

08 - Functions

JavaScript Functions

WE'LL COVER THE FOLLOWING



- Creating Functions
- Revisiting Setup & Draw Functions
- Summary
- Practice

Creating Functions

Functions are primary building blocks of JavaScript. They allow us to write programs in a more efficient and scalable manner. Functions help us to manage complexity by containing and grouping operations under a single executable name. We already know how to call functions by using the p5.js predefined functions such as **ellipse** or **background**. We even created our own functions as p5.js forces us to put our code into two function declarations, **setup** and **draw**. If we wanted to create our own functions, we would be following the same convention that we have been using for the creation, or declaration, of these functions.

To create (or to declare) a new function, we would start off by using the function keyword and then give the function a name of our choosing that ideally describes the behaviour or purpose of the function.

```
function functionName() {  
    // function body  
}
```

Next, to the function name we would open brackets. If we want to build a function that works with user input, we can define parameters inside the brackets that would act as placeholder variable names for the future user

brackets that would act as placeholder variable names for the future user input. We will see how this works in a bit.

Then we have curly braces. Inside curly braces can be referred to as the function body. In there, we would write the code that would construct the logic of the function. We can also make use of the parameters, the variable names we defined inside the brackets next to the function name, as part of the operations we want to perform inside the function body.

Let's look at a simple example. Notice how p5.js has an **ellipse** function but not a **circle** function. This is not really an issue since we can easily create a circle by providing the ellipse function with same width and height values. For argument's sake, though, let's create a **circle** function that would work with three values: the x and y position that we want to draw our circle at and the diameter of the circle.

Here is how to do it. Inside the brackets, we will be writing down variable names that would eventually be provided when this function is called. These names are called *parameters* as they parameterize the functionality of the operation we are creating. We will be using these parameters inside our function to be able to let the user control the inner workings of the function.

```
function circle(x, y, diameter) {  
  ellipse(x, y, diameter, diameter);  
}
```

We can choose anything as parameter names, but it usually makes sense to use names that communicate the intent clearly. So in our case, using the names; x, y, and radius makes sense.

After defining this function, we can call it by using its name and providing it with values. Values provided to the function are called *arguments* to the function. Notice that the function might fail or not work as expected if all the required arguments are not provided.

```
circle(width/2, height/2, 100);
```

Don't worry too much if you feel like the terminology is confusing. It might take some time to get used to it. The *parameters* of a function can be thought as the values that the user would eventually provide when they are using the function. These same values that are provided when calling the function are

function. Those same values that are provided when calling the function are referred to as *arguments*.

With the **circle** function, we don't need to worry about using the **ellipse** function to draw circles anymore. We can just use our own function to draw those perfectly round circles. Having implemented the **circle** function ourselves, we know that it actually uses the ellipse function under the hood to draw those circles. But the neat thing about functions is that we don't really need to know how they work once they are available to us. We can just use them without thinking how they are implemented. **ellipse** function that is implemented by the smart people that created p5.js might be using all sorts of things inside to draw an ellipse, but as far as we are concerned it draws an ellipse when it is called, and that's all that matters.

In this example, creating a circle function doesn't buy us too much efficiency. As a matter of fact, we can just pass three arguments to the **ellipse** function instead of four to draw a circle instead. But functions become really important to use when we are building more complex programs as they help us to manage the complexity by containing and grouping operations under a single executable name. Functions are essentially blackboxes. They encapsulate the code that they contain and whatever variables that are declared using the **var** keyword inside the function are not visible from outside the function.

Output
JavaScript
HTML

Console

Clear

message is not defined

Hello World!

The **console.log** function on line 15 will throw an error because the variable **message** is only visible from inside the function **sayHello**.

Functions can work with no input, a single input or multiple inputs and they either *return* a result or don't. Let me explain what I mean by returning a value.

Let's say we want to create a function that multiplies a given numeric value by itself, essentially calculating the square of the given number. Here is one function that does that, it receives a number as a parameter and creates text that displays that number on the screen. So it is somewhat useful since we can use this function to display to the screen and learn what might be the square of an absurdly large number.

