CSY1063 Web Development Week 12

Chris.Rafferty@northampton.ac.uk

Learning Objectives

- This week we will be covering
  - Debugging and common errors
  - String methods
  - Function
    - Arguments
    - Returns
    - Arrow functions
  - Local storage
  - Array methods

Debugging

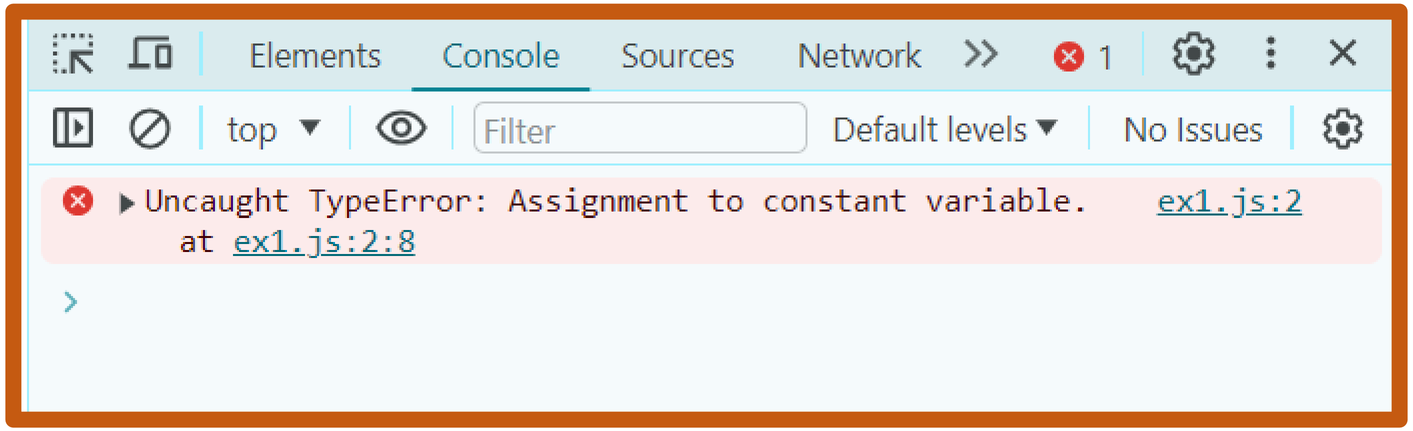
- If your code isn't working as you expected
  - Nothing is happening when you click on an element and you're expecting it to do something

- You need to start debugging it
- Debugging is finding and fixing errors or bugs in the code. When software does not work as expected, programmers study the code to determine why any errors occurred.

- Start by opening developer tools

Debugging pt2

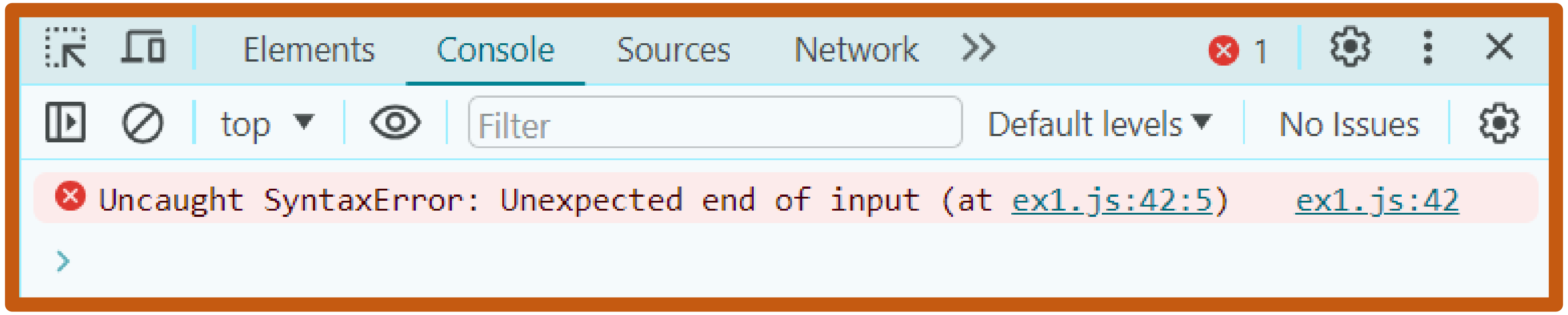
- Select "Console" and you'll see the JavaScript console
- Any errors in your code will be highlighted here



```
Uncaught TypeError: Assignment to constant variable.    ex1.js:2
    at ex1.js:2:8
```

