Chapter 10

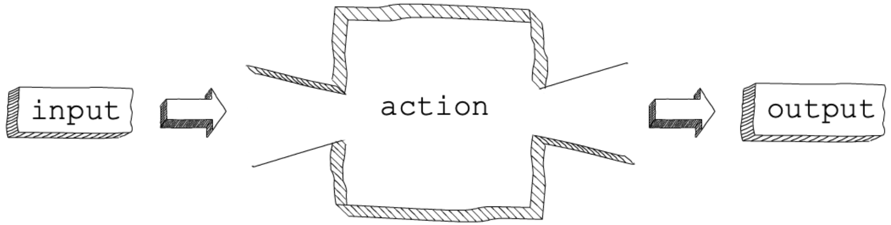
Functions at your beck and call

Chapter Breakdown

10.1 INTRODUCTION

FUNCTION ARE

- REUSABLE
- CALLABLE
- VERSATILE



They receive input and take an action to produce an output!

FUNCTION YOU ALREADY KNOW:

- CONSOLE.LOG
- TYPE CONVERSION
  - Number
  - String
  - Boolean
- String and Array Methods

10.2 USING FUNCTIONS!

Function Call

- Synonymous with function invocation or ‘invoking a function’
- Functions can be called with or without arguments
- Every function provides a return value
    - **NOTE!** If it doesn't provide an explicit return value undefined will be returned

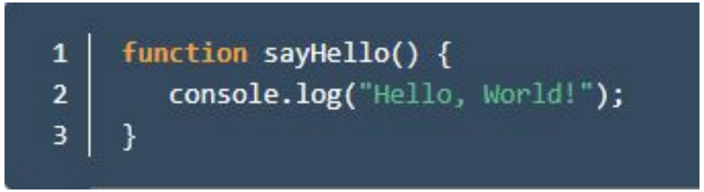
10.3 CREATING FUNCTIONS

FUNCTION SYNTAX

```
1               2                  3
function myFunction (parameter1,.....,parameterN) {
   //function body4
}
```

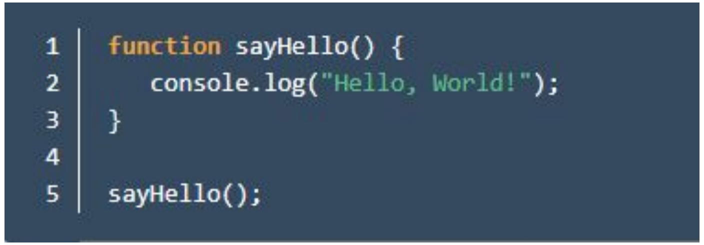
1. function is a keyword that instructs JS to create a new function using the definition that follows.
2. function name is determined by the programmer
3. function parameters are variables that can be used only with the function itself
   a. Unlike other languages Javascript does not allow you to specify types of parameters
4. function body is everything within the curly brackets

Defining a function makes it available to be used. It doesn't execute when it is defined!



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

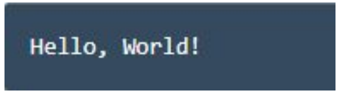
Function is defined but not called.



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

sayHello();
```

Function is defined and called. Console output is:

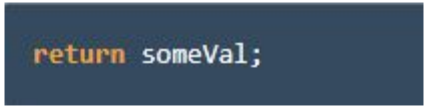


```
Hello, World!
```

10.4 FUNCTION INPUT AND OUTPUT

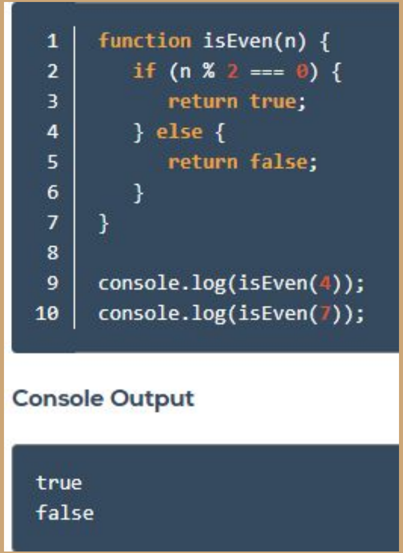
Return Statements

To return a value from a function that we create, we must use a return statement



