



# **FEWD - Functions**

**James Gallichio**

# Agenda

- Review
- Functions
- Anonymous Functions \* Callbacks
- Weather Application

# **Review**

# Functions

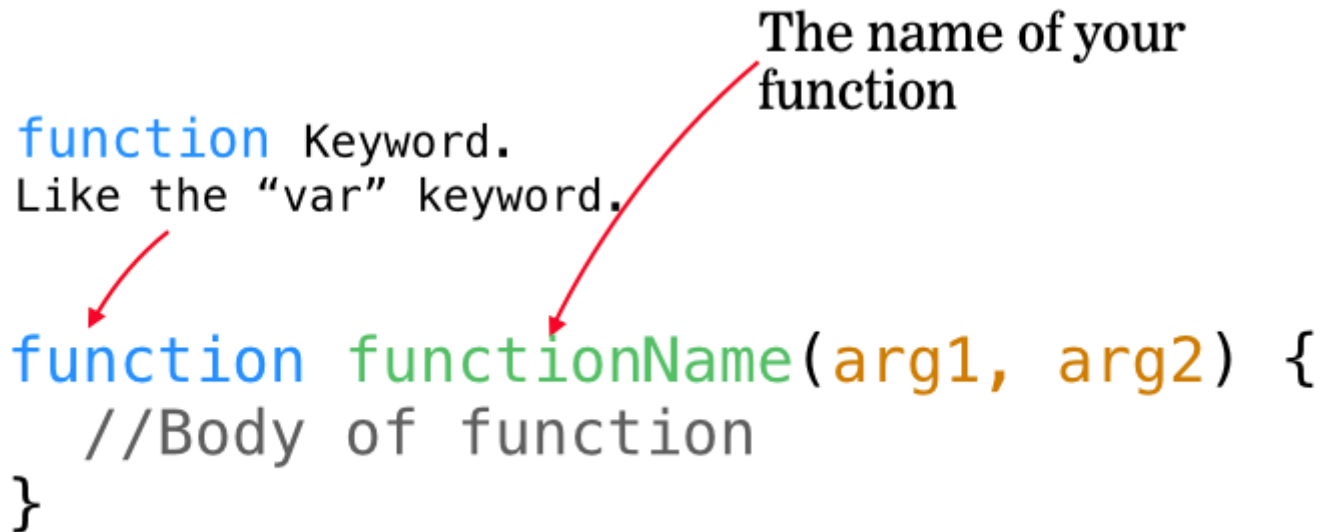
A set of instructions that together perform a specific task.

# Functions Syntax

`function` Keyword.  
Like the “var” keyword.

The name of your  
function


```
function functionName(arg1, arg2) {  
    //Body of function  
}
```

A diagram illustrating the syntax of a function declaration. It shows the code 'function functionName(arg1, arg2) {' with annotations. A red arrow points from the text 'function Keyword. Like the “var” keyword.' to the word 'function' in the code. Another red arrow points from the text 'The name of your function' to the word 'functionName' in the code. The code is color-coded: 'function' is blue, 'functionName' is green, and 'arg1, arg2' is orange. The closing brace '}' is on the same line as the opening brace '{'.

# Function Calls

```
function helloWorld() {  
  console.log("Hello Functions");  
}
```

```
helloWorld(); //Prints "Hello Functions to the  
console.
```

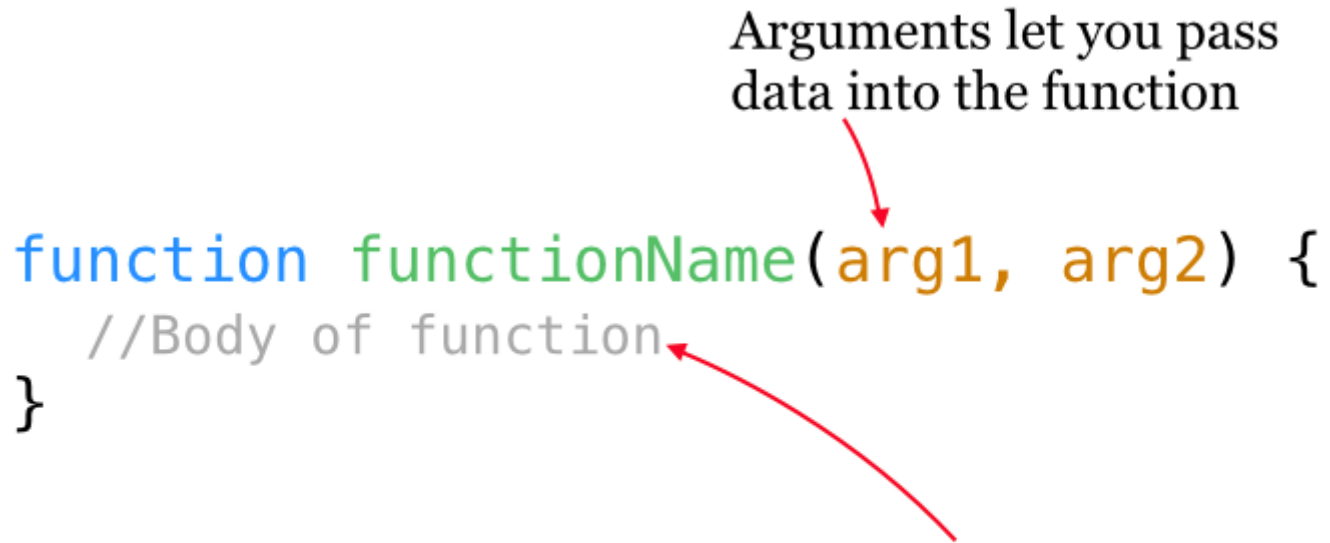


The brackets execute the function.  
Try calling the function without  
them to see what happens.

# Function Arguments

Arguments let you pass data into the function

```
function functionName(arg1, arg2) {  
    //Body of function  
}
```

A diagram illustrating the components of a function definition. A red arrow points from the text 'Arguments let you pass data into the function' to the arguments 'arg1, arg2' in the function signature. Another red arrow points from the text 'The functions executed code goes between the { } brackets. Much like an “if” statement.' to the body of the function, which is the code between the curly braces.

The functions executed code goes between the { } brackets. Much like an “if” statement.

# Function Arguments

```
function addAndPrint(num1, num2) {  
    var sum = num1 + num2;  
    console.log(sum);  
}
```

```
addAndPrint(1, 2); // Result is 3
```

```
addAndPrint(8, 2); // Result is 10
```



## Function return

We can tell functions to *return* a value back to the location it was called.

This allows us to continue using the results of a function after it has finished running.

# Function return

```
function getCircleCircumference(r) {  
  var pi = 3.142;  
  return 2 * pi * r;  
}  
  
var radius = 3;  
var circumference = getCircleCircumference(radius);
```



# Cash Register

<http://codepen.io/anon/pen/RpvLpB>

# **Anonymous Functions**

# Anonymous Functions

Normal function declaration:

```
$("#someButton").click(handleClickFunction);  
  
function handleClick() {  
  //Do some stuff  
}
```

Anonymous function:

```
$("#someButton").click(function() {  
  //Do some stuff  
});
```

**Where would we use an anonymous function?**

```
$(document).ready(function(){  
  // all our code goes in here  
});
```



# **Anonymous Cash Register**



# Function Callbacks

Some functions allow us to pass another function as an argument that will be executed once the original call has been completed.

# Function Callbacks

```
$("#description").slideDown("slow", function() {  
  
    $(".readMore").hide();  
});
```



**Temp Converter**

# Reading

- Arrays
- jQuery .each
- Recursive functions