Common error #1

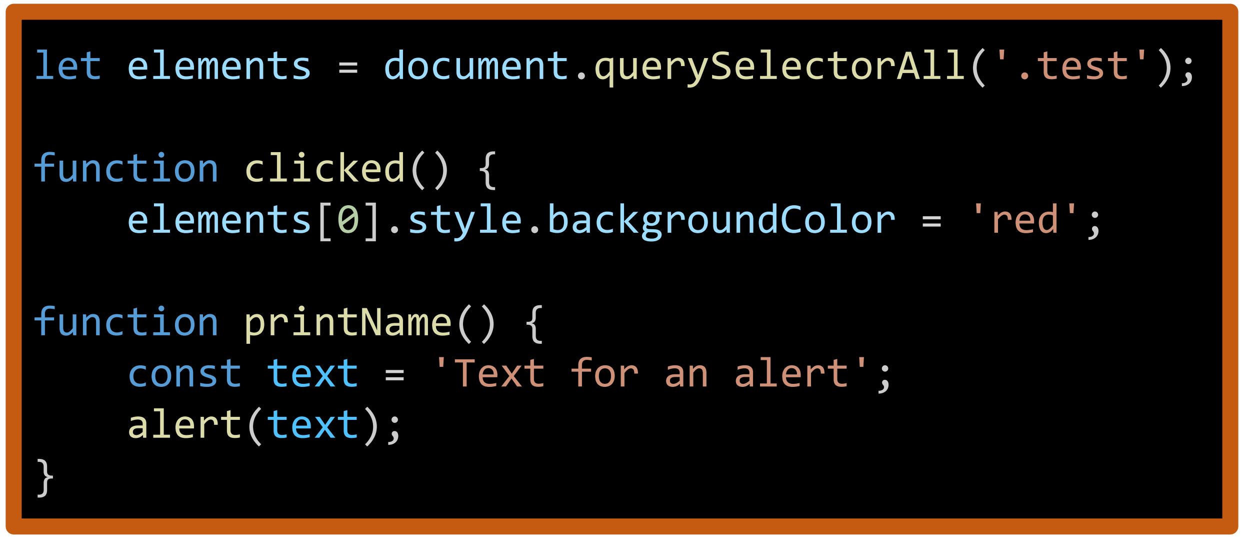
- Unexpected end of input
- Every opening brace { in JavaScript requires a closing brace }
- This error means that you are missing a closing brace somewhere



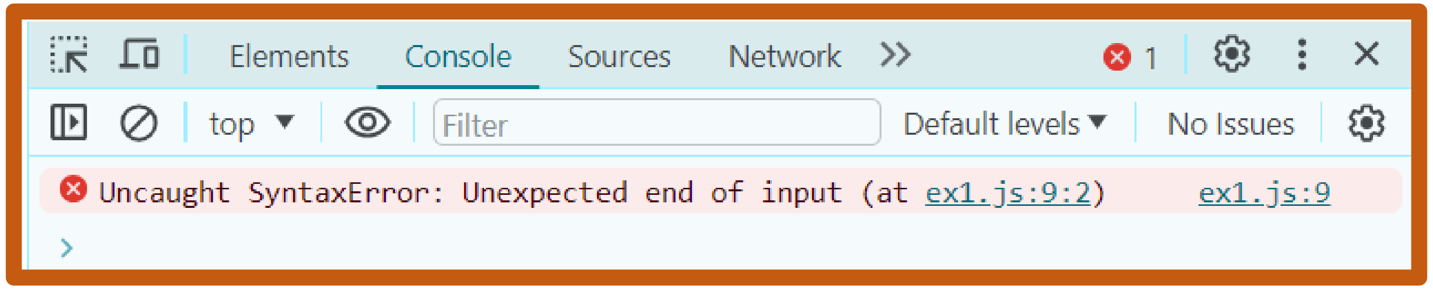
```
Uncaught SyntaxError: Unexpected end of input (at ex1.js:42:5)    ex1.js:42
```

Common error #1 - test

- Using the error message find the mistake



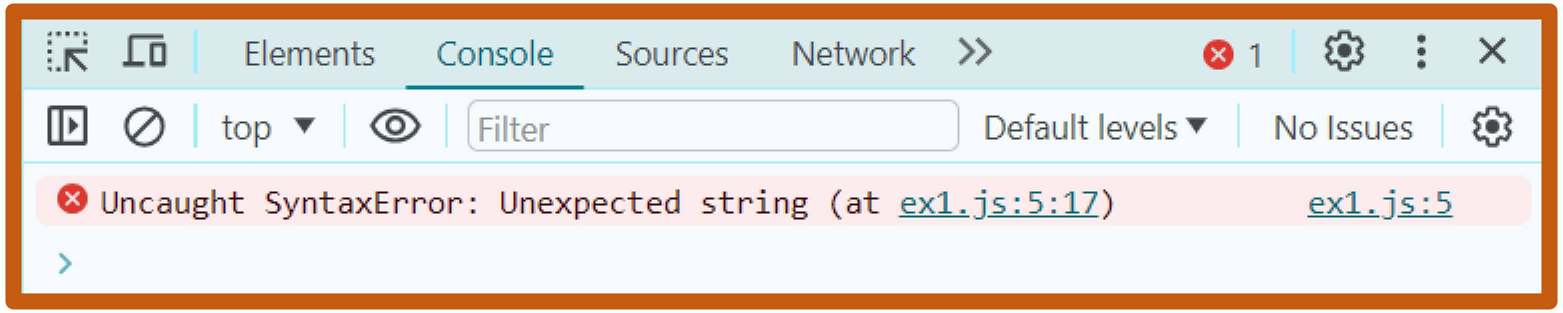
```
let elements = document.querySelectorAll('.test');

function clicked() {
    elements[0].style.backgroundColor = 'red';

function printName() {
    const text = 'Text for an alert';
    alert(text);
}
```



```
Uncaught SyntaxError: Unexpected end of input (at ex1.js:9:2)     ex1.js:9
```