```
return someVal;
```

- Return statements are options
- Return statements terminate function execution
    - Safe bet most of the time is to put your return statement as the last part of your function
- Functions can have more than 1 return statement. But only one of them will return a value.
    - Ex:



```
function isEven(n) {
    if (n % 2 === 0) {
        return true;
    } else {
        return false;
    }
}

console.log(isEven(4));
console.log(isEven(7));
```

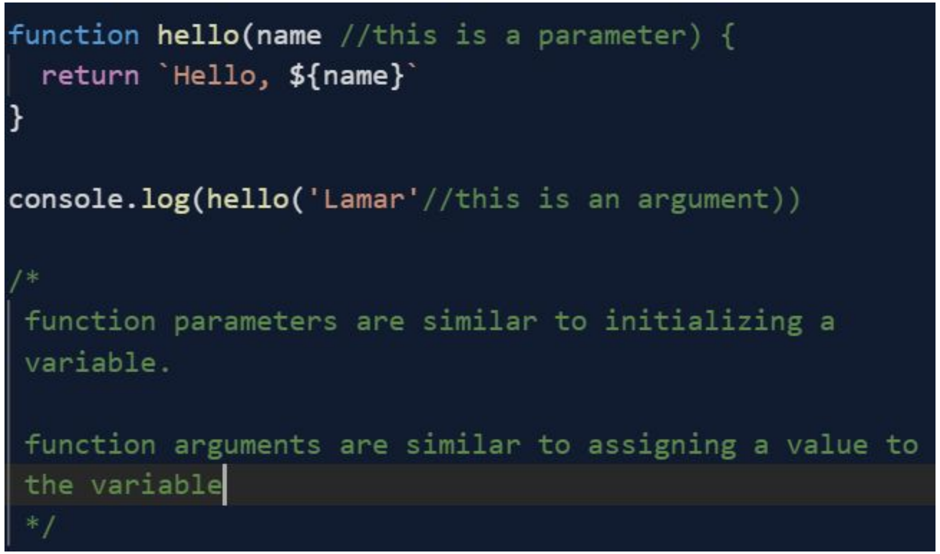
Console Output

```
true
false
```

FUNCTION(PARAMETER VS ARGUMENT)

A Parameter is part of the function definition, and behaves like a variable that only exists in your function.

An Argument is used when we invoke the function. It is a specific value that is used during the function call.



```
function hello(name //this is a parameter) {
  return `Hello, ${name}`
}

console.log(hello('Lamar'//this is an argument))

/*
 function parameters are similar to initializing a
 variable.

 function arguments are similar to assigning a value to
 the variable
 */
```

10.5 GOOD FUNCTION WRITING PROCESS

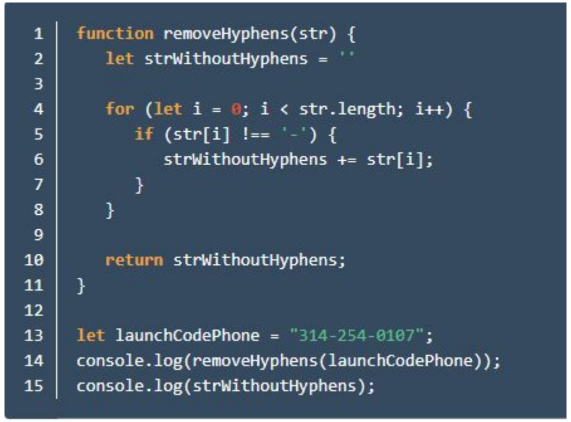
A Good Function-Writing Process

1. Design your function
    a. What data (that is, parameters) does my function need to do its job?
    b. Should my function return a value? (Hint: The answer is almost always "yes.")
    c. What should be the data type of my function's return value?
    d. What is a good, descriptive name for my function?
    e. What data types do we expect the parameters to be?
    f. What are good names for my parameters?
2. Create the base structure
3. Write the Body

10.6 PARAMETERS AND VARIABLES

Function Scope

- The extent to which a variable is visible within a program.
- Functions are our first example of limited variable scope
  - Ex: a variable defined using let inside a function is not visible outside of that function



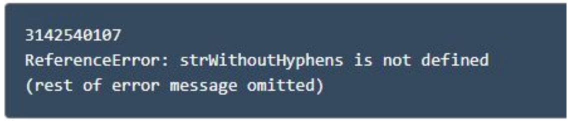
```javascript
function removeHyphens(str) {
   let strWithoutHyphens = ''

