JavaScript
Arrays, Objects and Functions

WEB DEVELOPMENT FUNDAMENTALS

Introduction

- Arrays
  - What are arrays?
  - Creating arrays
  - Accessing arrays
  - Array methods
  - Destructuring
- Objects
  - Creating objects
  - Accessing objects
  - Object functions
  - Destructuring objects

2

Arrays

WEB DEVELOPMENT FUNDAMENTALS

Creating arrays

- Arrays hold a set of related data, e.g. students in a class
  - The default approach is accessed by a numeric index

a is created with no data

c is a 3 element array of string

```
let a = Array();
let b = Array(10);
let c = Array("Tom", "Dick", "Harry");
let d = [1,2,3];
```

b is a 10 element array of undefined

d is shorthand for an array

4

Creating arrays

- Arrays in JavaScript have some idiosyncrasies
  - They can be resized at any time
  - They index at 0
    - So Array(3) would have element 0, 1 and 2
  - They can be sparsely filled
    - Unassigned parts of an array are undefined
  - They can be created in short hand using just square brackets

5

Accessing arrays

- Arrays are accessed with a square bracket notation

Access an array via its index

```
let classRoom = new Array(5);
classRoom[0] = "Dave";
classRoom[4] = "Laurence";
```

Elements 1 through 3 are not yet set

- Arrays have a length property that is useful in loops

```
for (let i = 0; i < classRoom.length; i++) {
console.log(classRoom[i]);
}
```

i has 1 added to it on each iteration of the loop

6

Array object methods

- Array objects have methods
- reverse()
- join([separator])
  - Joins all the elements of the array into one string, using the supplied separator or a comma
- sort([sort function])
  - Sorts the array using string comparisons by default
  - Optional sort function compares two values and returns sort order

```
var fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
var fruitString = fruit.join("---");

// Apples---Pears---Bananas---Oranges
console.log(fruitString);
```

7

Pop and push array methods

- The push() method
  - Adds a new element to the end of the array
  - Array's length property is increased by one
  - This method returns the new length of the array

```
var fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.push('Lemons')); //5

// ['Apples', 'Pears', 'Bananas', 'Oranges', 'Lemons']
console.log(fruit);
```

8

Pop and push array methods

- The pop() method
  - Removes the last element from the end of the array
  - The array's length property is decreased by one
  - This method returns the array element that was removed

```
var fruit = ['Apples', 'Pears', 'Bananas',
'Oranges'];
console.log(fruit.pop()); //Oranges

//['Apples', 'Pears', 'Bananas']
console.log(fruit);
```

9

Shift and unshift array methods

- The unshift() method
  - Adds a new element to the beginning of the array
  - Array's length property is increased by one
  - This method returns the new length of the array

```
var fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.unshift('Kiwis')); //5

//['Kiwis','Apples', 'Pears', 'Bananas', 'Oranges']
console.log(fruit);
```

10

Shift and unshift array methods

- The shift() method
  - removes the first element from the beginning of the array
  - Array's length property is decreased by one
  - This method returns the array element that was removed

```
var fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.shift()); //Apples

//['Pears', 'Bananas', 'Oranges']
console.log(fruit);
```

11

New Methods

- Array.from() creates a real Array out of array-like objects

```
let formElements = document.querySelectorAll('input, select, textarea');
formElements = Array.from(formElements);
formElements.push(anotherElement); //works fine!
```

- Array.prototype.find() returns the first element for which the callback returns true

```
["Chris","Bruford",22].find(function(n) { return n === "Bruford"}); //"Bruford"
```

- Similarly findIndex() returns the index of the first matching element

```
["Chris","Bruford",22].findIndex(function(n) { return n === "Bruford"}); //1
```

- fill() overrides the specified elements

```
["Chris","Bruford",22,true].fill(null); //[null,null,null,null]
["Chris","Bruford",22,true].fill(null,1,2); //["Chris",null,null,true]
```

