRect); // top-left coordinate placed at (200, 200)
```

The below methods apply specifically for **GRect** and **GOval** objects.

GRect: Setter Methods
setSize(width, height)
Sets the width and height of this GRect as specified.
setBounds(x, y, width, height)
Sets the width and height , as well as the GRect's top-left coordinate (x,y), as specified.

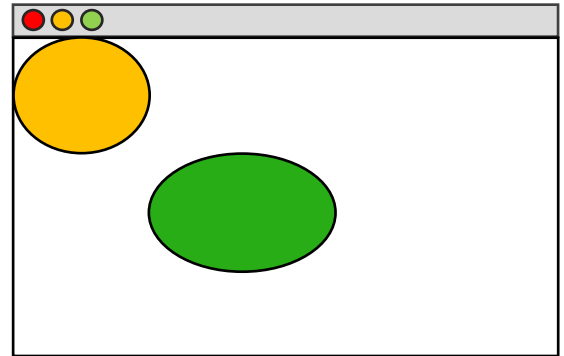
These next methods are specified by the **GObject** blueprint, and are thus inherited for use on **GRects**.

getX()	move(dx, dy)
getY()	movePolar(r, theta)
getWidth()	setLocation(x, y)
getHeight()	setColor(color)
getColor()	setFillColor(color)
contains(x, y)	setFilled(flag)
isFilled()	

You can read more about their documentation in the **GObject** section! ☺

GOval Class

GOval: Constructor Functions
function GOval(width, height)
Creates a new oval object with the specified width and height, positioned at the origin (0,0).
function GOval(width, height, x, y)
Creates a new oval with the specified width and height, and its top-left corner positioned at (x,y).



This code sample creates the visage on the above screen, using the constructors and graphical methods.

```
let gw = GWindow(1000, 500); // width is 1000, height is 500 pixels

let sun = GOval(200, 200);
sun.setFillColor("yellow");
sun.setFilled(true);
gw.add(sun);

let greenStar = GOval(400, 200, 200, 200); // 400 by 200 oval
greenStar.setFillColor("green");
greenStar.setFilled(true);
gw.add(greenStar); // top-left coordinate placed at (200, 200)
```

The below methods apply specifically for **GRect** and **GOval** objects.

GOval: Setter Methods
setSize(width, height)
Sets the width and height of this GRect as specified.
setBounds(x, y, width, height)
Sets the width and height , as well as the GRect's top-left coordinate (x,y), as specified.

These next methods are specified by the **GObject** blueprint, and are thus inherited for use on **GOvals**.

<code>getX()</code>	<code>move(dx, dy)</code>
<code>getY()</code>	<code>movePolar(r, theta)</code>
<code>getWidth()</code>	<code>setLocation(x, y)</code>
<code>getHeight()</code>	<code>setColor(color)</code>
<code>getColor()</code>	<code>setFillColor(color)</code>
<code>contains(x, y)</code>	<code>setFilled(flag)</code>
<code>isFilled()</code>	

You can read more about their documentation in the **GObject** section! ☺

GLine Class

GLine: Constructor Functions

function Gline(x0, y0, x1, y1)

Creates a new **GLine** line segment object connecting points **(x0, y0)** and **(x1, y1)**.



This code sample creates the visage on the above screen, using the constructor and graphical methods.

```
let gw = GWindow(800, 600); // width is 800, height is 600 pixels

let line = GLine(0, 0, 400, 300); // line from (0, 0) to (400, 300)
line.setColor("magenta");
gw.add(line);
```

GLine: Getter Methods

distanceSquared()

Returns the length of this line, **squared**, via the formula $(x1 - x0)^2 + (y1 - y0)^2$. To get the true length, you should take the square root, e.g., `Math.sqrt(line.distanceSquared())`

getStartPoint()

Returns the x and y coordinates of the start point as a **GPoint**. For the **GLine** above, we could write

```
let startPoint = line.getStartPoint();
let startX = startPoint.getX(); // 0
let startY = startPoint.getY(); // 0
```

getEndPoint()

Returns the x and y coordinates of the end point as a **GPoint**. For the **GLine** above, we could write

