

## Set 21: Writing Functions

**Skill 21.01: Explain the purpose of a function**

**Skill 21.02: Declare a function**

**Skill 21.03: Call a function**

**Skill 21.04: Pass a parameter to a function**

**Skill 21.05: Use a *return* statement in a function**

**Skill 21.01: Explain the purpose of a function**

### Skill 21.01 Concepts

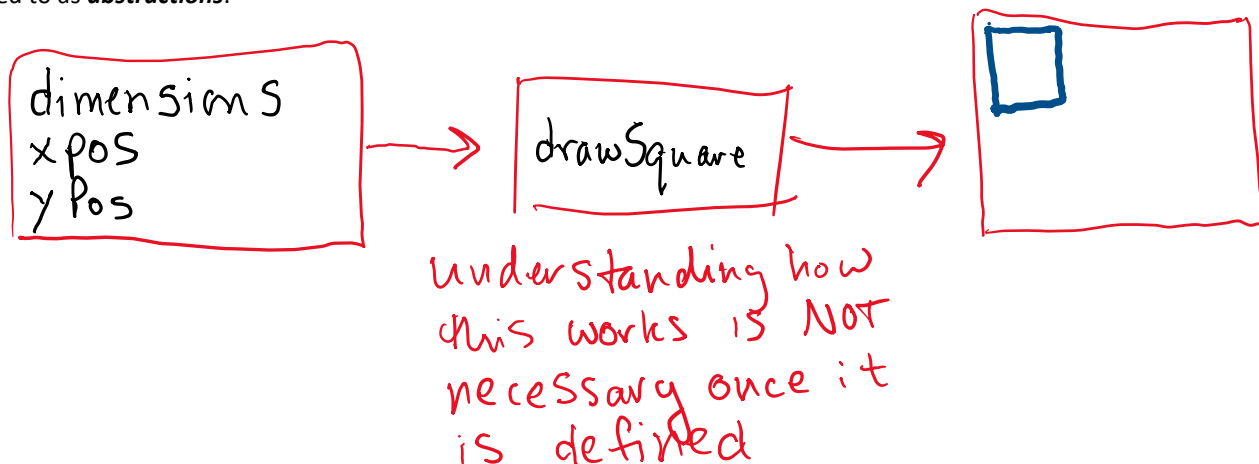
So far all the JavaScript we have been writing has been a series of commands. Consider the following code for drawing a box on the screen,

```
var square = document.createElement("div");
var dimensions = 200;
square.style.width = dimensions + "px";
square.style.height = dimensions + "px";
square.style.border = "thick solid #0000FF";
square.style.position = "absolute";
document.body.append(square);
```

That is a lot of code for such a simple task! For a program that required many boxes you would have lots of *repeated* code.

In programming, we often use code to perform a specific task multiple times. Instead of rewriting the same code, we can group a block of code together and associate it with one task, then we can reuse that block of code whenever we need to perform the task again. We achieve this by creating a **function**. A function is a reusable block of code that groups together a sequence of statements to perform a specific task.

It is possible to implement a function without understanding the underlying complexity. For example, once the *drawSquare* function is defined, it should be possible to draw many squares without understanding how it takes place. This process is illustrated below. Moreover, because functions help us manage the complexity of our code they are also referred to as **abstractions**.



### [Skill 21.01 Exercise 1](#)

#### Skill 21.02: Declare a function

##### Skill 21.02 Concepts

In JavaScript, there are many ways to create a function. One way to create a function is by using a function declaration. Just like how a variable declaration binds a value to a variable name, a function declaration binds a function to a name, or an identifier. Take a look at the anatomy of a function declaration below:

```
function greetWorld(){  
    console.log("Hello World!");  
}
```

Each portion of the above function is described below,

- **function** – the keyword used to create the function.
- **GreetWorld()** - the name of the function, or its identifier, followed by parentheses.
- **console.log("Hello World!");** - the function body, or the block of statements required to perform a specific task, enclosed in the function's curly brackets, { }.

