neueda

JavaScript Essentials

Contents

- Working with variables
- Using operators
- Performing tests
- Performing loops
- Defining functions

Working with Variables

- Overview of JavaScript variables
- Declaring variables
- JavaScript primitive data types
- Assigning to undeclared variables
- Doing arithmetic

Overview of JavaScript Variables

- As with algebra (x, y, z), JavaScript variables are used to hold values or expressions
  - A variable can have a short name, like x, or a more descriptive name, like `userName`

- Rules for JavaScript variable names:
  - Variable names are case sensitive (y and Y are two different variables)
  - Variable names must begin with a letter or the underscore character

Declaring Variables

- To create (or "declare") variables in JavaScript, use the `let` keyword
  - You can declare variables with no initial value (empty initially)

```
let x;
```

  - Or you can assign values to the variables when you declare them:

```
let x = 5;
```

- We will see more on this later using **let** and **var** and we will discuss scoping rules

JavaScript Primitive Data Types

- JavaScript defines a small set of primitive types:
  - string, number, boolean

```
let userName   = "Andy";
let monthName  = "December";

let dayOfMonth = 3;
let height     = 1.67;
let favColor   = 0xFF0000;

let isWelsh    = true;
let canSing    = false;
```

- JavaScript also supports the concept of objects
  - See later in course for details

Assigning to Undeclared Variables

- If you assign values to variables that have not yet been declared…
  - The variables will automatically be declared

```
let x = 5;
let userName = "Andy";
```

- The following groups of statements have the same effect:

```
x = 5;
userName = "Andy";
```

Doing Arithmetic

- You can do arithmetic using variables and constants
  - JavaScript supports all the operators you might expect
  - E.g. + - * /

```
let radius = 10;
let area = 3.14 * radius * radius;
console.log("Area of circle is " + area);
```

Your turn

Declare some variables to represent the car you drive, or if you do not have a car – a car you would like to drive! Firstly declare two variables called **make** and **model**, then declare a variable to be the **engineSize**, and declare a variable to be the **gear** your car is in.

Now initialise those variables with appropriate values. Put in some code so that the program will print out the values of these variables, things like;

```
The make is x
The gear is y
The engine size is z
```

You will need to use the string concatenator (+).

String Handling

- Use + to add strings (variables and/or string variables)

```
txt1 = "Hello";
txt2 = "world";
txt3 = txt1 + " " + txt2 + "!!!";
```

- Use += to concatenate text to a string variable

```
let output = "<h1>";
output += "This is message ";
output += "to the user";
output += "</h1>";
```

String Concatenation

- If you mix-and-match numbers in a string concatenation, everything gets stringified

```
let swansGoals = 0;
swansGoals++;
swansGoals++;
swansGoals++;
let output = "Swansea scored " + swansGoals + " against Cardiff :-)";
```

String Concatenation using Template Literals

- You can also concatenate strings that include JavaScript variables using templating
- The syntax uses
  - Back ticks (`)instead of quotes around the text
  - To place in a variable, use the **${**stickYourExpressionHere**}** syntax

```
var a = 3, b = 3.1415;
console.log(`PI is roughly ${a} or more roughly ${b}`);
```

Your turn

Modify the previous exercice so you will display the 3 variables in a single console.log

Use template literals instead of classic concatenation

Example: `The make is w, the model is x, the engine size y, and the gear is z.`

2. Using Operators

- Arithmetic operators
- Assignment operators
- String handling
- Comparison operators
- Logical operators
- The conditional operator

Arithmetic Operators

- Arithmetic operators are used to perform arithmetic between variables and/or values
- Imagine we've already set y= 5, the following table explains the arithmetic operators

Operator	Description	Example	x value
+	Addition	x = y + 2;	7
-	Subtraction	x = y – 2;	3
*	Multiplication	x = y * 2;	10
/	Division	x = y / 2;	2.5
%	Modulus (remainder)	x = y % 2;	1
++	Increment by 1	x = ++y;	6
--	Decrement by 1	x = --y;	4

Assignment Operators

- Assignment operators are used to assign values to JavaScript variables
- Imagine we've already set x=10 and y=5, the following table explains the assignment operators

Operator	Description	Example	Same as…	x value
=	Assignment	x = y;		5
+=	Add and assign	x += y;	x = x + y;	15
-=	Subtract and assign	x -= y;	x = x - y;	5
*=	Multiply and assign	x *= y;	x = x * y;	50
/=	Divide and assign	x /= y;	x = x / y;	2
%=	Modulus and assign	x %= y;	x = x % y;	0