```
let endPoint = line.getEndPoint();
let endX = endPoint.getX(); // 400
let endY = endPoint.getY(); // 300
```

GLine: Setter Methods

setStartPoint(x, y)

Sets the initial point and origin of this line to **(x,y)**, without changing the position of the end point.

setEndPoint(x, y)

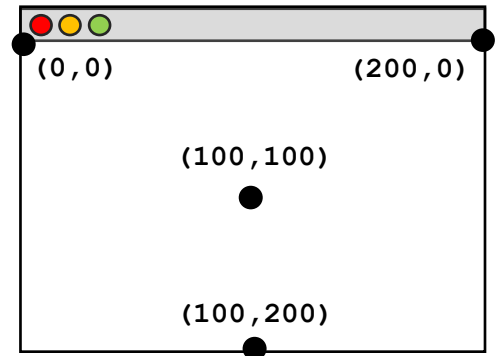
Sets the end point of this line to **(x,y)**, without changing the position of the start point or the origin.

Note that **GLine** objects also inherit the **getColor** and **setColor(color)** methods from **GObject**, as well as **contains(x,y)** to see if point **(x,y)** is contained in the line segment.

GPoint Class

This is a utility class that just stores an **x** and **y** coordinate.

GPoint: Constructor Functions	
function GPoint()	Creates a new point representing the origin (0,0).
function GPoint(x, y)	Creates a new point representing the given (x, y).

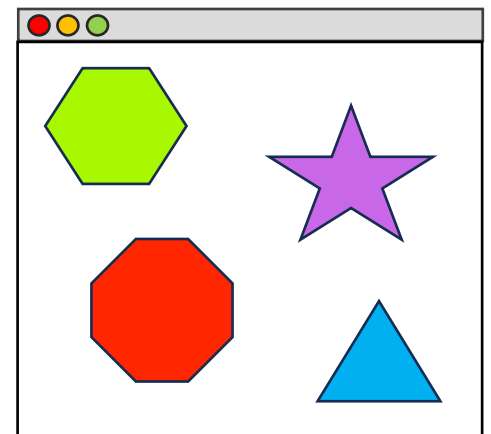


GPoint: Getter Methods	
getX()	Returns the x component of this GPoint object.
getY()	Returns the y component of this GPoint object.

GRect: Setter Methods	
setLocation(x, y)	Alter this's GPoint object's location to be the newly specified (x, y).
translate(dx, dy)	Adds displacements dx and dy to the x and y components, respectively. of this GPoint object.

GPolygon Class

GPolygon: Constructor Functions	
function GPolygon()	Creates a new empty polygon object with its origin at (0,0).
function GPolygon(x, y)	Creates a new empty polygon object with its origin at (x,y).
function GPolygon(points)	Creates a polygon object from an array of GPoint objects.



GPolygon: Getter Methods	
getCurrentPoint(x, y)	Returns a GPoint object representing the last point that was recently added to this GPolygon .

GPolygon: Setter Methods**addVertex(x, y)**

Adds a vertex at **(x, y)** relative to the polygon's origin (that is, **x** pixels to the right and **y** pixels downward from the polygon's origin). Note this is not necessarily **(x, y)** on the graphics window.

addEdge(dx, dy)

Adds a vertex that is shifted by **dx** and **dy** pixels to the right and downward, respectively, from wherever the last place was placed—or from the polygon's origin if this is the first vertex to be added.

addPolarEdge(r, theta)

Adds a vertex that is shifted by **r** pixels in the direction **theta** (where **theta** is measured in degrees counterclockwise from the positive x axis) from wherever the last point was placed—or from the polygon's origin if this is the first vertex to be added.

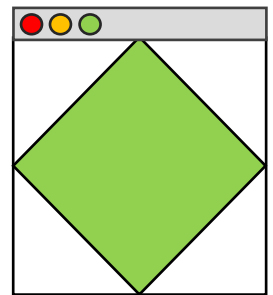
The code below creates a baseball diamond. Its vertices are defined so that the polygon's origin runs through its center, and the diamond is added to the window to have its origin be at the window's center.