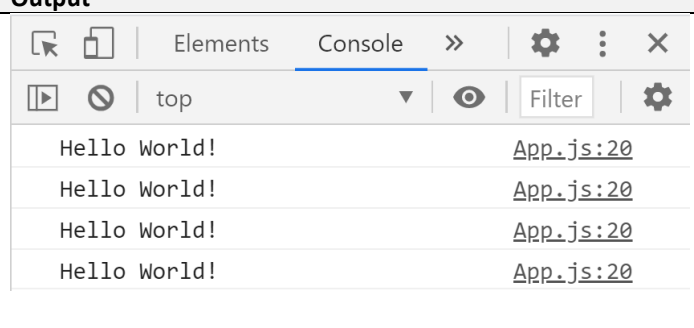
### [Skill 21.02 Exercise 1](#)

#### Skill 21.03: Call a function

##### Skill 21.03 Concepts

As we saw above, a function declaration binds a function identifier to a block of code.

However, a function declaration does not ask the code inside the function body to run, it just declares the existence of the function. The code inside a function body runs, or executes, only when the function is called. To call a function in your code, you type the function name followed by parentheses.

Code	Output
<pre>greetWorld(); greetWorld(); greetWorld(); greetWorld();  function greetWorld(){     console.log("Hello World!"); }</pre> <p><i>calls greetWorld 4 times</i></p>	 <p>The screenshot shows a browser's developer console with the 'Console' tab selected. It displays four log entries, each with the text 'Hello World!' and the source 'App.js:20'. The console interface includes standard controls like a search bar, a filter button, and icons for clearing, saving, and zooming.</p>

In the example above, the function call, `greetWorld()`, executes the function body, or all of the statements between the curly braces.

Notice in the above example, that once a function is defined, we can call it as many times as needed.

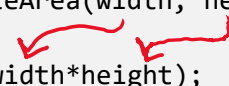
### [Skill 21.03 Exercises 1](#)

#### Skill 21.04 Pass a parameter to a function

##### Skill 21.04 Concepts

So far, the functions we've created execute a task without an input. However, some functions can take inputs and use the inputs to perform a task. When declaring a function, we can specify its parameters. Parameters allow functions to accept input(s) and perform a task using the input(s). We use parameters as placeholders for information that will be passed to the function when it is called.

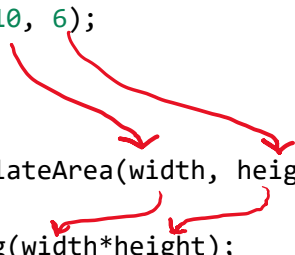
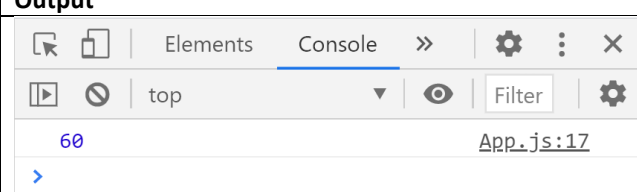
```
function calculateArea(width, height){  
    console.log(width*height);  
}
```

A diagram with two red arrows. One arrow starts from the word 'width' in the function call 'calculateArea(width, height)' and points to the parameter 'width' in the function definition 'function calculateArea(width, height)'. The second arrow starts from the word 'height' in the function call and points to the parameter 'height' in the function definition.

In the example above, *calculateArea*, computes the area of a rectangle, based on two inputs, *width* and *height*. The parameters are specified between the parenthesis as *width* and *height*, and inside the function body, they act just like regular variables.