12

New Methods

- .entries(), .keys() & .values() each return a sequence of values via an iterator:

```
let arrayEntries = ["Chris","Bruford",22,true].entries();
console.log(arrayEntries.next().value); //[0,"Chris"]
console.log(arrayEntries.next().value); //[1,"Bruford"]
console.log(arrayEntries.next().value); //[2,22]
```

```
let arrayKeys = ["Chris","Bruford",22,true].keys();
console.log(arrayKeys.next().value); //0
console.log(arrayKeys.next().value); //1
console.log(arrayKeys.next().value); //2
```

```
let arrayValues = ["Chris","Bruford",22,true].values();
console.log(arrayValues.next().value); //"Chris"
console.log(arrayValues.next().value); //"Bruford"
console.log(arrayValues.next().value); //22
```

13

for...of loop

- The for-of loop is used for iterating over iterable objects (more on that later!)
- For an array it means we can loop through the array, returning each element in turn

```
//will print 1 then 2 then 3
let myArray = [1,2,3,4];
for (el of myArray) {
    if (el === 3) break;
    console.log(el);
}
```

- We could also loop through any of the iterables returned by the methods .entries(), .values() and .keys()

14

Objects

WEB DEVELOPMENT FUNDAMENTALS

Objects – data structures

- Objects in JavaScript are key – value pairs
  - Where standard arrays are index – value pairs
  - Keys are very useful for providing semantic data

```
var student = new Object();
student["name"] = "Caroline";
student["id"] = 1234;
student["courseCode"] = "LGJAVSC3";
```

- The object can have new properties added at any time
  - Known as an expando property

```
student.email = "caroline@somewhere.com";
```

16

Objects – accessing properties

- The key part of an object is often referred to as a property
  - It can be directly accessed

```
student.email ;
student["email"];
```

- When working with objects, the for in loop is very useful
  - key holds the string value of the key
  - student is the object
  - So it loops for each property in the object

```
for (let key in student) {
    console.log(`${key}:${student[key]});
}
```

17

Objects – literal notation

- There is an alternative syntactic approach to defining objects

```
let student2 = { name: "David", id: 1235, courseCode: "LGJAVSC3" };
```

- This can be combined into more complex arrays
  - Below is an indexed array containing two object literals
  - Note the comma separator

```
let classRoom = [
    { name: "David", id: 1235, courseCode: "LGJAVSC3" },
    { name: "Caroline", id: 1234, courseCode: "LGJAVSC3" }
]
```

18

Dynamic Property Names

- Dynamic property names

```
let power = 200;
n = 0;

let myCar = {
    power,
    ["prop_" + ++n]: n
};
```

19

Object.assign()

- The assign() method has been added to copy enumerable own properties to an object
- Can use this to merge objects

```
let obj1 = {a: 1};
let obj2 = {b: 2};
let obj3 = {c: 3};

Object.assign(obj1,obj2,obj3);
console.dir(obj1); //{a: 1, b: 2, c: 3}
```

- Or copy objects

```
let obj1 = {a: 1};

let obj2 = Object.assign({},obj1);
console.dir(obj2);
```

20

Everything is an object

- JavaScript is an object based programming language
  - All types extend from it
  - Including functions
  - Function is a reserved word of the language
- Theoretically, we could define our functions like this
  - Then call it using doStuff();

```
var doStuff = new Function('alert("stuff was done")');
```

- In the above example, we have added all the functionality as a string
  - The runtime will instantiate a new function object
  - Then pass a reference to the doStuff variable
  - Allowing us to call it in the same way as any other function

21

Review

- Arrays and Objects are essential collections that allow us to gather data under one roof that can then be acted upon in a coherent and concise manner
- JavaScript is an object based language
  - Everything is an object behind the scenes
  - Many very useful objects built into JavaScript

22

Functions

WEB DEVELOPMENT FUNDAMENTALS

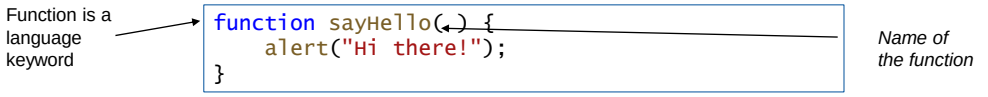
Functions – about

- 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

24

Functions – creating

- The function keyword is used to create JavaScript functions



- Parameters may be passed into a function

```
function sayHelloToSomeone(name) {
    alert(`Hi there ${name}!`);
}
```

