Title: Note on Javascript Functions?

Answer:

What is a Function?

- A function is a ***subprogram*** designed to perform a particular task.
- Functions are executed when they are called. This is known as ***invoking*** a function.
- Values can be ***passed*** into functions and used within the function.
- Functions ***always*** `return` a value. In JavaScript, if no `return` value is specified, the function will return `undefined`.
- Functions are ***objects***.

Define a Function.

There are a few different ways to define a function in JavaScript:

A **Function Declaration** defines a named function. To create a function declaration you use the `function` keyword followed by the name of the function. When using function declarations, the function definition is hoisted, thus allowing the function to be used before it is defined.

```
function name(parameters){
  statements
}
```

A **Function Expressions** defines a named or anonymous function. An anonymous function is a function that has no name. Function Expressions are not hoisted, and therefore cannot be used before they are defined. In the example below, we are setting the anonymous function object equal to a variable.

```
let name = function(parameters){
  statements
}
```

An **Arrow Function Expression** is a shorter syntax for writing function expressions. Arrow functions do not create their own `this` value.

```
let name = (parameters) => {
  statements
}
```

Parameters vs. Arguments.

In JavaScript, you may have heard the terms *parameters* and *arguments* used interchangeably. While very similar, there is an important distinction to make between these two keywords.

Parameters are used when defining a function, they are the *names* created in the function definition. In fact, during a function definition, we can pass in up to 255 parameters! Parameters are separated by commas in the `()`. Here's an example with two parameters — `param1` & `param2`:

```
const param1 = true;
const param2 = false;
```

```
function twoParams(param1, param2){
  console.log(param1, param2);
}
```

Arguments, on the other hand, are the *values* the function receives from each parameter when the function is executed (invoked). In the above example, our two arguments are `true` & `false`.

Invoking a Function.

Functions execute when the function is called. This process is known as invocation. You can invoke a function by referencing the function name, followed by an open and closed parenthesis: `()`.

Lets explore an example.

> If you're using Google Chrome, open up your dev console so you can code along with these examples: **[WINDOWS]: Ctrl + Shift + J [MAC]: Cmd + Opt + J**

First, we'll define a function named `logIt`. This function will take one parameter, `name`. When executed, the function will log that `name` back to the console:

```
function logIt(name){
  console.log(name);
}
```

To invoke our function, we call it, while passing in the singular parameter. Here I am calling this function with the name *Joe*:

```
logIt('Joe');
// Joe
```

If your function has no parameters, you can invoke it with an empty set of parenthesis:

```
function logIt2(){
  console.log('The second one');
}
```

```
logIt2();
// The second one
```

Function Return.

Every function in JavaScript returns `undefined` unless otherwise specified.

Let's test this by creating and invoking an empty function:

```
function test(){};
```

```
test();
// undefined
```

Awesome, as expected, `undefined` is returned.

Now, we can customize what is returned in our function by using the `return` keyword followed by our return value. Take a look at the code below:

```
function test(){
  return true;
};
```

```
test();
// true
```

In this example we explicitly tell the function to return `true`. When we invoke the function, that's exactly what happens.

But why is this important?

It's important because the value that a function returns, is actually returned to the caller of the function. Take a look at this code:

```
let double = function(num) {
   return num * 2;
}
```

This is a function expression that creates a function that will return two times the value of our input parameter `num`. We can then invoke the function and save the return value to a variable:

```
let test = double(3);
```

When we log out our `test` value, we get `6`:

```
console.log(test);
// 6
```

Awesome! The `return` variable not only returns values from a function, but it assigns them to whatever called the function!

Another important rule of the return statement is that it stops function execution immediately.

Consider this example where we have two return statements in our test function:

```
function test(){
  return true;
  return false;
};
```

```
test();
// true
```

The first return statement immediately stops execution of our function and causes our function to return `true`. The code on line three: `return false;` is *never* executed.

Function Objects.

Functions are function objects. In JavaScript, anything that is not a primitive type (`undefined`, `null`, `boolean`, `number`, or `string`) is an object. Objects in JavaScript are *extremely* versatile. Because of this, we can even pass a function as a parameter into another function.

When a function accepts another function as a parameter, or returns a function, it is called a higher-order function. You've probably already used a bunch of higher order functions and don't even know it: `Array.prototype.map` and `Array.prototype.filter` are higher order functions (Just to name a couple). *You can check out some of my previous articles to learn more about objects and higher order functions in JavaScript…*

Key Takeaways.

This is a lot of information to digest. Here's a list of the important stuff:

- A function is a ***subprogram*** designed to perform a particular task.
- Function definitions are hoisted — expressions are not.
- Functions are executed when they are called. This is known as ***invoking*** a function.
- Values can be ***passed*** into functions and used within the function. The name of the value is called a parameter. The actual value itself is called an argument.
- Functions ***always*** `return` a value. In JavaScript, if no `return` value is specified, the function will return `undefined` by default.
- Functions are ***objects***.

Tags: functions / methods, javascript