Common error #2

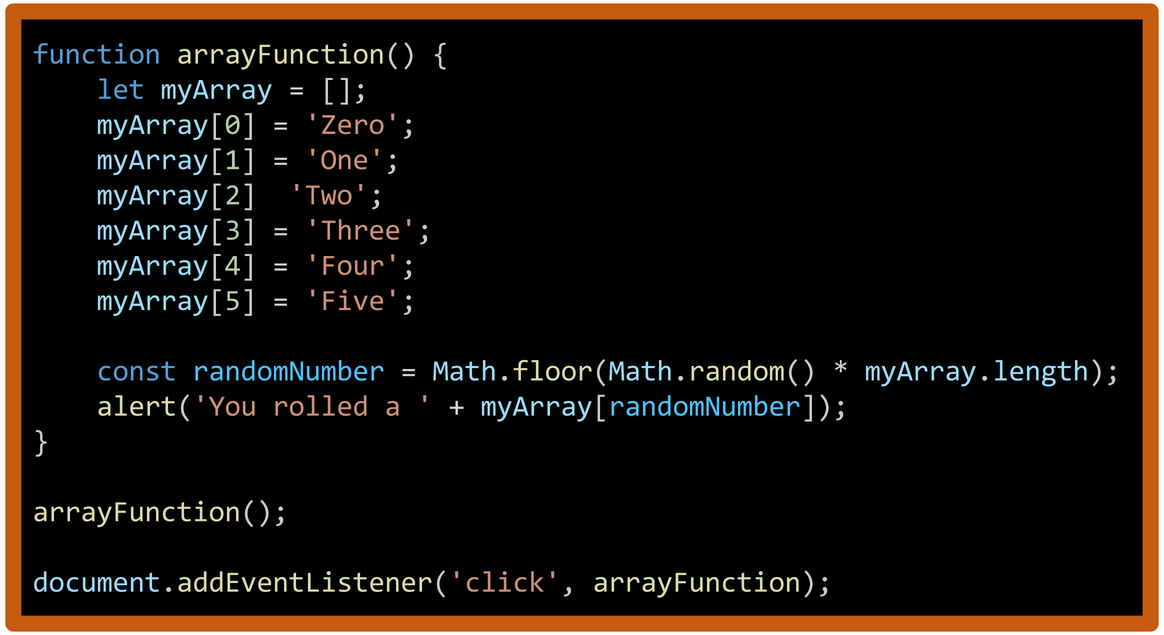
- Unexpected string/Unexpected variable/unexpected number
- This means there is a syntax error, you are missing part of the code or have the code in the wrong order



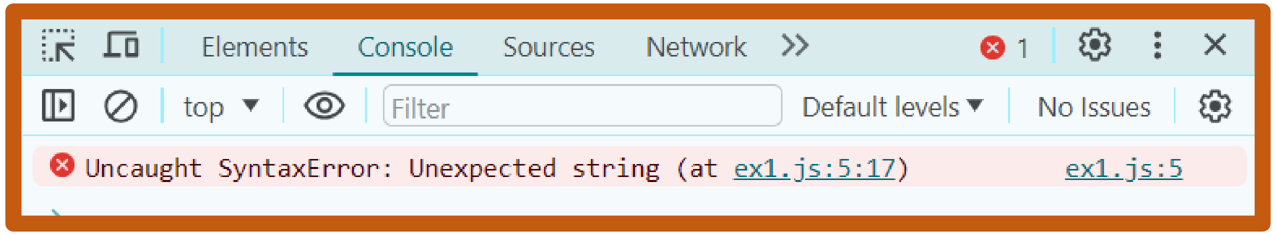
```
Uncaught SyntaxError: Unexpected string (at ex1.js:5:17)          ex1.js:5
```

