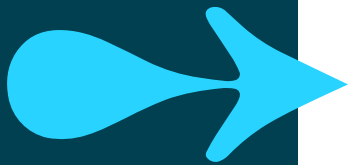Slide 1

Slide 2

INTRODUCTION

Functions
• What are functions?
• Creating functions
• Calling functions
Scope
• What is scope?
• Functions and scope

QA Functions – about

- Functions are one of the most important concepts in JavaScript
- Functions allow us to block out code for execution when we want
  - Instead of it running as soon as the browser processes it
  - Also allows us to reuse the same operations repeatedly
    - Like console.log();
  - Functions are first-class objects and are actually a type of built-in type
    - The keyword function actually creates a new object of type Function

3

In JavaScript a function is a type. It is a first-class object. Whenever we declare a function, we are actually declaring a variable of type function with the name of the function becoming the name of the variable.

When we invoke a function, we are actually calling a method of that function (a function on the function) called call – the invocation syntax is simply sugar.

Slide 4

Functions are created with the reserved word function. The newly created function block has mandatory curled braces and a function name. In the first example above, a simple function has been created that calls an alert and takes no input.

In example two, we are passing in a value this is known as a parameter or argument. It is at this point an optional parameter. If it was passed in blank, it would be undefined as a type. In the exercise, we will consider the idea of passing in the parameter and checking its value.

The first two functions just go off and do 'something'; they would often be known as 'sub routines' and are effectively a diversion from the main program allowing us to reuse code.

The third example uses the return keyword. It calculates or achieves some form of result and then returns this result to the program. It would normally be used in conjunction with a variable as we will see on the next slide.

The function must be declared before it is called. It must be declared in the current block, a previously rendered block or an external script file you have referenced, or else the function call will raise an error.

Parameters need to be passed in the correct order and remember that JavaScript has no type checking externally. We would need to be sure what is going into the function call is not garbage. Any parameters passed in are local copies and do not refer to the original variable.

The `return` statement has two purposes in a function. The first is as a method of flow control: when a return statement is encountered, the function exits immediately, without executing any code which follows the return statement. The second, and more important use of the return statement, returns a value to the caller by giving the function a value.

If a function contains no return statement, or the return statement does not specify a value, then its return value is the special undefined value, and attempting to use it in an arithmetic expression will cause an error.

QA Arrow Functions

- Can be declared as **const** (or **let**) setting a variable name to be a function
- Syntax:

```
scope name = () => implicit return;              // no arguments
scope name = arg => implicit return;             // single argument
scope name = (arg1, arg2) => implicit return;    // multiple arguments
scope name = ( …args) => ( implicit return );    // rest arguments and bracketed return
scope name = () => {                             // code block with optional defined
  // function block                              // return
  // return if required
};
```

6

Slide 7

The implementations in the top code block would be the same as the following:

```
const noArgFnImpRet = function() { return `Hello World`; }        //
returns `Hello World` when called

const noArgFnCodeBlk = function () { // Some implementation code with
return if required }

const sglArgFn = function(arg) { console.log(arg); }            // outputs
value of arg

const multiArgFn = function (num1, num2) { return num1 * num2; }  //
outputs value when called
```

Slide 8

Default values & rest parameters

- Default values were a long-standing problem with a fiddly solution
- Can provide a value for the argument and if none is passed to the function, it will use the default.

```
function doSomething(arg1, arg2, arg3=5) {
    return(arg1 + arg2 + arg3);
}
console.log(doSomething(5,5)); //15
```

- If the last named argument of a function is prefixed with … then it's value and all further values passed to the function will be captured as an array:

```
function multiply(arg1, ...args) {
args.forEach((arg,i,array) => array[i] = arg*arg1);
return args;
}
console.log(multiply(5,2,5,10)); //[10,25,50], 5 = arg1, [2, 5, 10] = …args
```

8

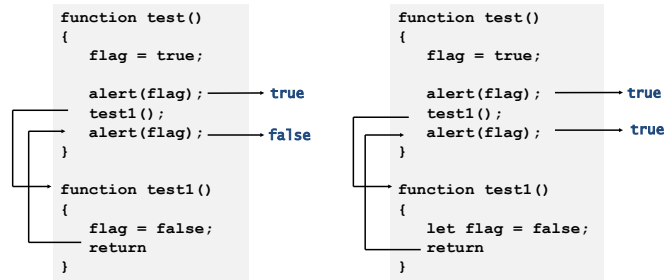The syntax used as the argument to the args.forEach is known as an Arrow function. This is a shorthand method to declare an anonymous function and is extensively used in JavaScript. The left hand side of the => indicates the arguments that are passed into the function. The right hand side is the code to execute.
This is the same as:

Args.forEach(function(arg, i, array) { array[i] = arg*arg1;});

Slide 9

Functions – scope (1)

- Scope defines where variables can be seen
  - Use the let keyword to specify scope to the current block
  - If you don't use let, then variable has 'global' scope

```
function test()
{
    flag = true;

    alert(flag); ────────→ true
    test1();
    alert(flag); ────────→ false
}

function test1()
{
    flag = false;
    return
}
```

```
function test()
{
    flag = true;

    alert(flag); ────────→ true
    test1();
    alert(flag); ────────→ true
}

function test1()
{
    let flag = false;
    return
}
```

Slide 10

Functions – scope (2)

- In the code sample to the left the flag variable is explicitly defined at global level
- In the code sample to the right it is declared in the scope of test
  - Can test1 see it?

```
let flag = true;

function test()
{
    alert(flag);        true
    test1();
    alert(flag);        false
}

function test1()
{
    flag = false;
    return
}
```
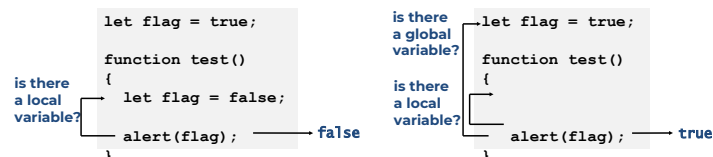
```
function test()
{
    let flag = true;

    alert(flag);        true
    test1();
    alert(flag);        true
}

function test1()
{
    flag = false;
    return
}
```

10

Slide 11

The scoping rules and the way identifiers are resolved follow an inside to outside flow in JavaScript.

If this code is executing in a function, then the first object to be checked is the function itself: local variables and parameters. After that, the 'global' store of variables is checked.

We discuss functions in greater detail later, but while we are talking about scope: variables declared in a formal parameter list are implicitly local.

```
function test(flag)      // This flag has local scope
{
  flag = false
}
```

Slide 12

The global object

- Global object for client-side JavaScript is called window
  - Global variables created using the **var** keyword are added as properties on the global object (window)
  - Global functions are methods of the current window
  - Current window reference is implicit

```
var a = 7;          window.a = 7;
alert(b);           window.alert(b);
```
                                        ← **These are equivalent**

  - Global variables created using the let keyword are NOT added as properties on the window

12

Slide 13

The global object

- Unless you create a variable within a function or block it is of global scope
  - The scope chain in JavaScript is interesting
  - JavaScript looks up the object hierarchy not the call stack
    - This is not the case in many other languages
  - If a variable is not seen in scope, it can be accidently added to global
    - Like the example in the previous slide

13

Slide 14

QA QuickLab 7- Functions

- Create and use functions
- Returning data from a function

Slide 15

REVIEW

Functions allow us to create re-usable blocks of code

Scope is a critical concept to understand and utilise in your JavaScript programming career

Functions are first-class objects, meaning we can pass them round as we would other objects and primitives