A Button object type

One of the best ways to make code reusable and powerful is to use object-oriented programming, especially for making UI controls like buttons. In object-oriented programming, we think of our program world in terms of abstract object types that have particular behavior, and then we create specific instances of those object types with particular parameters. If you don't remember how to do it in JavaScript, review it here.

To use OOP to make buttons, we'll need to define a `Button` object type and then define methods on it, like to draw it and handle mouse clicks. We'd like to be able to write code that looks like this:

```
var btn1 = new Button(...);
btn1.draw();

mouseClicked = function() {
  if (btn1.isMouseInside()) {
     println("Whoah, you clicked me!");
  }
}
```

Let's contrast that to the code we wrote in the last article:

```
var btn1 = {...};
drawButton(btn1);

mouseClicked = function() {
  if (isMouseInside(btn1)) {
     println("Whoah, you clicked me!");
  }
}
```

It's very similar, isn't it? But there's a big difference -- all the functions are defined on the `Button` object type, they actually belong to the buttons. There's a tighter coupling of properties and behavior, and that tends to lead to cleaner and more reusable code.

To define the `Button` object type, we need to start with the constructor: the special function that takes in configuration parameters and sets the initial properties of the object instance.

As a first attempt, here's a constructor that takes in x, y, width, and height:

```
var Button = function(x, y, width, height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
};


var btn1 = new Button(100, 100, 150, 150);
```

That certainly works, but I have another approach I'd like to recommend. Instead of taking in individual parameters, the constructor could take in a configuration object.

```
var Button = function(config) {
    this.x = config.x;
    this.y = config.y;
    this.width = config.width;
    this.height = config.height;
    this.label = config.label;
};
```

The advantage of the config object is that we can keep adding more parameters for the constructor to handle (like `label`), and it's still easy for us to understand what each parameter does when we construct a button:

```
var btn1 = new Button({
    x: 100, y: 100,
    width: 150, height: 50,
    label: "Please click!"});
```

But we can go a step farther than that. What if most buttons will be the same width or height? We shouldn't have to keep specifying the width and height parameters for every button, we should only have to specify them when necessary. We can have the constructor check if the property is actually defined in the config object, and set a default value if not. Like so:

```
var Button = function(config) {
    this.x = config.x || 0;
    this.y = config.y || 0;
    this.width = config.width || 150;
    this.height = config.height || 50;
    this.label = config.label || "Click";
};
```

Now we can just call it with a subset of the properties, because the other ones will be set to the default value:

```
var btn1 = new Button({x: 100, y: 100, label: "Please click!"});
```

All that work for a constructor, ay? But, it'll be worth it, I swear.

Now that we have the constructor squared away (buttoned away?), let's define a bit of behavior: the `draw` method. It'll be the same code as the `drawButton` function, but it will grab all the properties from `this`, since it's defined on the object prototype itself:

```
Button.prototype.draw = function() {
    fill(0, 234, 255);
    rect(this.x, this.y, this.width, this.height, 5);
    fill(0, 0, 0);
    textSize(19);
    textAlign(LEFT, TOP);
    text(this.label, this.x+10, this.y+this.height/4);
};
```

Once that's defined, we can call it like so:

```
btn1.draw();
```

Here's a program that uses that `Button` object to create 2 buttons - notice how easy it is to create and draw multiple buttons:

```
var Button = function(config) {

  this.x = config.x || 0;

  this.y = config.y || 0;

  this.width = config.width || 150;

  this.height = config.height || 50;

  this.label = config.label || "Click";

};
```

```
Button.prototype.draw = function() {
   fill(0, 234, 255);
   rect(this.x, this.y, this.width, this.height, 5);
   fill(0, 0, 0);
   textSize(19);
   textAlign(LEFT, TOP);
   text(this.label, this.x+10, this.y+this.height/4);
};


var btn1 = new Button({
   x: 100,
   y: 100,
   label: "Please click!"
});
btn1.draw();



var btn2 = new Button({
   x: 100,
   y: 213,
   label: "No! Click ME!"
});
btn2.draw();
```

We skipped the hard part, however: handling clicks. We can start by defining a function on the `Button` prototype that will report true if the user clicked inside a particular button's bounding box. Once again, this is just like our function from before, but it grabs all the properties from `this` instead of a passed in object:

```
Button.prototype.isMouseInside = function() {
    return mouseX > this.x &&
           mouseX < (this.x + this.width) &&
           mouseY > this.y &&
           mouseY < (this.y + this.height);
};
```

Now we can use that from inside a `mouseClicked` function:

```
mouseClicked = function() {
    if (btn1.isMouseInside()) {
        println("You made the right choice!");
    } else if (btn2.isMouseInside()) {
        println("Yay, you picked me!");
    }
};
```