Common error #2 - test

- Using the error message find the mistake



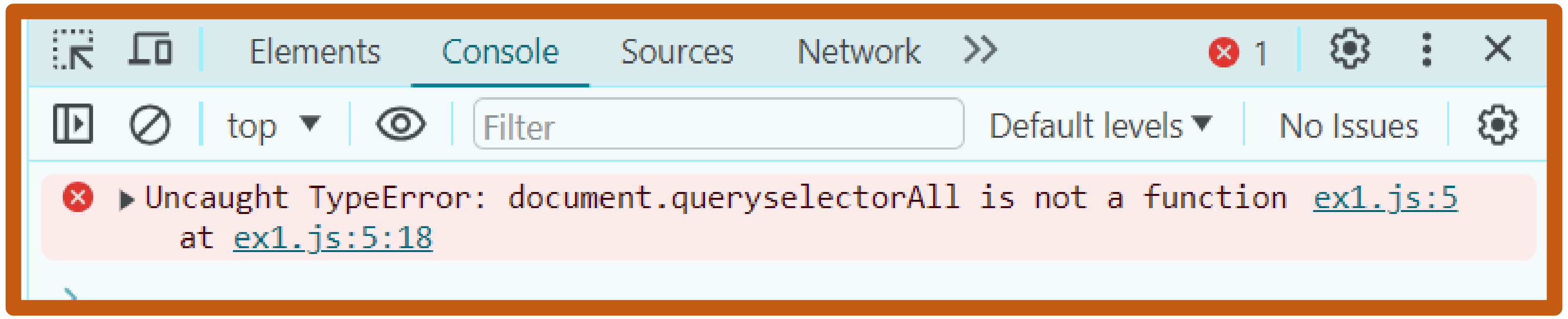
```
function arrayFunction() {
    let myArray = [];
    myArray[0] = 'Zero';
    myArray[1] = 'One';
    myArray[2]  'Two';
    myArray[3] = 'Three';
    myArray[4] = 'Four';
    myArray[5] = 'Five';

    const randomNumber = Math.floor(Math.random() * myArray.length);
    alert('You rolled a ' + myArray[randomNumber]);
}

arrayFunction();

document.addEventListener('click', arrayFunction);
```



```
Uncaught SyntaxError: Unexpected string (at ex1.js:5:17)        ex1.js:5
```

Common error #3

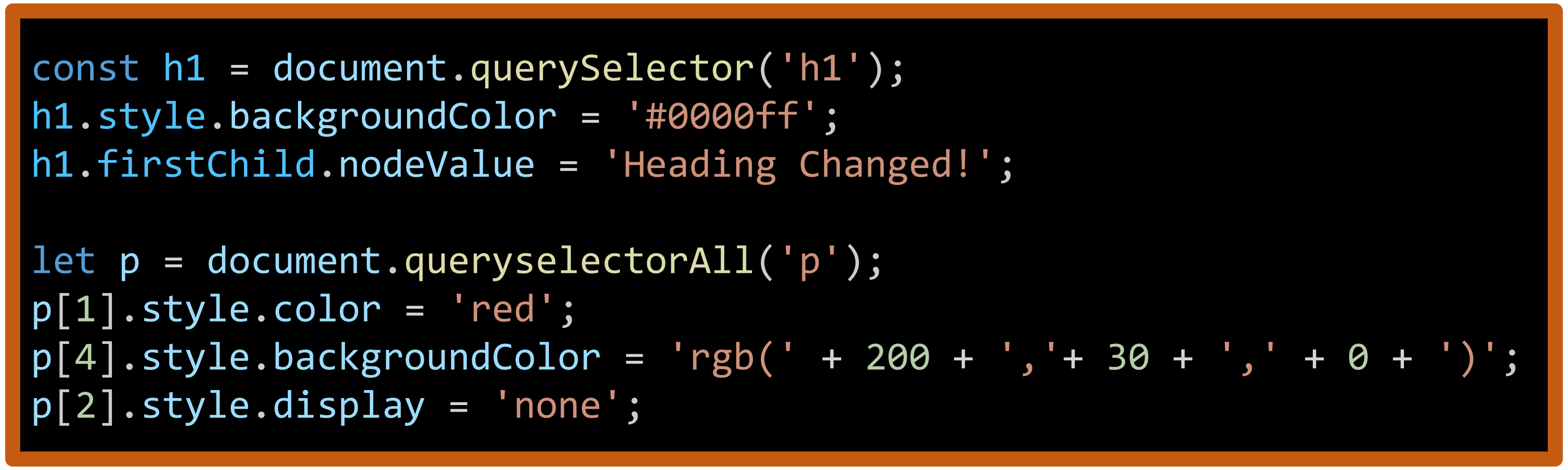
- Typos
- JavaScript is case sensitive
- It's very easy to get an error like this



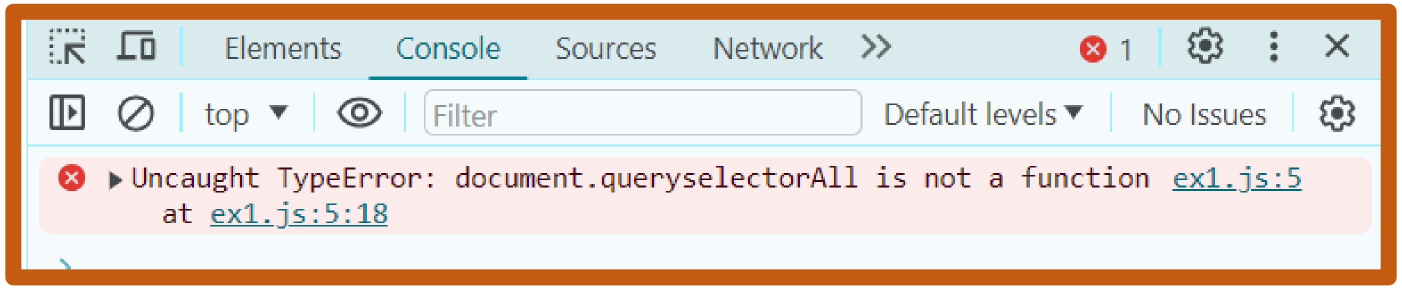
```
Uncaught TypeError: document.queryselectorAll is not a function ex1.js:5
    at ex1.js:5:18
```