Comparison Operators

- Comparison operators are used to compare two variables and/or values
- Imagine we've already set y=5, the following table explains the comparison operators

Operator	Description	Example	Result
>	Is greater than?	y > 8	false
>=	Is greater than or equal?	y >= 8	false
<	Is less than?	y < 8	true
<=	Is less than or equal?	y <= 8	true

```
if (userAge >= 18)
  alert("Adult");
```

Equality and Identity Operators

- Equality operators compare two variables and/or values to see if they have the same value
  - Will coerce (i.e. convert) values to same type if necessary

Operator	Description	Example	Result
==	Equality test	5 == "5"	true
!=	Non-equality test	5 != "5"	false

Strict Checking

- Identity operators compare two variables and/or values to see if they have the same value
- Will NOT coerce (i.e. convert) values to same type

Operator	Description	Examples	Result
===	Identity test	5 === "5" 5 === 5	false true
!==	Non-identity test	5 !== "5" 5 !== 5	true false

Logical Operators

- Logical operators are used to combine multiple test conditions
- Imagine we've already set x=10 and y=5, the following table explains the logical operators

Operator	Description	Example	Result
&&	Logical and	(x < 20 && y > 1)	true
||	Logical or	(x == 20 || y == 20)	false
!	Logical not	!(x == y)	true

```
if (userAge >= 18 && userAge <= 30)
  alert("You are eligible to go on an 18-30 holiday!");
```

The Conditional Operator

- The "conditional operator" is an if-test in a single expression
  - General syntax:

```
(test-expression) ? true-expression : false-expression
```

  - Example:

```
let favTeam;

… code to ask user for favourite team …

let msg = (favTeam == "Swansea") ? "Good choice" : "Bad choice";

console.log("Your team is " + favTeam + "[" + msg + "]");
```

Your turn

In the same script, declare a variable `isStarted` and set it to true or false

With a if condition, you will display a sentence according the value of `isStarted`

Ex: The car xxx is started

Now in a variable named `status`, using a conditionnal operator (_?_:_) you will set the value as "started" or "off"

Display the same sentence using the `status` variable.

Arrays

Data structure that can hold multiple values at once, stored in a single variable.

```
let arrayName = []; // Empty array
let fruits = ["Apple", "Banana", "Cherry"];

console.log(fruits[0]); // Output: Apple
console.log(fruits[2]); // Output: Cherry
```

Arrays, common methods 1/2

Description	Usage	Result
Adds one or more elements to the end of the array	fruits.push("Orange")	["Apple", "Banana", "Cherry", "Orange"]
Removes the last element of the array	fruits.pop()	["Apple", "Banana", "Cherry"]
Removes the first element of the array	fruits.shift()	["Banana", "Cherry"]
Adds one or more elements to the beginning of the array	fruits.unshift("Mango")	["Mango", "Banana", "Cherry"]
Merges two or more arrays	fruits.concat(["Pineapple", "Grapes"])	["Mango", "Banana", "Cherry", "Pineapple", "Grapes"]

Arrays, common methods 1/2

Description	Usage	Result
Returns a portion of an array	fruits.slice(1, 3)	["Banana", "Cherry"]
Adds/Removes elements from an array	fruits.splice(1, 1, "Peach")	["Mango", "Peach", "Cherry"]
Returns the first index of a specified element	fruits.indexOf("Cherry")	2
Checks if an array contains a specified element	fruits.includes("Banana")	true
Joins all elements of an array into a string	fruits.join(", ")	"Mango, Peach, Cherry"

Flow Control

- Overview of conditional statements
- if statements
- if…else statements
- if…else…if statements
- switch statements

Overview of Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions
  - You can use conditional statements in your code to do this

- JavaScript has the following conditional statements:
  - **if**
    - Use this statement to execute some code only if a specified condition is true
  - **if…else**
    - Use this statement to execute some code if the condition is true and another code if the condition is false
  - **if…else…if**
    - Use this statement to select one of many blocks of code to be executed
  - **switch**
    - Use this statement to select one of many blocks of code to be executed

if Statements

- Use `if` statements to execute some code only if a specified condition is true

```
if (condition)
{
  code to be executed if condition is true
}
```

- Example

```
let d = new Date();
let time = d.getHours();

if (time < 12)
{
  document.write("<b>Good morning!</b>");
}
```

if...else Statements

- Use `if...else` statements to execute some code if a condition is true, otherwise execute other code

```
let d = new Date();
let time = d.getHours();

if (time < 12)
{
  document.write("<b>Good morning!</b>");
}
else
{
  document.write("<b>Good day!</b>");
}
```

if…else…if Statements

- Use `if…else…if` statements to select one of several blocks of code to be executed

```
…
if (time < 12)
   …
else if (time < 17)
   …
else
   …
```

switch Statements (1 of 2)