   for (let i = 0; i < str.length; i++) {
      if (str[i] !== '-') {
         strWithoutHyphens += str[i];
      }
   }

   return strWithoutHyphens;
}

let launchCodePhone = "314-254-0107";
console.log(removeHyphens(launchCodePhone));
console.log(strWithoutHyphens);
```

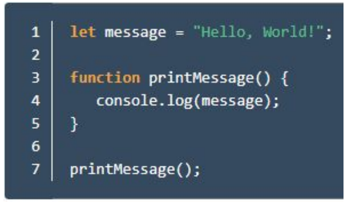
Console Output



```
3142540107
ReferenceError: strWithoutHyphens is not defined
(rest of error message omitted)
```

Variable Shadowing

- In some cases a variable defined outside a function may be visible within the function
  - This is NOT the case for all variables and we will explore in depth in a later chapter
- Shadowing occurs when you have a variable outside of a function that is the same name as the variable inside the function.

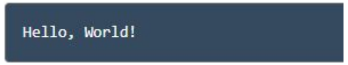


```javascript
let message = "Hello, World!";

function printMessage() {
   console.log(message);
}

printMessage();
```

Console Output



```
Hello, World!
```

10.7 NAMING FUNCTIONS

NAMING FUNCTIONS

1. Use Camel Case
2. Use Verb/Noun Pairs when Applicable
3. Use Descriptive Names

EXAMPLES

1. camelCase
   a. astronautCount
   b. fuelTempCelsius
2. Use Verb/Noun Pairs
   a. fillUpGasTank
   b. eatFood
3. Use Descriptive Names
   a. convertKilometersToMiles
   b. convertCelsiusToFahrenheit

10.8 COMPOSING FUNCTIONS

FUNCTION COMPOSITION

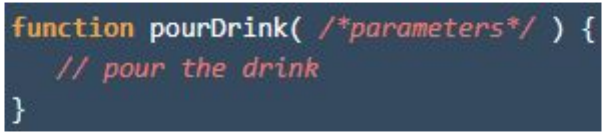
USING FUNCTIONS TO BUILD OTHER FUNCTIONS

- Functions should do exactly one thing
    - Easier to debug
    - Easier to read
    - More reusable

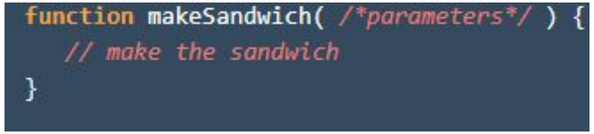
FUNCTION COMPOSITION RECIPE

1. Ask yourself if the function you want to write can be broken down into smaller functions or tasks
   a. makeLunch → makeSandwich and pourDrink

2. Write your task functions

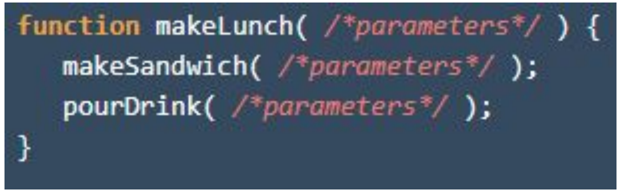


```
function pourDrink( /*parameters*/ ) {
   // pour the drink
}
```



```
function makeSandwich( /*parameters*/ ) {
   // make the sandwich
}
```

3. Write your main function utilizing your task functions



```
function makeLunch( /*parameters*/ ) {
   makeSandwich( /*parameters*/ );
   pourDrink( /*parameters*/ );
}
```

4. Enjoy readable code that's easier to debug!

10.9 WHY CREATE FUNCTIONS

REASONS TO CREATE FUNCTIONS!

1. Reduces repetition and DRY's out your code
2. Makes your code more readable
3. Reduce complexity
4. Enable Code Sharing

TO BE CONTINUED...