

# CPSC1520 – JavaScript 1 Exercise: Working with Functions

## 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.

The function declaration below will be used to explore this concept further:

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

*Example 1. Sample function to update the innerHTML of a matched element*

## Example 1 Function Declaration Breakdown

### 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 a 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). For example, using our function from Example 1 to update the innerHTML of the first h1 on a page would look like the following:

```
updateInnerHTML( 'h1', 'updated text' );
```

#### Example 2. Calling a function

The first argument ('h1') represents the selector for the element we wish to update, and the second argument ('updated text') 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. Let's give it a try.

#### Exercise Steps

1. Open your Chrome web browser
2. Open the accompanying HTML document (functions\_exercise.html) in the browser (drag and drop)
3. Open the console
4. Copy the function from the example above into the console and press <enter>
  - a. This will declare the function for use
5. Now, use the function to update the innerHTML of the h2.intro element to whatever you like
  - a. Refresh the page once you are done to return to the original text

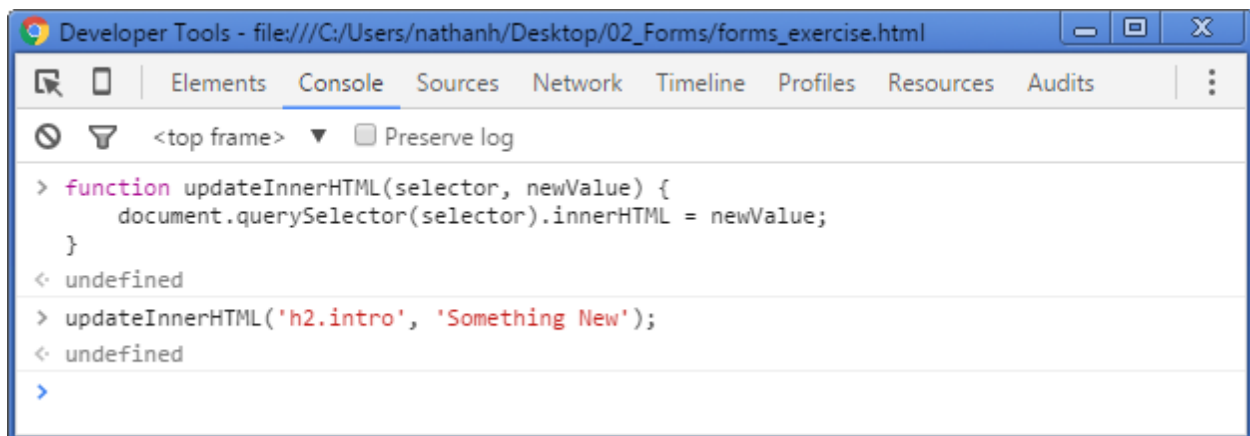


Figure 1. Sample output from the above steps

#### Returning a Value

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 example.

The following function will simply return a string that is enclosed within strong tags.

```
function strong(value) {  
    return '<strong>' + value + '</strong>';  
}
```

*Example 3. A function that returns a value*

This example introduces not only the idea of the return statement but also **concatenation**. Concatenation is simply the joining of two strings to form a new string (think of adding two strings together). In this case, the concatenation operator ('+') is used to join the string literal '<strong>' to the passed in parameter value forming a new string. This resultant string is then concatenated again with the trailing '</strong>' string literal to form yet another new string, which is then returned by the function (note: a function may only return a single value).

