# Week 01 - Using JavaScript

This material can be found in this [link](https://codeberg.org/kaduardo/shu-sad-javascript/src/branch/main/week01-re-intro-js).

## Introduction

This week you will be solving a variety of programming problems using JavaScript. None of these are the types of problems with which JavaScript is usually associated. There is no DOM-manipulation, no URL parsing and no event-driven code here. Instead you’ll be working on the types of problems that are solved in introductory and advanced programming classes in many different languages.

These exercises will gently introduce you to JavaScript if you don’t know the language and will give you a chance to stretch your skills if you do.

You will probably find the Mozilla Developer Network useful as you develop your solutions: <https://developer.mozilla.org/en-US/docs/Web/javascript>

You are free to choose on how to run your code.

**Notes**

* You should try to solve each of the problems using both prototypes and ECMAScript 15 style classes.
* JavaScript has comprehensive libraries. Don’t use them unless explicitly asked by the exercise.
* Avoid built-in JavaScript functions such as Array’s sort() routine. These exercises are not about finding the “correct” answer. They are to test/improve/stretch your programming abilities.

## The Exercises

### Shapes

1. Define a base class that represents generic features of two dimensional shapes.
2. Extend your base class by defining squares, rectangles and rhomboids.
3. Demonstrate that your classes work in a small program.
4. Optionally define an HTML page containing a form that can be used to enter values for the properties of one of your shapes. Can you use the data that is entered to draw the shapes on an HTML5 canvas?

### Sorting

Here is an array of numbers:

let values = [ 10, 7, 3, 9, 1, 7, 4 ]

1. Write small programs that will sort the array in ascending order. You should implement all of the following sorting algorithms:
   * Bubble sort
   * Insertion sort
   * Quick sort
2. What happens when you try to sort this array

let values = [ “the”, “cat”, “sat”, “on”, “the”, “mat” ]

1. Can you sort the strings in descending order?

### Working with dates

You should investigate the date handling facilities that javaScript has built-in.

1. Create your own class to hold dates and times. Your class should make use of JavaScript Date class internally.
2. Add accessor methods for each component of the date.
3. Add methods to return formatted dates in US and UK styles.
4. Add a method that adds two dates together.
5. Write a method that returns today’s date as a string and prints it to the console.
6. Modify your date returning method so that you can perform simple data arithmetic and formatting.
   1. Calculate the date in one week.
   2. What is the date in 17 days? Does this code work across month boundaries?
7. Write a method which converts from UK to US date formats.
8. Generalize your previous solution such that given a string that represents a UK date and a format string you can return the date formatted appropriately. The format string should contain only the letters Y, M and D and a character to use as a separator (ie Y-D-M).
9. Write a method that receives an arbitrary number of dates as individual arguments and have it return the oldest. What happens if one of your arguments is a string that cannot represent a date?

### Temperature conversions

If you look at Wikipedia you’ll see that there are at least eight temperature scales. Write a function (or set of functions) which allows you to convert temperatures between any pair of temperature scales. The eight scales are Kelvin, Celsius, Fahrenheit, Rankine, Delisle, Newton, Rèaumur and Rømer. Try to avoid writing 56 functions!

https://en.wikipedia.org/wiki/Conversion\_of\_units\_of\_temperature

### Handling JSON

Use this JSON document (a SHU timetable) in the following exercises.

{ "classes":[  
 { "room":"9310", "subject":"Fundamentals of Programming", "start":"10:00", "time":1 },  
 { "room":"521", "subject":"Programming for Beginners", "start":"10:00", "time":1 },  
 { "room":"9306", "subject":"Networking and Server Administration", "start":"14:00", "time":2 },  
 { "room":"310", "subject":"Fundamentals of Programming", "start":"15:00", "time":1 }  
]}

Write code that displays

1. The whole data structure
2. The first data item
3. All item except the first
4. All classes taught in room 310
5. All classes not taught in room 310
6. Classes which are one hour long
7. Classes on the third floor of building nine
8. Classes which are not titled Fundamentals of something

### Pascal’s triangle

Pascal’s triangle is a triangle of numbers

1  
 1 1  
 1 2 1  
 1 3 3 1  
 1 4 6 4 1  
 1 5 10 10 5 1

That is computed as follows: - The first row just contains a 1. - The following rows are computed by adding together adjacent numbers in the row above, and adding a 1 at the beginning and at the end.

1. Define a function named pascal such that (pascal n) computes the nth row of Pascal’s triangle.
2. Write a small program that will draw Pascal’s timetable. You should be able to set the total number of lines displayed.

### Prime numbers

The Sieve of Eratosthenes is an ancient method for finding prime numbers. The process is to:

* Start by writing out all the numbers from 2 to (say) 100.
* The first number (2) is prime. Now cross out all multiples of 2.
* The first remaining number (3) is also prime. Cross out all multiples of 3.
* The first remaining number (5) is also prime… and so on.

When no numbers remain, you have found all the prime numbers in the range with which you started.

1. Define a function named crossOut such that crossOut(m, numbers) removes all multiples of m from numbers. You may choose to implement crossOut recursively or, better, use a list comprehension.
2. Now define the recursive function sieve that applies Eratosthenes’ sieve to the list of numbers it is given, and returns a list of all the prime numbers that it finds.

* This is a recursive function with a list as its argument, so you must see to it that the list gets smaller in each recursive call. Take an empty argument list as your base case.

1. Use sieve to construct the list of primes from 2 to 100.

### Counting coins

There are four types of common coins in US currency: quarters (25 cents), dimes (10), nickels (5) and pennies (1). There are 6 ways to make change for 15 cents:

1. A dime and a nickel;  
2. A dime and 5 pennies;  
3. 3 nickels;  
4. 2 nickels and 5 pennies;  
5. A nickel and 10 pennies;  
6. 15 pennies.

How many ways are there to make change for a dollar using these common coins where 1 dollar = 100 cents? Write a recursive change counter.

### Decoding Roman numerals

Create a function that takes a Roman numeral as its argument and returns its value as a numeric decimal integer. You don’t need to validate the form of the Roman numeral.

“Modern” Roman numerals are written by expressing each decimal digit of the number to be encoded separately, starting with the leftmost digit and skipping any 0s. So 1990 is rendered “MCMXC” (1000 = M, 900 = CM, 90 = XC) and 2008 is rendered “MMVIII” (2000 = MM, 8 = VIII). The Roman numeral for 1666, “MDCLXVI”, uses each letter in descending order.

Note that Roman numerals embed subtractions to create values such as 90 using the pair XC. Finding and correctly handling these subtractions can be awkward. You may start by assuming that there are no subtractions embedded within your Roman numeral.

## References and useful links

* Mozilla Developers Network - JavaScript tutorial - <https://developer.mozilla.org/en-US/docs/Web/javascript>
* MDN A re-introduciton to JavaScript (JS tutorial) (archived) - https://lia.disi.unibo.it/materiale/JS/developer.mozilla.org/en-US/docs/Web/JavaScript/A\_re-introduction\_to\_JavaScript.html
* Digital Ocean - How to code in JavaScript tutorial series - <https://www.digitalocean.com/community/tutorial_series/how-to-code-in-javascript>
* codesource.io - Object-Oriented Programming in JavaScript - <https://codesource.io/object-oriented-programming-in-javascript/>