- Use a `switch` statement to select one of many blocks of code to be execute, based on the value of a variable

```
switch(n) {
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

switch Statements (2 of 2)

- You can have any number of case branches
- Each case value must be a constant value
- Use break to prevent "fall-through" to next branch
- The default branch is optional

```
let d = new Date();
theMonth = d.getMonth();
switch (theMonth) {
case 11:
case 0:
case 1:
  document.write("Winter");
  break;
case 2:
case 3:
case 4:
  document.write("Spring");
  break;
case 5:
case 6:
case 7:
  document.write("Summer");
  break;
default:
  document.write("Autumn");
  break;
}
```

Performing Loops

- Overview of loop statements
- for statements
- while statements
- do…while statements
- for…in statements
- Jump statements

Overview of Loop Statements

- Often when you write code, you want the same block of code to run several times
  - You can use loop statements in your code to do this

- JavaScript has the following loop statements:
  - **for**
    - Loops through a block of code a specified number of times
  - **while**
    - Loops through a block of code while a specified condition is true
    - The test about "whether to continue" is at the start of the loop
  - **do…while**
    - Loops through a block of code while a specified condition is true
    - The test about "whether to continue" is at the end of the loop
  - **for…in**
    - Loops through a collection of items, to perform some task on each item

for Loops

- Use `for` loops when you know in advance how many times the script should run

```
for (initialization; test; update)
{
  code to be executed
}
```

- Example

```
let i = 0;
for (i = 0; i < 5; i++)
{
  console.log("The number is " + i);
}
```


35

while Loops

- Use `while` loops to loop through a block of code while a specified condition is true
  - Note: the test condition is at the *start* of the loop body

```
while (condition)
{
  code to be executed
}
```

```
let i = 0;
while (i < 5)
{
  document.write("The number is " + i + "<br/>");
  i++;
}
```

do…while Loops

- Use **do…while** loops to loop through a block of code while a specified condition is true **(loops at least once)**
  - Note: the test condition is at the end of the loop body

```
let i = 0;
do
{
  document.write("The number is " + i + "<br/>");
  i++;
}
while (i < 5);
```

for…in Loops

- Use `for…in` loops to loop through all the items in a collection (e.g. an array)
  - Perform some task on each item

```
for (variable in collection) {
  code to be executed
}
```

```
let players = new Array();
players[0] = "Leon Britton";
players[1] = "Nathan Dyer";
players[2] = "Joe Allen";

let c;
for (c in players) {
  document.write("Great Swans player: " + players[c] + "<br/>");
}
```

Jump Statements

- Use **break** to immediately terminate a loop:
- Use **continue** to abandon the current iteration

```
let i;
for (i = 1; i < 20; i++)
{
  if (i % 10 == 0)
    break;

  if (i % 3 == 0)
    continue;

  console.log("i is " + i + "<br/>");
}
console.log("The end");
```

Your turn

Part 1 Using if / else

1. In your previous script, firstly, put in some logic to print out either that the car is a powerful car or a weak car based on the engine size, for example, if the size is less than or equal to 1.3.
2. Now, using an if / else if, display to the user a suitable speed range for each gear, so for example, if the gear is 5, then display the speed should be over 45mph or something. If the gear is 1, then the speed should be less than 10mph etc.

Part 2 Looping

1. We will now need to generate a loop which loops around all the years between 1900 and the year 2000 and print out all the leap years to the command console. You can use either a for or while loop to do this.
2. Bonus (Optional) Once you have done this, set it so that after 5 leap years have been displayed, it breaks out of the loop and prints ‘finished’.
3. Add all the leap years in an array.

Part 3 Using switch / case

1. Now re-write part 1 to use a switch / case construct. It should do exactly the same thing as the if / else if construct. Don’t forget to use break.

Defining Functions

- Overview of functions
- Defining functions
- Simple example
- Returning a value
- Local vs. global variables
- Additional considerations

41

Overview of Functions

- A function contains code that will be executed by an event or by a call to the function
- You may call a function from anywhere within the code

Defining Functions

- To define a function:
  - Use the `function` keyword
  - Define parameters (i.e. inputs) in brackets, ()
  - Implement the function body in braces, {}

```
function functionName(parameter1, parameter2, ... , parameterN) {
  function body code
}
```


43

Function Parameters

- Function parameters are named, but not typed
  - The parameter types are determined at run time, when you pass argument values into the function