```
let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);

let diamond = GPolygon();
diamond.addVertex(GWINDOW_WIDTH / 2, 0); // right corner
diamond.addVertex(0, GWINDOW_HEIGHT / 2); // top corner
diamond.addVertex(-GWINDOW_WIDTH / 2, 0); // left corner
diamond.addVertex(0, -GWINDOW_HEIGHT / 2); // bottom corner

diamond.setFilled(true);
diamond.setFillColor("green");

gw.add(diamond, GWINDOW_WIDTH / 2, GWINDOW_HEIGHT / 2);
```

**GArc Class**

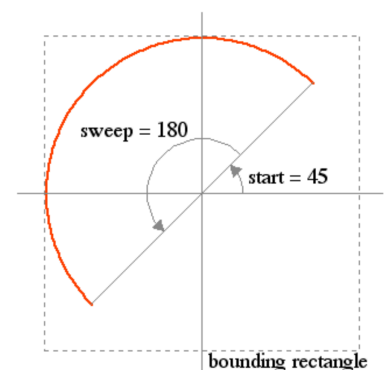
The **GArc** class represents an arc formed by taking a section from the perimeter of an oval. An arc is defined by its **start angle**, its **sweep angle**, and its bounding box, i.e. the top-left coordinate **(x,y)**, and width and height of its corresponding oval.

GArc: Constructor Functions**function GArc(width, height, start, sweep)**

Creates a new GArc while procrastinating setting the arc's location.

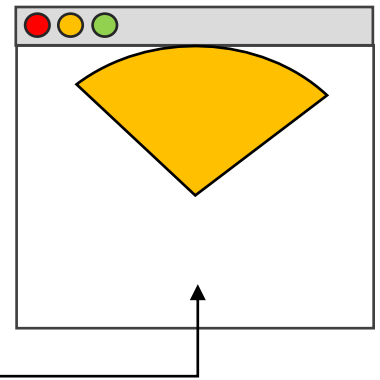
function GOval(x, y, width, height, start, sweep)

Creates a new GArc with its six relevant geometric parameters.



Note: Angles are measured in degrees going counterclockwise from the positive x axis. Negative angles signify a clockwise direction.

```
let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
let start = 45;
let sweep = 90;
let arc = GArc(0, 0, gw.getWidth(), gw.getHeight(),
               start, sweep);
arc.setFillColor("yellow");
arc.setFilled(true);
gw.add(arc);
```



GArc: Getter Methods

getStartAngle()

Returns the starting angle for this **GArc** object.

getSweepAngle()

Returns the sweep angle for this **GArc** object.

getStartPoint()

Returns the x and y coordinates of where the arc **begins** on the circumference as a **GPoint**.

getEndPoint()

Returns the x and y coordinates of where the arc **ends** on the circumference as a **GPoint**.

GArc : Setter Methods

setStartAngle(angle)

Sets the starting angle in degrees for this **GArc** object.

setSweepAngle(angle)

Sets the sweep angle in degrees for this **GArc** object.

setFrameRectangle(x, y, width, height)

Sets this **GArc**'s bounding box to match the coordinates and dimensions given.

These methods are specified by the **GObject** blueprint, and are thus inherited for use on **GArc** objects.

getX()	move(dx, dy)
getY()	movePolar(r, theta)
getWidth()	setLocation(x, y)
getHeight()	setColor(color)
getColor()	setFillColor(color)
contains(x, y)	setFilled(flag)
isFilled()	

You can read more about their documentation in the **GObject** section! ☺

GImage Class

GImage: Constructor Functions
function GImage(source)
Creates a new image object from the given source (e.g., image URL, local file name, e.g., " cat.png "), positioned at the origin (0,0) .
function GImage(source, x, y)
Creates a new image object from the given source (e.g., URL, local file name, e.g., " cat.png "), positioned at the given (x,y) coordinate.



The **GImage** is loaded asynchronously, which means your code will start running while the **GImage** is still loading in. Because the program won't know the width or height of the image until it's fully loaded in, it's recommended to add a "**load**" event listener that waits for the image to load in its entirety, e.g.,