Common error #3 - test

- Using the error message find the mistake



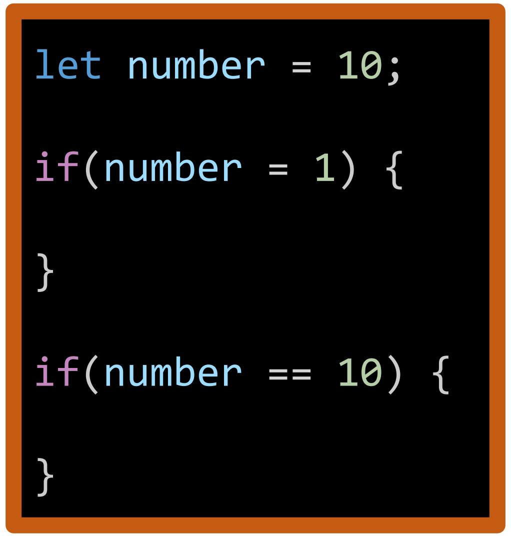
```
const h1 = document.querySelector('h1');
h1.style.backgroundColor = '#0000ff';
h1.firstChild.nodeValue = 'Heading Changed!';

let p = document.queryselectorAll('p');
p[1].style.color = 'red';
p[4].style.backgroundColor = 'rgb(' + 200 + ','+ 30 + ',' + 0 + ')';
p[2].style.display = 'none';
```



```
Uncaught TypeError: document.queryselectorAll is not a function    ex1.js:5
    at ex1.js:5:18
```

Common error #4

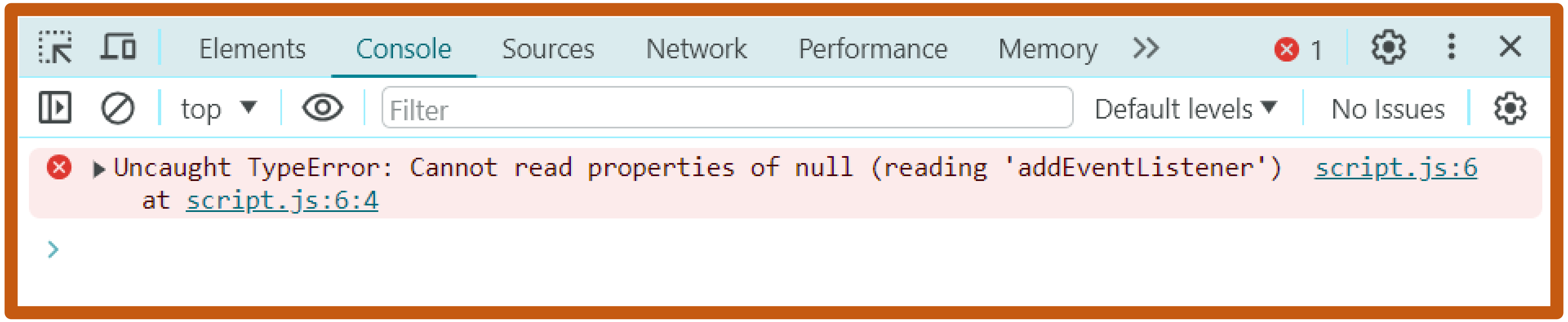
- = and ==. This won't produce an error in the console
- When using an if statement and comparing two values, you must used the == operator instead of =
  - = will assign the number variable the value 1
  - == should be used for if statements



```
let number = 10;

if(number = 1) {

}

if(number == 10) {

}
```

Common error #5

- Typos!
- Get in the habit of checking for everything
- Null means no value



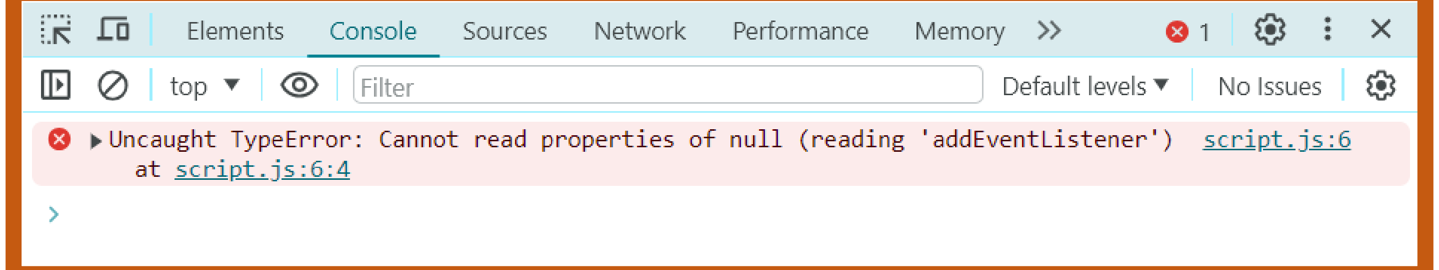
```
Uncaught TypeError: Cannot read properties of null (reading 'addEventListener')   script.js:6
    at script.js:6:4
```