```
function functionName(parameter1, parameter2, ... , parameterN) {
  function body code
}
```

Invoking Functions

- To invoke (i.e. call) a function:
  - Use the name of the function
  - Pass argument values inside the brackets, ()

```
functionName(value1, value2, ... , valueN);
```


45

Returning a Value

- You can return a value from a function
  - Using the `return` keyword

```
function constructMessage(firstName, lastName) {
  return "Hello " + firstName + " " + lastName;
}

function displayMessage(firstName, lastName) {
  let msg = constructMessage(firstName, lastName);
  console.log(msg);
}

constructMessage(“Nick”, “Todd”);
displayMessage(“Nick”, ”Todd”);
```

Scoping

- Variables declared without use of **let** are always global regardless of where you declare them

```
function sayHello(name) {
    console.log(name)
    anotherGlobalVariable = name;
}

globalNameVariable = "Nick"
sayHello(globalNameVariable)
console.log(anotherGlobalVariable)
```


47

Take care of var!

- Consider the following

```
function sayHello(name) {
    for(var i=0; i<3; i++) {
        console.log(i)
    }
    console.log(i) // does this work? It does!
```

- Using var scopes to the function
- If we use **let**, it scopes to the curly brackets – much better!

```
function sayHello(name) {
    for(let i=0; i<3; i++) {
        console.log(i)
    }
    console.log(i) // does this work? Not any more
```

Using const

- **const** is an alternative to the let keyword – variables defined with **const** cannot be reassigned.

```
let name = "Nick";
name = "Dave”;

const age = 27;
age = 28; //this line won’t work!
```

Parameter Passing is Optional

- The number of arguments you pass into a function doesn’t have to match the number of parameters in the function
  - If you pass too few arguments, the value of any parameters you haven't supplied is `undefined`
  - If you pass too many arguments, the surplus arguments are just ignored

- This means JavaScript doesn't support overloading
  - If you define two functions with the same name but different numbers of parameters…
  - Then the 2nd function definition replaces the 1st function definition

Default Values

- Since ECMA6, functions can also provide default values for when the value is not passed in

```
function sayHello(name=“unknown”) {
    console.log(name);
}

sayHello(); // name will be unknown in this case
```

Your turn

1. Create a function that take a temperature in farenheit and converts it into centigrade (you can find the formula online).
2. Test it

Plain Objects (Object Literals)

- Simple key-value pairs enclosed in curly braces.
- They are created directly using the {} syntax and don’t require a class or constructor function (see later)
- Often used for grouping related data together without complex behavior.

Plain Objects (Object Literals)

```
let person = {
    id: 10,
    name: "Tom",
    address: {
        street: "123 Main St",
        city: "New York",
        state: "NY",
        zip: "10001"
    }
};

// Accessing properties
console.log(person.name); // Output: Tom
console.log(person.address.street); // Output: 123 Main St
console.log(person.address.city); // Output: New York
```

Assigning Variables from Objects

- There are also some convenient syntaxes for assigning variables from objects
- For example, if you have a complex object structure, and you want two variables that refer to two bits of the structure you can do this

```
let complexObject = {
    name: "Fred",
    age: 30,
    address: {
        line1: "1 High Street",
        line2: "Bristol"
    }
}
let {name, address} = complexObject;
console.log(name);
console.log(address.line2);
```

Spread Syntax

- Another convenient syntax is referred to as spread syntax used a lot in React applications
- It allows for arrays or objects to be expanded when referenced

```
let complexObject = {
    name: "Fred",
    age: 30,
    address: {
        line1: "1 High Street",
        line2: "Bristol"
    }
}
function takeInObjectAndAugmentIt(someObject) {
    return {
        ...someObject,
        country: "UK"
    }
}
console.log(JSON.stringify(takeInObjectAndAugmentIt(complexObject)));
```

```
{"name":"Fred","age":30,"address":{"line1":"1
High Street","line2":"Bristol"},"country":"UK"}
```

Your turn

1 -Create a person object with at least name and age.
- display the full object: console.log(obj)
- display the name only

2 - Create a Copy of the Object:
- Using the spread syntax, create a new object called updatedPerson that is an exact copy of person.
- Print out the updatedPerson object to verify that it matches person.

3 - Update a Property Using Spread Syntax:
- Modify the name property of updatedPerson to "Jerry" without directly mutating the original person object.
- Print both person and updatedPerson to confirm that only updatedPerson has changed.

Summary

- Working with variables
- Using operators
- Performing tests
- Performing loops
- Defining functions

58