```
let image = GImage("JerryFormal.png");

let onceImageLoaded = function() { // executes only after image is loaded
    image.scale(10); // make Jerry BIG
}

image.addEventListener("load", onceImageLoaded);
gw.add(image);
```

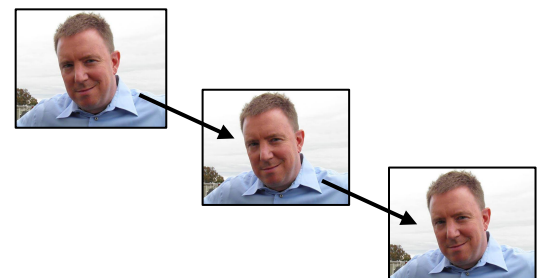
GImage: Setter Methods
addEventListener("load", listener);
Adds the given listener to this image, not the GWindow. The only valid event type here is " load ", and only when the image has been fully loaded does the listener function execute.
scale(factor)
Scales the image on-screen by the scale / multiplier factor , which applies to both dimensions.
scale(xFactor, yFactor)
Scales the image on-screen by xFactor and yFactor times in the x and y dimensions, respectively.

There are image methods to access & manipulate individual pixels, though you won't need them for this course ☺. You're also welcome to modify and resize images elsewhere before loading into JavaScript!

Additionally, **GImage** objects inherit the following methods from the **GObject** blueprint.

```
getX()
getY()
getWidth()
getHeight()
contains(x, y)

move(dx, dy)
movePolar(r, theta)
setLocation(x, y)
```





GLabel

The **GLabel** class enables you to display text on the graphics window.

GImage: Constructor Functions

function GLabel(text)

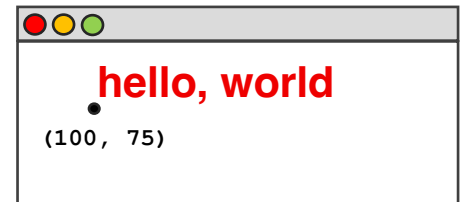
Creates a new GLabel object displaying the given string text, positioned at the origin (0,0).

function GImage(source, x, y)

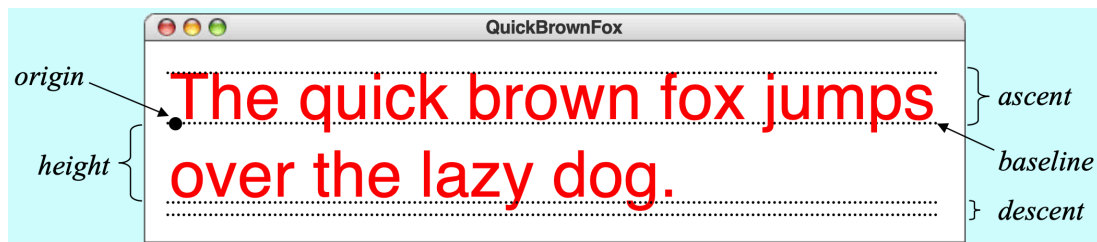
Creates a new GLabel object displaying the given string text, positioned at the given (x,y).

The program below etches the string "hello, world" on the graphics window.