Common error #5 - test

- Using the error message find the mistake

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, initial-
          scale=1.0">
    <title>Week 12</title>
    <script src="script.js" defer></script>
</head>
<body>
    <h1 id="heading">Heading...</h1>
    <p>Paragraph</p>
</body>
</html>
```

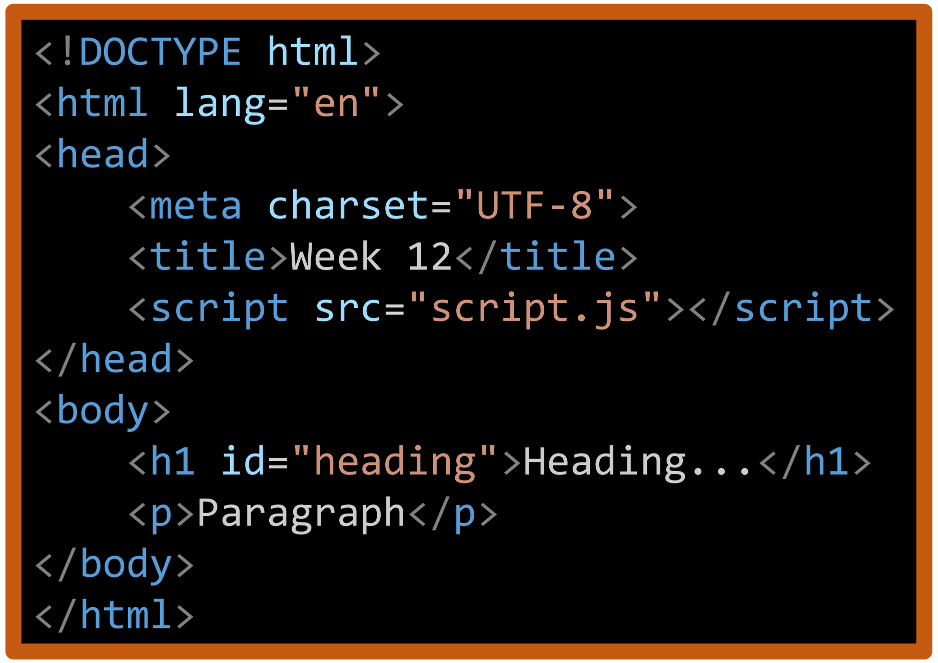
```javascript
let paragraph = document.querySelectorAll('p')[0];
const h1 = document.querySelector('heading');

paragraph.firstChild.nodeValue = 'New text for paragraph';

h1.addEventListener('click', function(){
    h1.firstChild.nodeValue = 'Heading Changed!!!';
});