Try it out below, clicking each of the buttons:

```
var Button = function(config) {
  this.x = config.x || 0;
  this.y = config.y || 0;
  this.width = config.width || 150;
  this.height = config.height || 50;
  this.label = config.label || "Click";
};

Button.prototype.draw = function() {
  fill(0, 234, 255);
  rect(this.x, this.y, this.width, this.height, 5);
  fill(0, 0, 0);
  textSize(19);
  textAlign(LEFT, TOP);
  text(this.label, this.x+10, this.y+this.height/4);
};

Button.prototype.isMouseInside = function() {
  return mouseX > this.x &&
      mouseX < (this.x + this.width) &&
      mouseY > this.y &&
      mouseY < (this.y + this.height);
};
```

```
var btn1 = new Button({
   x: 100,
   y: 100,
   label: "Please click!"
});
btn1.draw();



var btn2 = new Button({
   x: 100,
   y: 213,
   label: "No! Click ME!"
});
btn2.draw();



mouseClicked = function() {
   if (btn1.isMouseInside()) {
      println("You made the right choice!");
   } else if (btn2.isMouseInside()) {
      println("Yay, you picked me!");
   }
};
```

But, there's something that irks me about the way we've set up that click handling. The whole point of object oriented programming is to bundle up all the behavior related to an object inside the object, and to use properties to customize behavior. But, we've left some of the behavior dangling outside the object, the `println`s inside `mouseClicked`:

```
mouseClicked = function() {
    if (btn1.isMouseInside()) {
        println("You made the right choice!");
    } else if (btn2.isMouseInside()) {
        println("Yay, you picked me!");
```

```
    }
};
```

Those print statements should be better tied to each button somehow, like as something that we pass into the constructor. Just looking at it the way it is now, we could decide to pass a message into the constructor config, and define a `handleMouseClick` function to print it out:

```
var Button = function(config) {
    ...
    this.message = config.message || "Clicked!";
};

Button.prototype.handleMouseClick = function() {
    if (this.isMouseInside()) {
        println(this.message);
    }
};

var btn1 = new Button({
    x: 100,
    y: 100,
    label: "Please click!",
    message: "You made the right choice!"
});

mouseClicked = function() {
   btn1.handleMouseClick();
};
```

That's much nicer, since now everything associated with the each button's particular behavior is wrapped up in the constructor. But it's also overly simple. What if we wanted to do something besides printing a message, like draw a few shapes or change scenes, something that would take a few lines of code? In that case, we'd want to provide the constructor with more than just a string-- we actually want to provide it with a bunch of code. How can we pass around a bunch of code?

...With a function! In JavaScript (but not all languages), we can pass functions as parameters to functions. That's useful in many situations, but particularly useful when defining behavior for UI controls like buttons. We can tell the button, "hey, here's this function, it's a bunch of code that I want you to call when the user clicks the button." We refer to those functions as ***"callback"*** functions because they won't be called immediately, they'll be "called back" at an appropriate time later.

We can start by passing an `onClick` parameter that's a function:

```
var btn1 = new Button({
    x: 100,
    y: 100,
    label: "Please click!",
    onClick: function() {
       text("You made the right choice!", 100, 300);
    }
});
```

We then have to make sure our constructor sets an `onClick` property according to what's passed in. For the default, in case there's no `onClick` passed in, we'll just create a "no-op" function -- a function that does "no operations" at all. It's just there so that we can call it and not experience an error:

```
var Button = function(config) {
    // ...
    this.onClick = config.onClick || function() {};
};
```

Finally, we need to actually call back the callback function once the user clicks the button. That's actually pretty simple- we can just call it by writing the property name we saved it into and following that with empty parantheses:

```
Button.prototype.handleMouseClick = function() {
    if (this.isMouseInside()) {
        this.onClick();
    }
};
```

And now we're done - we have a `Button` object that we can easily create new buttons out of, making each button look different and responding differently to click events. Click around on the example below, and see what happens when you change button parameters:

```
var Button = function(config) {
   this.x = config.x || 0;
   this.y = config.y || 0;
   this.width = config.width || 150;
   this.height = config.height || 50;
   this.label = config.label || "Click";
   this.onClick = config.onClick || function() {};
};


Button.prototype.draw = function() {
   fill(0, 234, 255);
   rect(this.x, this.y, this.width, this.height, 5);
   fill(0, 0, 0);
   textSize(19);
   textAlign(LEFT, TOP);
   text(this.label, this.x+10, this.y+this.height/4);
};


Button.prototype.isMouseInside = function() {
   return mouseX > this.x &&
        mouseX < (this.x + this.width) &&
        mouseY > this.y &&
        mouseY < (this.y + this.height);
```

```
};
Button.prototype.handleMouseClick = function() {
    if (this.isMouseInside()) {
        this.onClick();
    }
};

var btn1 = new Button({
    x: 100,
    y: 100,
    label: "Please click!",
    onClick: function() {
        text("You made the right choice!", this.x, this.y+this.height);
    }
});
btn1.draw();


var btn2 = new Button({
    x: 100,
    y: 213,
    label: "No! Click ME!",
    onClick: function() {
        text("Yay, you picked me!", this.x, this.y+this.height);
    }
});
btn2.draw();

mouseClicked = function() {
    btn1.handleMouseClick();
```

```
    btn2.handleMouseClick();

};
```

Now that you have that as a template, you could customize your buttons in other ways, like different colors, or have them respond to other events, like mouseover. Try it out in your programs!