```
let gw = GWindow(500, 200);
let label = GLabel("hello, world", 100, 75);
label.setFont("36px Helvetica");
label.setColor("Red");
gw.add(label);
```



Note: A **GLabel**'s origin (x,y) differs slightly from other **GObjects**, as illustrated above, and relies on the following concepts. The **baseline** is the imaginary line on which the characters rest, and the **origin is the point on the baseline at which the label begins**, i.e. approximately the bottom-left corner of the first line.



GLabel: Getter Methods

getFont()

Returns the font this label is using. The font string is formatted as a CSS fragment, e.g.,

- "italic 36px Helvetica" ○ "bold 14px Courier, 'Courier New', Monaco"
- "24px 'Times New Roman'" ○ "oblique bold 44px 'Lucida Blackletter', serif"

or a garden variety of other fonts described in our JavaScript textbook, pp. 129-131 ☺.

getLabel()

Returns the text currently displayed by this **GLabel** as a string, e.g., "hello, world".

getDescent()

Returns the descent of this text font, which is the maximum distance in pixels that characters **extend below the baseline**.

getAscent()

Returns the ascent of this text font, which is the maximum distance in pixels that characters **extend above the baseline**.

getTextAlign()
Returns the current horizontal text alignment of this font, which is either "left" , "right" , or "center" . The default value is "start" , which is "left" in left-to-right languages, e.g., English, French, Swahili, Hindi, Thai, and "right" in right-to-left languages, e.g., Arabic, Hebrew, Farsi.
getBaseline()
Returns the current baseline / vertical alignment of this font, which can take on a range of possible values, typically "top" , "middle" , "bottom" , or the default value "alphabetic" .

GLabel: Setter Methods
setFont(font)
Sets this GLabel 's font to the specified font string. See getFont() above for the CSS formatting.
setLabel(str)
Sets this GLabel object to display the string specified by str .
setTextAlign(alignment)
Sets this GLabel 's horizontal text alignment, see getTextAlign() above for the options.
setBaseline(baselineAlignment)
Sets this GLabel 's baseline or vertical text alignment, see getBaseline() above for the options.

These methods are specified by the **GObject** blueprint, and are thus inherited for use on **GArc** objects.

getX()	move(dx, dy)	
getY()	movePolar(r, theta)	
getWidth()	setLocation(x, y)	
getHeight()	setColor(color)	
getColor()		

Note that **.getX()** and **.getY()** return the x and y coordinates, respectively, of the **GLabel** origin. When using **.getHeight()**, note the height of a **GLabel** is defined as the distance in pixels between successive baselines, which is usually wider and roomier than the text itself, i.e. its ascent and descent.

GCompound

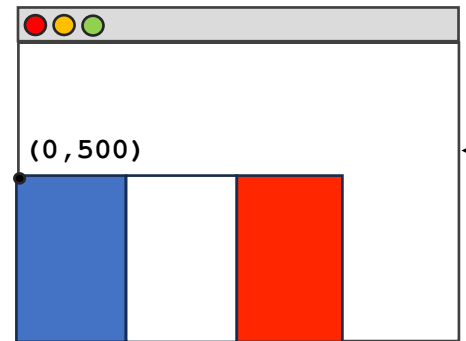
GCompound: Constructor Functions

function GCompound()

Creates a new **GCompound**, with its internal origin placed at (0, 0) when added to the graphics window.

function GCompound(x, y)

Creates a new **GCompound**, and positions its internal origin at the given coordinate (x, y) on the window.



A **GCompound** makes it possible to combine and layer several **GObjects** into one composite structure.

It's akin to a **GWindow** in that you can add objects to it, and akin to a **GObject** in that you can add it to the window. It possesses its own internal coordinate system, and you add objects to the **GCompound** relative to its internal origin. The example below displays a French flag , made up of three **GRect**s.

```
let gw = GWindow(1000, 1000);

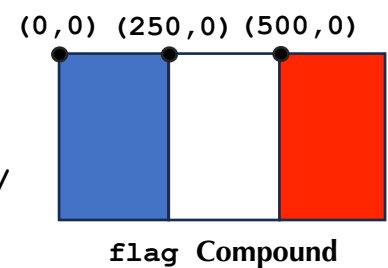
let createStripe = function(color) {
  let rect = GRect(250, 500);
  rect.setFilled(true);
  rect.setFill-color(color);
  return rect;
} /* each stripe has width 250, height 500 pixels */

let flag = GCompound();
let blueStripe = createStripe("blue");
flag.add(blueStripe, 0, 0);

let whiteStripe = createStripe("white");
flag.add(whiteStripe, 250, 0);