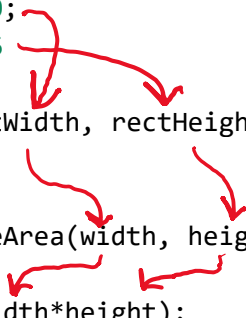
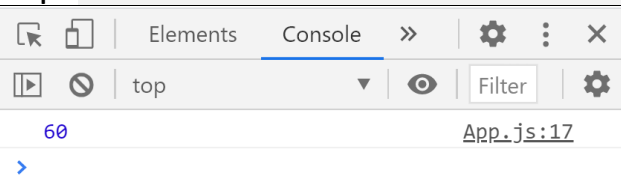
When calling a function that has parameters, we specify the values in the parentheses that follow the function name. The values that are passed to the function when it is called are called arguments. Arguments can be passed to the function as values or variables.

Now, let's consider the flow of data as we call the *calculateArea* function above,

Code	Output
<pre>calculateArea(10, 6);  function calculateArea(width, height){     console.log(width*height); }</pre>  A diagram with two red arrows. One arrow starts from the number '10' in the function call 'calculateArea(10, 6)' and points to the parameter 'width' in the function definition 'function calculateArea(width, height)'. The second arrow starts from the number '6' in the function call and points to the parameter 'height' in the function definition.	 A screenshot of a web browser's developer console. The 'Console' tab is selected. It shows a single log entry with the value '60' in blue text. To the right of the value, it says 'App.js:17'. Above the console, there are tabs for 'Elements' and 'Console', and a search bar with the text 'top'.


In the function call above, the number 10 is passed as the width and 6 is passed as height. Notice that the order in which arguments are passed and assigned follows the order that the parameters are declared.

Now let's look at another example,

Code	Output
<pre> var rectWidth = 10; var rectHeight = 6;  calculateArea(rectWidth, rectHeight);  function calculateArea(width, height){     console.log(width*height); } </pre> 	

In the above example, the variables *rectWidth* and *rectHeight* are initialized with the values for the *height* and *width* of a rectangle before being used in the function call.

By using parameters, *calculateArea* can be reused to compute the area of any rectangle!

Code	Output
<pre> var rectWidth = 10; var rectHeight = 6;  calculateArea(rectWidth, rectHeight); calculateArea(rectWidth*2, rectHeight*2); calculateArea(rectWidth/2, rectHeight/2);  function calculateArea(width, height){     console.log(width*height); } </pre>	

#### [Skill 21.04 Exercises 1](#)

#### **Skill 21.05 Use a return statement in a function**

##### **Skill 21.05 Concepts**

When a function is called, the computer will run through the function's code and evaluate the result of calling the function. By default that resulting value is undefined.

Code	Output
<pre>function rectangleArea(width, height){     var area = width * height; }  console.log(rectangleArea(5, 7));</pre>	<pre>undefined                                     App.js:6 &gt;  </pre>

In the code example, we defined our function to calculate the area of a width and height parameter. Then `rectangleArea` was invoked with the arguments 5 and 7. But when we went to print the results we got *undefined*. Did we write our function wrong? No! In fact, the function worked fine, and the computer did calculate the area as 35, but we didn't capture it. So how can we do that? With the keyword `return`!

Code	Output
<pre>function rectangleArea(width, height){     var area = width * height;     return area; }  console.log(rectangleArea(5, 7));</pre>	<pre>35   App.js:7 &gt;</pre>

To store information from a function call, we use a ***return*** statement. To create a return statement, we use the ***return*** keyword followed by the value that we wish to return. Like we saw above, if the value is omitted, *undefined* is returned instead.

When a *return* statement is used in a function body, the execution of the function is stopped when the *return* statement is reached and the function is exited. Any code following a *return* statement will not be executed.

The *return* keyword is powerful because it allows functions to produce an output. We can then save the output to a variable for later use. Consider the example below,

Code	Output
<pre>var myArea = rectangleArea(10, 5); console.log(myArea);  function rectangleArea(width, height){     var area = width * height;     return area; }</pre>	<pre>50   App.js:4 &gt;</pre>

#### [Skill 21.05 Exercises 1](#)