ARRAY & LOOP

Arrays are very useful for organizing and manipulating large amounts of data(like if you want to store 300 cars in 1 variable, name of students in the school, etc). Loops are very handy for performing the same operation on each element of an array. Then we'll learn Test-Driven Development (TDD), which is the practice of writing tests for your code before you write the code itself.

A. Array

An array can hold many values under a single name, and you can access the values by referring to an index number.

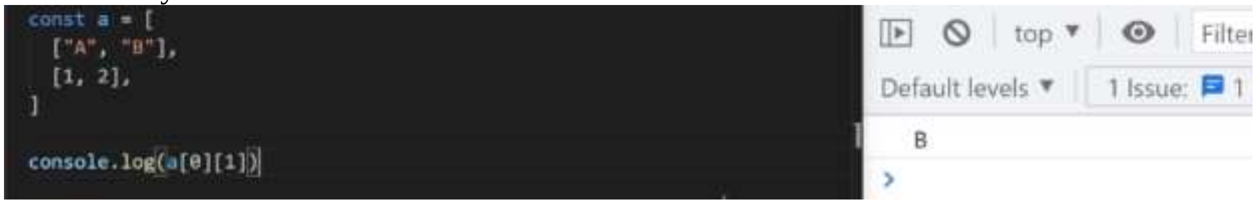
1. Creating an Array
   - Using Array Literal Method

     ```
     const array_name = [item1, item2, ...];
     ```

     It is a common practice to declare arrays 'const'
   - Using 'new'

     ```
     const cars = new Array("Saab", "Volvo", "BMW");
     ```

     The two examples(array literal method & using 'new') above do exactly the same. For simplicity, readability and execution speed, use the array literal method.
2. Nested Array

   ```
   const a = [
     ["A", "B"],
     [1, 2],
   ]

   console.log(a[0][1])
   ```

   ```
   B
   ```

   Nested Array artinya ada array dalam array(dg kata lain elemen array juga merupakan array). Cara mengaksesnya pun menggunakan array 2-dimensi, yangmana pada gambar di atas a[0][1] artinya mengakses elemen indeks ke-0 dari a, lalu mengakses elemen indeks ke-1 dari a[0], sehingga yang dipilih merupakan "B".

3. Note
   - .toString() converts an array to a string of (comma separated) array values.
   - The full array can be accessed by referring to the array name

     ```
     const cars = ["Saab", "Volvo", "BMW"];
     document.getElementById("demo").innerHTML = cars;
     ```
   - Array are special kinds of 'Objects'
     So You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

     ```
     myArray[0] = Date.now;
     myArray[1] = myFunction;
     myArray[2] = myCars;
     ```
   - Array Proeprties and Methods
     The real strength of JS arrays are the built-in array properties and methods:
     - .length //return the number of elements
     - .sort() //Sorts the array
   - Looping array elements
     You can use 'for' loop or 'Array.forEach()' function
   - The Difference Between Arrays and Objects
     In JavaScript, arrays use numbered indexes.

In JavaScript, objects use named indexes.
- You should use objects when you want the element names to be strings (text).
- You should use arrays when you want the element names to be numbers.
Note: associative arrays is arrays with named indexes, example:

```
const person = [];
person["firstName"] = "John";
```

Note: JS doesn't support Associative Array.

4. Basic Array Methods: https://www.w3schools.com/js/js_array_methods.asp
5. Array Search Methods: https://www.w3schools.com/js/js_array_search.asp
6. Array Sort Methods: https://www.w3schools.com/js/js_array_sort.asp
7. Array Iteration Methods: https://www.w3schools.com/js/js_array_iteration.asp
8. Array Const: https://www.w3schools.com/js/js_array_iteration.asp

B. Loop(ing through array)

1. For..of loop

```
const cats = ["Leopard", "Serval", "Jaguar", "Tiger", "Caracal", "Lion"];

for (const cat of cats) {
  console.log(cat);
}
```

In this example, 'for (const cat of cats)' says:
1) Given the array 'cats', get the first item in the array.
2) Assign it to the variable 'cat' and then run the code between the curly braces {}.
3) Get the next item, and repeat 2) until you've reached the end of the collection.

2. map() & filter() method

1) map()
(creates a new array by) performing a function on each array element. Example:

```
function toUpper(string) {
  return string.toUpperCase();
}

const cats = ["Leopard", "Serval", "Jaguar", "Tiger", "Caracal", "Lion"];

const upperCats = cats.map(toUpper);

console.log(upperCats);
// [ "LEOPARD", "SERVAL", "JAGUAR", "TIGER", "CARACAL", "LION" ]
```

2) filter()
to test each item in a collection, and create a new collection(/array) containing only items that match. Example:

```
function lCat(cat) {
  return cat.startsWith("L");
}

const cats = ["Leopard", "Serval", "Jaguar", "Tiger", "Caracal", "Lion"];

const filtered = cats.filter(lCat);

console.log(filtered);
// [ "Leopard", "Lion" ]
```

Note: map() & filter() often used with Function Epressions.

```
const cats = ["Leopard", "Serval", "Jaguar", "Tiger", "Caracal", "Lion"];

const filtered = cats.filter((cat) => cat.startsWith("L"));
console.log(filtered);
// [ "Leopard", "Lion" ]
```