const body = document.querySelector('body');
body.appendChild(document.createElement('div'));
```



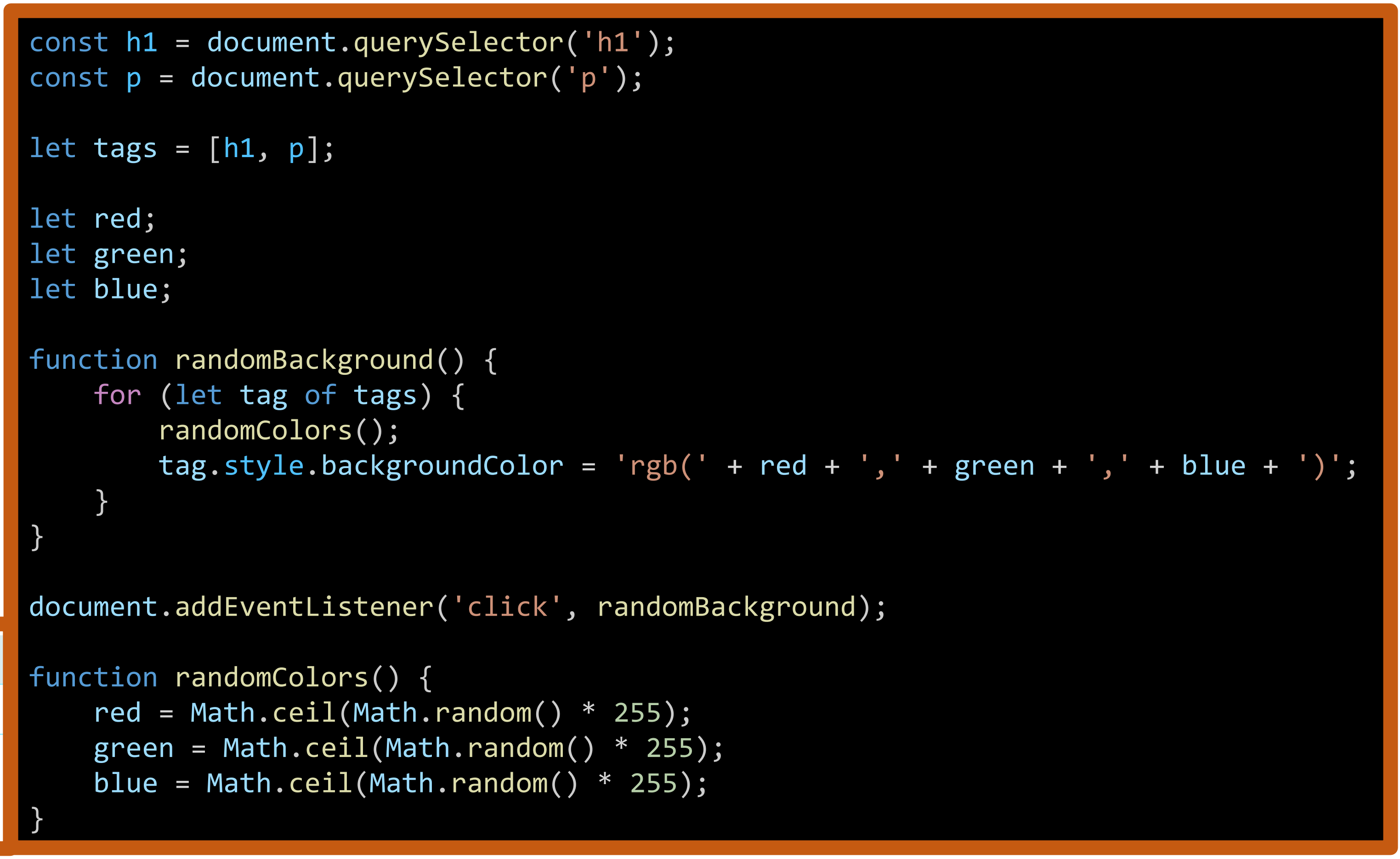
```
Uncaught TypeError: Cannot read properties of null (reading 'addEventListener')   script.js:6
    at script.js:6:4
```

Common error #6 - test

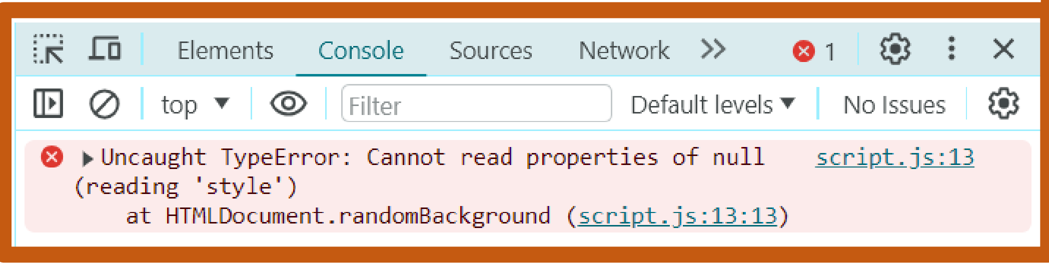
- Using the error message find the mistake



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Week 12</title>
    <script src="script.js"></script>
</head>
<body>
    <h1 id="heading">Heading...</h1>
    <p>Paragraph</p>
</body>
</html>
```



```javascript
const h1 = document.querySelector('h1');
const p = document.querySelector('p');

let tags = [h1, p];

let red;
let green;
let blue;

function randomBackground() {
    for (let tag of tags) {
        randomColors();
        tag.style.backgroundColor = 'rgb(' + red + ',' + green + ',' + blue + ')';
    }
}

document.addEventListener('click', randomBackground);

function randomColors() {
    red = Math.ceil(Math.random() * 255);
    green = Math.ceil(Math.random() * 255);
    blue = Math.ceil(Math.random() * 255);
}
```



```
Uncaught TypeError: Cannot read properties of null    script.js:13
(reading 'style')
    at HTMLDocument.randomBackground (script.js:13:13)
```