- It may optionally return a value

```
function returnAGreetingToSomeone(name) {
    return `Hi there ${name}!`
}
```

25

Functions – calling

- Functions once created can be called
- Use the function name
- Pass in any parameters, ensuring the order
- If the function returns, pass back result

```
sayHelloToSomeone("Dave");
let r = returnAGreetingToSomeone("Adrian");
```

- Parameters are passed in as value based
  - The parameter copies the value of the variable
  - For a primitive, this is the value itself
  - For an object, this is a memory address

26

Default Values & Rest Parameters

- Default values were a long standing problem with a fiddly solution
- Now we 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
```

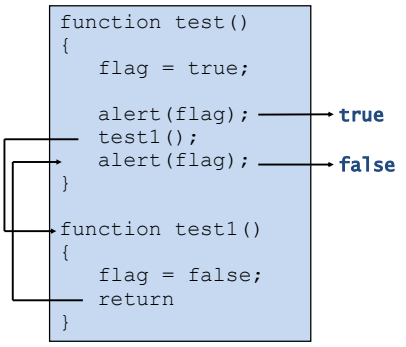
- If the last named argument of a function is prefixed with … then it's value and all further values passed to the function will be captured as an array:

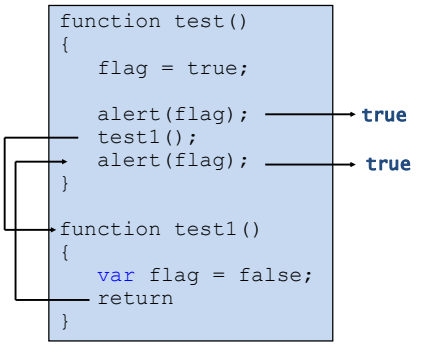
```
function multiply(arg1, ...args) {
args.forEach((arg,i,array) => array[i] =
arg*arg1);
return args;
}

console.log(multiply(5,2,5,10)); //[10,25,50]
```

27

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

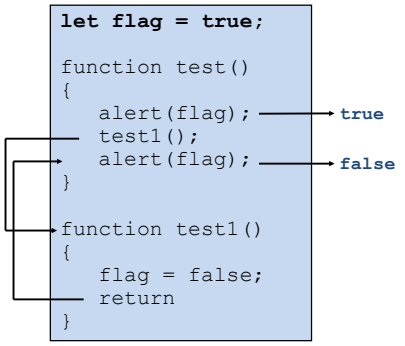


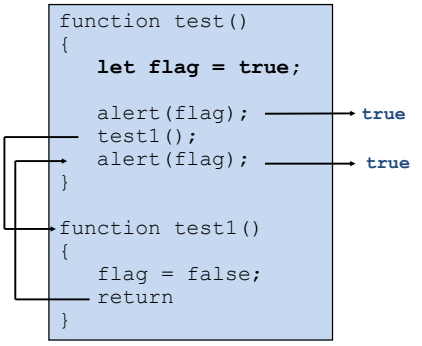


28

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?

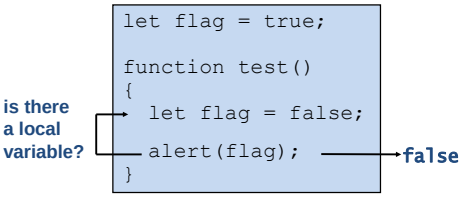


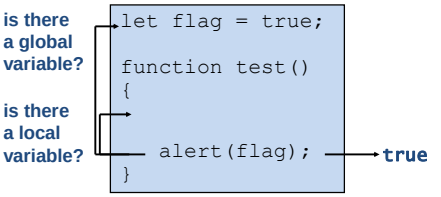


29

Functions – local vs. global scope

- Scope Chains define how an identifier is looked up
  - Start from inside and work out





- What happens if there is not a local or global variable?
  - One is added to global scope!

30

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;
alert(b);
```

```
window.a = 7;
window.alert(b);
```

These are equivalent

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

31

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

32

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

33

Exercise

- Creating and manage arrays
- Create and use functions
- Return an object from a function

34