

# Functions Cheat Sheet

---

Here are some notes on what's been covered in this unit. Feel free to copy this and extend it to make your own cheat sheet.

## Defining and Calling JavaScript Functions

---

- In JavaScript, a **function** can be:
  - Made up of either a single reusable statement or a group of reusable statements.
  - Called from anywhere in the program, which allows for the statements inside a function to not be written over and over again.
- To define a function, use the following syntax:

```
var myFunctionName = function() {  
    // Body of the function  
};
```

- To use, or **call**, a function, simply type the name of your function, followed by `()` (plus any inputs you might be passing in).

```
myFunctionName();
```

## Parameters and Arguments

---

- **Parameters** are the variables that are defined in the function's declaration when the function is defined. They allow us to provide some extra information for a function.
  - In the following function example, `firstName`, `lastName`, `year` and `city` in the parentheses that follow the word `function` are the parameters:

```
var greetUser = function (firstName, lastName, year, city) {  
    console.log("Hello " + firstName + " " + lastName + " born in " + year + " from " +  
    city + "!" );  
};
```

- **Arguments** are the actual values passed to the function when the function is called.
  - In the following example, `"Bruce"`, `"Wayne"`, `1939` and `"Gotham"` are the arguments:

```
greetUser("Bruce", "Wayne", 1939, "Gotham");
```

## Return Statements

---

- **Return statements** allow us to "spit out" or "hand back" a value to the line where we called the function.
  - We can then store that value that we returned in a variable, or work with it:

```
var addBonusPoints = function (score) {  
  if (score > 50) {  
    return score + (score * .10);  
    // if score is 55, then 60.5 will be returned  
  }  
  
  return score;  
};  
  
var totalPoints = addBonusPoints(55);  
// => 60.5  
// The variable totalPoints will now hold 60.5
```

- Additionally, a return statement will cause the function that contains it to immediately end when that line is run. In the example below, we are simply using return to exit the function if muted is equal to true, instead of returning a value:

```
var rockAndRoll = function (muted) {  
  var song = "It's only Rock 'N' Roll";  
  var artist = "Rolling Stones";  
  
  if (muted === true) {  
    return;  
    // Here we use return as a way to exit a function, instead of returning any value  
  }  
  console.log("Now playing: " + song + " by " + artist);  
};  
  
rockAndRoll(true);
```

## Variable Scope

---

- When we declare variables inside a function, those variables will only be accessible from within that function. This is known as **scope**.

- If a variable is declared inside a function, it is local to that function and therefore referred to as a **local variable**.
  - This also means it has **local scope**. When we have a variable with local scope, it cannot be referenced outside of that function, which means it cannot be called or used outside of the brackets in which it's contained.
- When a variable is declared outside a function, it is referred to as a **global variable**. A global variable has **global scope**, which means all scripts and functions on a web page can access it.

# Defining and Calling Functions

1. In the "JavaScript" panel in JS Bin, define a function `recitePoem`.

- Inside the function, log "Roses are red, violets are blue." to the console.
- Call the `recitePoem` function in the "JavaScript" panel.

```
var recitePoem = function () {  
    console.log("Roses are red, violets are blue.");  
};  
  
recitePoem();
```

2. In the "JavaScript" panel in JS Bin, define a function `playSong`.

- Inside the function, log "Cheer up sleepy Jean, Oh, what can it mean." to the console.
- Call the `playSong` function.

```
var playSong = function () {  
    console.log("Cheer up sleepy Jean, Oh, what can it mean.");  
};  
  
playSong();
```

3. In the "JavaScript" panel in JS Bin, define a function `twoByFour`.

- Inside the function, log `2 * 4` to the console.
- Call the `twoByFour` function.

```
var twoByFour = function () {  
    console.log(2 * 4);  
};  
  
twoByFour();
```

## Parameters and Return Statements

1. In the "JavaScript" panel in JS Bin, define a function `sayHello`.

- It should accept one parameter, `name`.
- Inside the function, *return* a greeting in the following format: (i.e., "Hello, *name*").
- After hitting the "Run" button in the "Console" panel, call the `sayHello` function in the "Console" panel, using "Bill" as the argument.
- You should see "Hello, Bill" displayed in the console.

```
var sayHello = function (name) {
    return "Hello, " + name;
};

sayHello("Bill");
```

2. In the "JavaScript" panel in JS Bin, define a function areBothEven.

- It should accept two parameters, num1 and num2.
- Inside the function, *return* true if num1 and num2 are both even, but false if they are not.
- After hitting the "Run" button in the "Console" panel, call the areBothEven function in the "Console" panel, using 2 and 4 as the arguments.
- You should see true displayed in the console.

```
var areBothEven = function (num1, num2) {
    if (num1 % 2 === 0 && num2 % 2 === 0) {
        return true;
    } else {
        return false;
    }
};

areBothEven(2, 4);
```

3. In the "JavaScript" panel in JS Bin, define a function hotOrNot.

- It should accept one parameter, temp.
- Inside the function, *return* "Hot!" if temp is greater than or equal to 70, but "Not hot." if temp is less than 70.
- After hitting the "Run" button in the "Console" panel, call the hotOrNot function in the "Console" panel, using 76 as the argument.
- You should see "Hot!" displayed in the console.
- Test out the function using different numbers for the argument when calling the function to make sure everything is working.

```
var hotOrNot = function (temp) {
    if (temp > 70) {
        return "It's hot!";
    } else {
        return "It's not hot.";
    }
};

hotOrNot(76);
```

## Scope

1. We are getting a reference error when we try to log "Hello " and "name" to the console. See if you can move the `console.log` statement to where it can access the variable `name`.

```
var sayHello = function () {  
  var name = "Marie";  
  console.log("Hello " + name);  
};  
  
sayHello();
```

2. We are trying to keep track of the total amount of points that a player has scored, but each time the function is called, our score is being set to zero again. Which statement could you move outside the function so that the total is set to 0 when the page first loads, but every time the function is called the total is incremented by one?

```
var total = 0;  
  
var scorePoint = function () {  
  total += 1;  
  console.log(total);  
};  
  
scorePoint();  
scorePoint();  
scorePoint();
```