Output

JavaScript

HTML

100



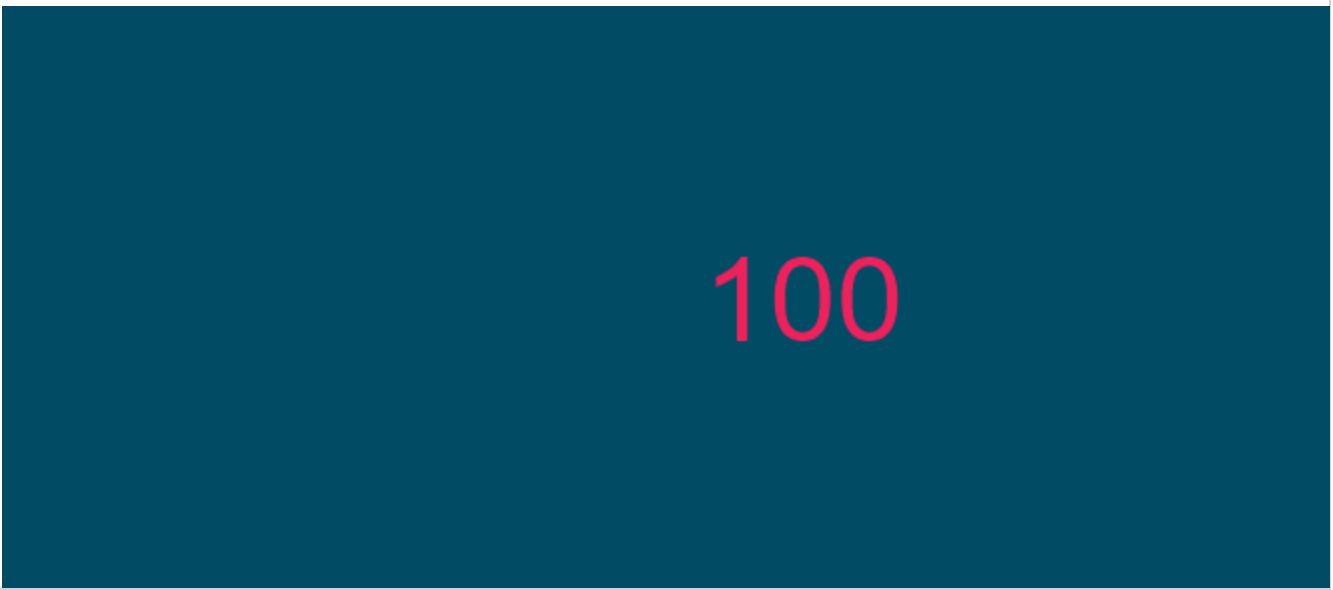
But if we wanted to use this resulting number in another calculation, we might hit a roadblock. This function is not *returning* the number to us; it is just displaying it on the screen. Calling this function affects the environment that we are in, but doesn't *return* a value for it to be used in further calculations. Some of the functions we have seen so far, like **ellipse**, **rect**, etc..., behaved in a similar fashion where they do something but don't actually return a value as a result of that calculation. Whereas, the **random** function when executed doesn't display anything on the screen but returns a value which we can capture in a variable.




To be able to return values from a function, we can use the **return** keyword. Let's alter the **squared** function to both, display the results on the screen and also to return a value.

Output

JavaScript

HTML





Console

Clear

100

100

100

100

100

Now, this function returns a value which we are using in a **console.log** function. Whenever the program comes across the keyword **return**, the program terminates the execution of the function and returns the value that is declared next to it to the caller of the function. This means that if we had any other lines below the **return** keyword, they wouldn't get executed since **return** terminates the execution of the current function. The **return** keyword is only available inside functions, trying to use it from outside a function will result in an error. As outside of a function, there is nothing to be returned.

Revisiting Setup & Draw Functions

Now that we learned about creating our functions, it is important to emphasize the difference in between creating a function vs. calling a function one more time. Notice how when we created our functions, we had to call them in order for them to be executed. For example, in this code example, we are only creating, or declaring a function:

```
function myFunction() {  
}
```

To be able to make use of this function, we need to execute it, by calling it with its name and using parentheses next to that name.

```
myFunction();
```

Notice one thing that is slightly strange when working in p5.js. We never really call the **setup** and **draw** functions and yet they get executed anyway! This is due to how p5.js is architected. p5.js handles the execution of the **setup** and **draw** functions for us as their execution follows some simple rules which are:

- **setup** function gets executed before the **draw** function.
- **setup** function is only executed one time, whereas **draw** function is

- **setup** function is only executed one time, whereas **draw** function is executed continuously.

Summary

We have been making use of functions from the moment we started to use p5.js. Its very own architecture depends on the existence of two functions inside our programs that has to have the name: **setup** and **draw**. Moreover, we have been using functions that are available to use that comes with the p5.js library such as **ellipse**, **rect**, etc.

We have seen that functions can be built to work with external user input or not. We can also build functions that either returns a value using the **return** keyword or not.

Functions are a way to create modular blocks of code that can be reused throughout our code to make our programs more maintainable and scalable by decreasing the amount of code we need to write. Whenever we find ourselves repeating a block code in multiple places, that's likely a good candidate to create a function from.

Practice

Create a function called **grid** that would work with three parameters; a **numX** and a **numY** parameter that would create **numX** amount of shapes (say rectangles) on the x-axis and **numY** amount of shapes on the y-axis and a **size** parameter that would set the size of the shapes.

For example:

```
grid(10, 30, 20); // Would create 10 x 30 rectangles of size 20px.
```

Output
JavaScript
CSS (SCSS)