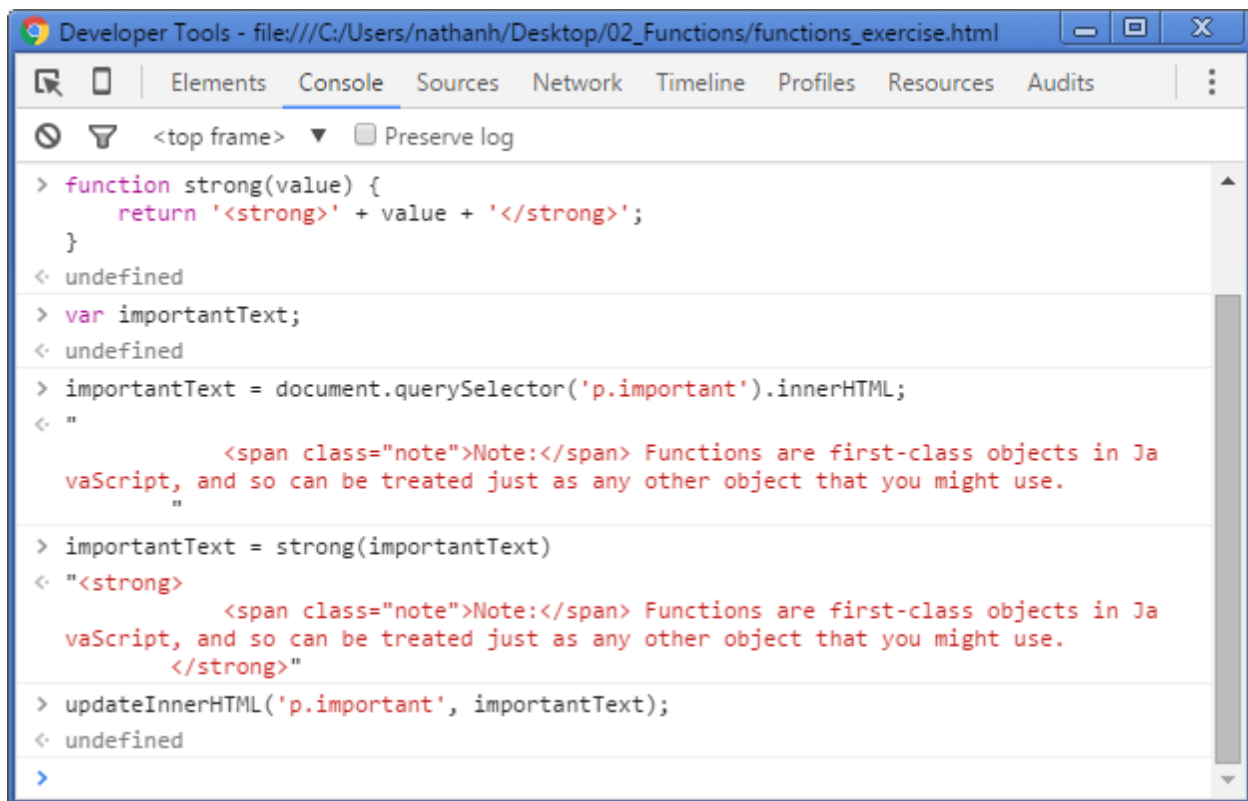
The final result of the concatenation operations is then returned to where the function was originally called. For example, calling the function from within the console will simply have the result returned to the global scope and we should see the result displayed.

This means that we can use the function call anywhere we would expect to use the returned value (note: there are many built-in functions in JavaScript and we will be introduced to only a fraction of what's available in this course).

As a final example, we will use this newly declared function to update the innerHTML of the p.important paragraph.

#### Exercise Steps

1. Declare the strong function in the console
2. Declare a variable to hold the current innerHTML value from p.important (name this variable appropriately)
3. Assign this variable the strong string value by using the strong function and passing in this same variable in as the argument
4. Finally, update the p.important innerHTML using the updateInnerHTML function



The screenshot shows the Chrome Developer Tools interface with the 'Console' tab selected. The address bar shows the file path: file:///C:/Users/nathanh/Desktop/02\_Functions/functions\_exercise.html. The console displays the following sequence of commands and their outputs:

```
> function strong(value) {  
    return '<strong>' + value + '</strong>';  
}  
< undefined  
> var importantText;  
< undefined  
> importantText = document.querySelector('p.important').innerHTML;  
< "  
    <span class="note">Note:</span> Functions are first-class objects in Ja  
vaScript, and so can be treated just as any other object that you might use.  
"  
> importantText = strong(importantText)  
< "<strong>  
    <span class="note">Note:</span> Functions are first-class objects in Ja  
vaScript, and so can be treated just as any other object that you might use.  
    </strong>"  
> updateInnerHTML('p.important', importantText);  
< undefined  
>
```

Figure 2. Sample output from the above steps

As you can see, functions can prove to be powerful tools when building up web applications.