**High-Order Functions**

1. **Functions as Data**

JS functions behave like any other data types in the language. When created, they are also objects. Therefore, we can:

+) Assign functions to variables (Function expressions)

+) Reassign them to new variables

e.g., We have a function with a very long variable name. We can call this function by assigning it to another variable:

const thisIsAVeryLongFunctionName = () => {  
    console.log("I’m doing very important work!");  
};

const short = thisIsAVeryLongFunctionName;  
short(); // This function call barely takes any space!

NOTE: We see that we do not include () while assigning. This is because we are assigning the function itself, not the value it returns.

+) Functions also have properties and methods:

**Properties**

.length 🡪 number of arguments expected by the funtion

.name 🡪 name of the function (original function passed to variable)

**Methods**

.toString() 🡪 Returns a string representing source code of the function.

|  |
| --- |
| .call()  .apply() 🡪 Related to Objects. Details in **Advanced Objects**  .bind() |

1. **Functions as Parameters**

Since functions can behave like any other type of data in JS, functions can accept other functions as parameters.

Functions that get passed in as parameters are called ***callback functions****.*

When we call a higher-order function, and pass another function in as argument, we do not call the callback function in the argument. Doing so would evaluate to passing in the return value (which is too obvious). \*\*With callback functions, we pass in the function itself by typing the callback function name, ***without ()***.

🡪 We do not just pass in the return value of the callback funtion. We pass in the whole function, that we can call within the higher order function.

const higherOrderFunc = param => {  
  param();  
  console.log(`I just invoked ${param.name} as a callback function!`);  
}  
  
const anotherFunc = () => {  
  console.log('I\'m being invoked by the higher-order function!');  
}  
  
higherOrderFunc(anotherFunc); // I’m being invoked by the higher-order function!

// I just invoked anotherFunc as a callback function!

The above is called Synchronous running, as the higher order function is ran line by line until the end, in order.

\*\*Another type is Asynchronous (used a lot when working with setTimeOut(), clickListener(), etc.) This is when we pass an anonymus callback function, meaning:

<higherOrderFunction> **(**<callback param> => {

//defining callback funtion

}**)**