3. break & continue

1) Exiting loops with ‘break’

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

2) Skipping iteration with ‘continue’

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

3) Break/Continue from nested loop

We use *label* to jump from nested loop. A *label* is an identifier with a colon before a loop:

```
labelName: for (...) {
  ...
}
```

Example:

```
for (let i = 0; i < 3; i++) {

  for (let j = 0; j < 3; j++) {

    let input = prompt(`Value at coords (${i},${j})`, '');

    // what if we want to exit from here to Done (below)?
  }
}

alert('Done!');
```

Solution:

```
outer: for (let i = 0; i < 3; i++) {

  for (let j = 0; j < 3; j++) {

    let input = prompt(`Value at coords (${i},${j})`, '');

    // if an empty string or canceled, then break out of both loops
    if (!input) break outer; // (*)

    // do something with the value...
  }
}

alert('Done!');
```

The ‘continue’ directive can also be used with a label. In this case, code execution jumps to the next iteration of the labeled loop.

Note:
```
break label; // jump to the label below (doesn't work)

label: for (...)
```

A ‘break’ directive must be inside a code block. Technically, any labelled code block will do, e.g.:
```
label: {
  // ...
  break label; // works
  // ...
}
```

A ‘continue’ is only possible from inside a loop.

4. Rest Parameters [func(…args)]
The rest parameter syntax allows a function to accept an indefinite number of arguments as an array. Example:
```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}

console.log(sum(1, 2, 3));
// Expected output: 6

console.log(sum(1, 2, 3, 4));
// Expected output: 10

```
Syntax:
```
function f(a, b, ...theArgs) {
  // …
}
```

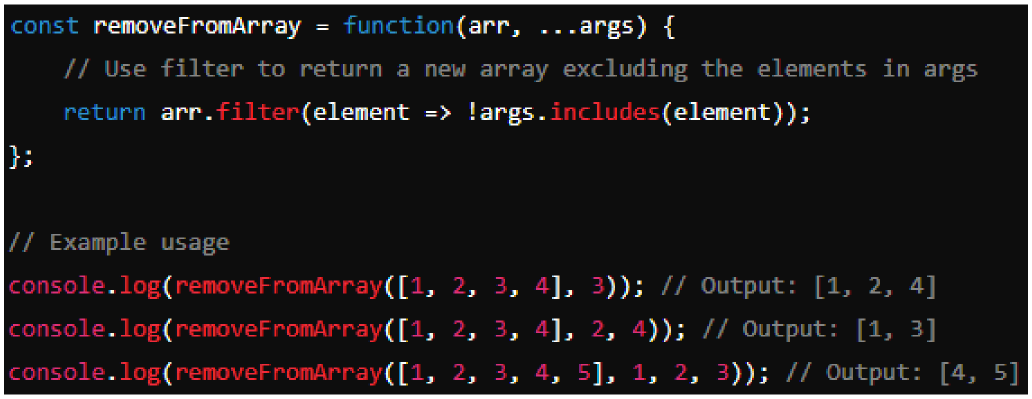
A function definition can only have one rest parameter, and the rest parameter must be the last parameter in the function definition. which will cause all remaining (user supplied) parameters to be placed within an Array object.
```
function myFun(a, b, ...manyMoreArgs) {
  console.log("a", a);
  console.log("b", b);
  console.log("manyMoreArgs", manyMoreArgs);
}

myFun("one", "two", "three", "four", "five", "six");

// Console Output:
// a, one
// b, two
// manyMoreArgs, ["three", "four", "five", "six"]
```
```
function wrong1(...one, ...wrong) {}
function wrong2(...wrong, arg2, arg3) {}
```
Array Methods can be used on rest parameters:
```
function sortRestArgs(...theArgs) {
  const sortedArgs = theArgs.sort();
  return sortedArgs;
}

console.log(sortRestArgs(5, 3, 7, 1)); // 1, 3, 5, 7
```

Another example of implementation …arg :



```
const removeFromArray = function(arr, ...args) {
    // Use filter to return a new array excluding the elements in args
    return arr.filter(element => !args.includes(element));
};

// Example usage
console.log(removeFromArray([1, 2, 3, 4], 3)); // Output: [1, 2, 4]
console.log(removeFromArray([1, 2, 3, 4], 2, 4)); // Output: [1, 3]
console.log(removeFromArray([1, 2, 3, 4, 5], 1, 2, 3)); // Output: [4, 5]
```

5. While.. & do..while
   Jika for.. digunakan untuk perulangan yg diketahui jumlahnya, maka while digunakan untuk perulangan yg tidak diketahui jumlahnya. misal: push up 10x(menggunakan for), push up sampai tidak kuat push up lagi(menggunakan while).
   Do..while sama seperti while hanya saja do..while akan dieksekusi minimal 1x(berbeda dengan while yg dieksekusi minimal 0x).

C. Test Driven Development (TDD)

TDD is a phrase you often hear in the dev world. It refers to the practice of writing automated tests that describe how your code should work before you actually write the code. We will teach you the art of actually writing these tests later in the course.