let redStripe = createStripe("red");
flag.add(redStripe, 500, 0);

gw.add(flag, 0, 500); // sets compound's origin to (0,500) in window
```



GCompound: Setter Methods

add(gObject)

Adds the given **gObject** to the compound, positioned at the **gObject**'s stored x and y coordinates.

add(gObject, x, y)

Adds the given **gObject** to the compound, positioned at (x,y) relative to the compound's origin.

remove(gObject)

Removes the given **gObject** from the compound.

removeAll()

Removes all the graphical objects currently stored inside this compound.

GCompound: Getter Methods**getElementAt(x, y)**

Returns the most recently added **GObject** in the compound overlapping the specified coordinate.

getElement(index)

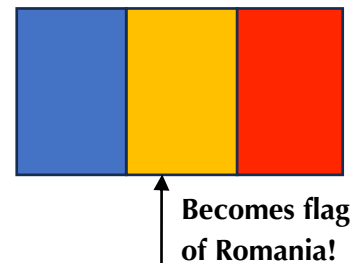
Returns the **GObject** inside the **GCompound** at the specified index, where index 0 was the first / earliest **GObject** added to the **GCompound**, and the last index is the most recently added **GObject**.

While the methods above do exist, if you need to access graphical objects stored inside a compound, it's recommended to also store them as additional variable fields inside the **GCompound** object, e.g.,

```
flag.blueStripe = blueStripe;
flag.whiteStripe = whiteStripe;
flag.redStripe = redStripe;
```

That way, if you want to modify just the white **GRect** of the compound flag, you can select just the field **flag.whiteStripe**, and change it accordingly.

```
flag.whiteStripe.setFillColor("yellow");
```



These methods are specified by the **GObject** blueprint, and are thus inherited for use on **GCompounds**.

```
getX()
```

```
getY()
```

```
getWidth()
```

```
getHeight()
```

```
move(dx, dy)
```

```
movePolar(r, theta)
```

```
setLocation(x, y)
```

Note: When you move a **GCompound**, all the graphical components inside move together in unison 😊.

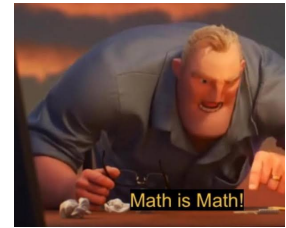
GMath Class

This class is an ecosystem of convenience methods you can use when working with graphics. Unlike the class methods you've seen earlier, these ones are **static**, which means you call them directly with the class name, e.g.,

```
let result = GMath.functionName(parameters...);
```

as opposed to first creating an object of that class, then calling the method. For example, `let rect = GRect(width, height)` followed by the line `rect.setFillColor("green")`. This makes sense here as we're only filling that specific rectangle, and there can exist many rectangles (that aren't green).

However, there is only one Math (see *meme above*), so it makes little sense to create **GMath** objects ☺.



GMath: Static Methods	
<code>GMath.round(x)</code>	
Returns	the input x rounded to the nearest integer.
<code>GMath.sinDegrees(angle)</code>	
Returns	the sine of the given angle in degrees.
<code>GMath.cosDegrees(angle)</code>	
Returns	the cosine of the given angle in degrees.
<code>GMath.tanDegrees(angle)</code>	
Returns	the tangent of the given angle in degrees.
<code>GMath.toDegrees(angle)</code>	
Returns	the degree equivalent of the input angle , which is expressed in radians.
<code>GMath.distance(x, y)</code>	
Returns	the straight-line distance from the origin to the point (x, y) .
<code>GMath.distance(x0, y0, x1, y1)</code>	
Returns	the length of the line from the point (x0, y0) to the point (x1, y1) .
<code>GMath.angle(x, y)</code>	
Returns	the angle, in degrees, from the origin to the point (x, y) . Angles are measured counterclockwise from the positive x-axis , and recall y values increase you go down in the window.
<code>GMath.angle(x0, y0, x1, y1)</code>	
Returns	the angle, in degrees, from the point (x0, y0) to the point (x1, y1) .

