Lab - 2 Working with Functions

Introduction

Functions are (at their simplest form) collections of statements that may be run as a group at a later time. These can be useful as they allow developers to reuse code and to break larger problems into simpler smaller ones.

As an example, from our previous exercise, we could create a function that could update the innerHTML of the first element that matches a selector. This can be done by declaring a function that accepts two parameters, which represent incoming values that the function needs to operate properly. These two parameters will represent the selector for the element to update and the new innerHTML value to use.

Task Writing A Function

Create an index.js file in the js folder and add the following javaScript function to your file. The function has two parameters, a selector and a string of text that you want to update.

```
function updateInnerHTML (selector, string){
   document.querySelector(selector).innerHTML = string
}
```

Function Signature

The first line in the example above is known as the function signature; it informs us of how to use the function. The reserved word function is used to indicate that we are declaring a new function. Following the reserved word we see updateInnerHTML, which is the identifier (often referred to as the name) of the function.

The name of the function is important as it is used to invoke or call the function at a later time. Following the name, we find the parameter list enclosed within parentheses. You can specify as many parameters as are necessary (including none) for the function to operate correctly. These parameters represent the actual passed in argument values when being properly called.

Function Body

Note that the body (enclosed within braces) is not executed at the time the browser sees the declaration. The body of the function is what's known as a block-statement, which is simply any group of statements that are enclosed within braces. This block of code is effectively executed as a whole. In this case, the body is only made up of a single statement, which simply attempts to assign a new value to an existing element's innerHTML property.

Calling The Function

To actually use the function, it must be called or invoked. This is simply done by referencing the name of the function followed by any required arguments (the parentheses are necessary, even if there are no arguments).

Task Calling The Function

In order to update the text of the h1 we need to call the function and send it two arguments the element to select and a string of text to add to the h1 element.

```
function updateInnerHTML (selector, string){
    document.querySelector(selector).innerHTML = string
  }

updateInnerHTML('h1', "New Topic")
```

The first argument ('h1') represents the selector for the element we wish to update, and the second argument ('Updated Text For H1') represents the new value for the innerHTML. These values are passed in order to the corresponding parameters (i.e. selector and newValue) in the function declaration. These actual values are then substituted in place of the parameters where they appear in the function body.

Returning a Value From A Function

Our first function is very simple and doesn't do a whole lot, which is fine in this case. Many useful functions are quite simple and easy to use. One of the most common reasons to declare a function is to have it perform some calculation or return some result. This can be done by including a return statement in your function body. The return statement is simply a statement that begins with the reserved word ***return*** followed by an expression or a value.

Let's explore functions that return a value by creating another simple task.

Task Creating a function with a return value

Create a function called strong. This function accepts a single parameter, a string that will be add to a template literal. Note that to add a parameter to the template literal we must first jump out of the template literal to the javascript using the ${} syntax. This allows us to add javascript between the curly braces.

Though the template literal looks like markup it is just text (string).

```
function strong(string){
    return `<strong>${string}</strong>`
}
```

Create a variable called template and call the strong function with any text you like. Open the dev tools and go to the console tab. You should see the return value from the function as a string with the text you sent as an argument from the function call. The return value may look like html to you but to JavaScript it isn't. To JavaScript the return value is a string. JavaScript has a few methods for adding html'ish text to the DOM. We will take a look at this in the next task.

```
function strong(string){
        return `<strong>${string}</strong>`
}

const template = strong('making things useful')
console.log(template)
```

Task Adding Text Using InsertAdjacentHTML()

Let's rewrite the function that we first created and send the template we created along with the section element.

```
  const updateInnerHTML = function (selector, htmlString){
  document.querySelector(selector, htmlString).insertAdjacentHTML('beforeend', htmlString)
  }
updateInnerHTML('section',template)
```

You should see the strong element now inserted into the section element. The insertAdjacentHTML(location, string) can take two parameters

Parameter location

'beforebegin': Before the element itself.

'afterbegin': Just inside the element, before its first child.

'beforeend': Just inside the element, after its last child.

'afterend': After the element itself.