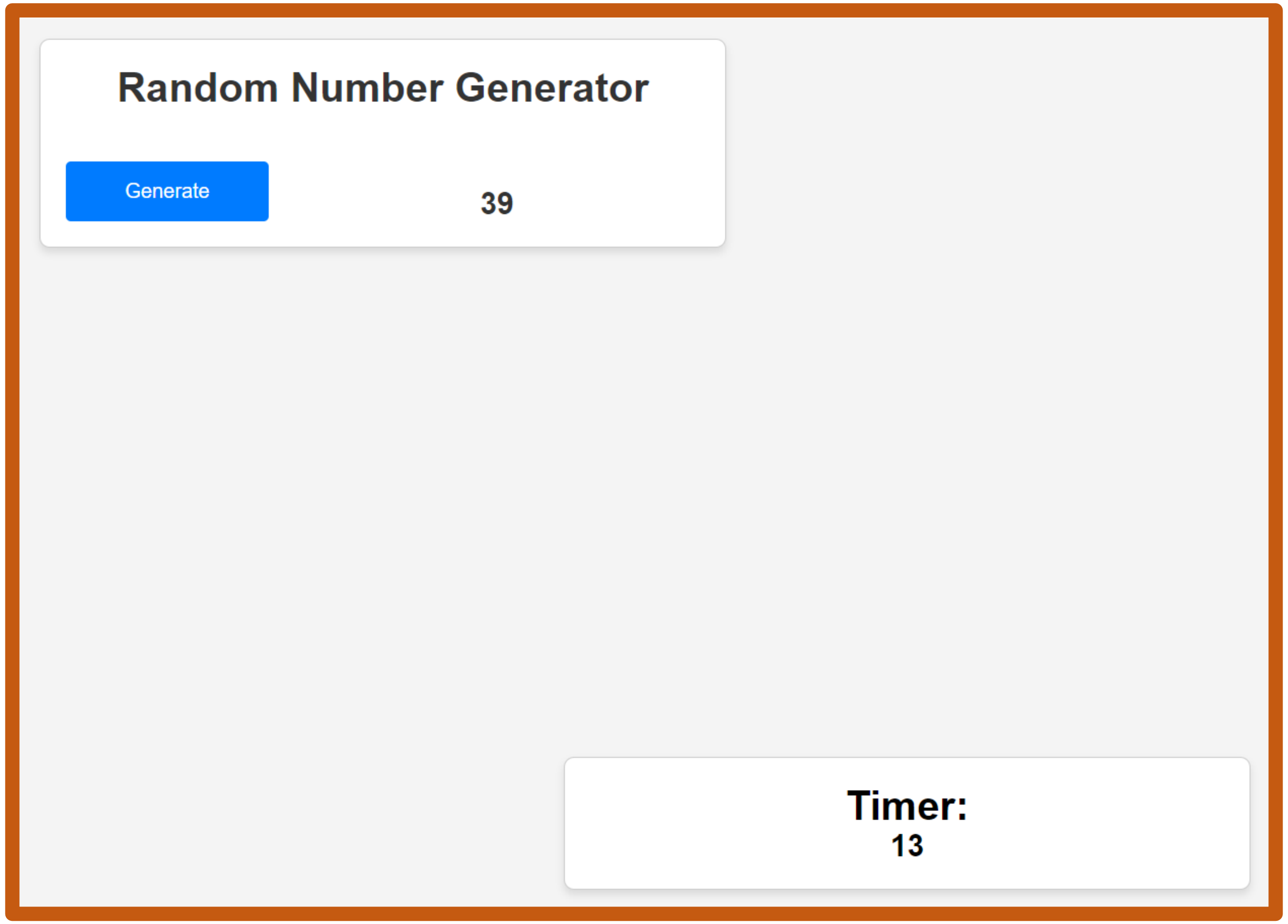
Exercise

- Download ex1.zip
- Using the console try and fix the code
- The random number generator should create and display a new random number when the button is clicked

- A timer in the bottom right should go up every 1 second

- **Hint**: Fix one error at a time

Exercise 1 example



Strings

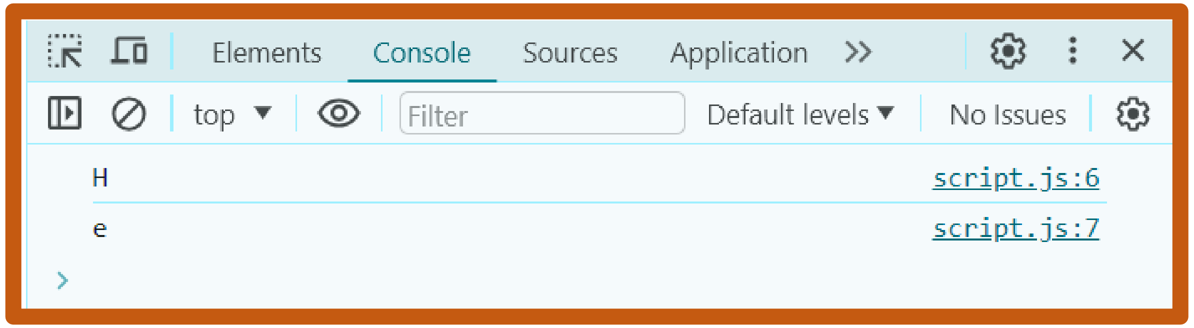
- Strings are one of the most common pieces of data we use in JavaScript
- We can use a few useful methods which can make managing strings significantly easier

- There are inbuilt methods we can use to
    - Separate a string and return it in an array
    - Capitalize all the characters
    - Remove any unnecessary spaces
    - Check to see if a string includes a specific word

Strings - charAt

- The charAt() method returns the character at a given index in a string
  - charAt(0) returns the first letter
  - charAt(1) returns the second letter

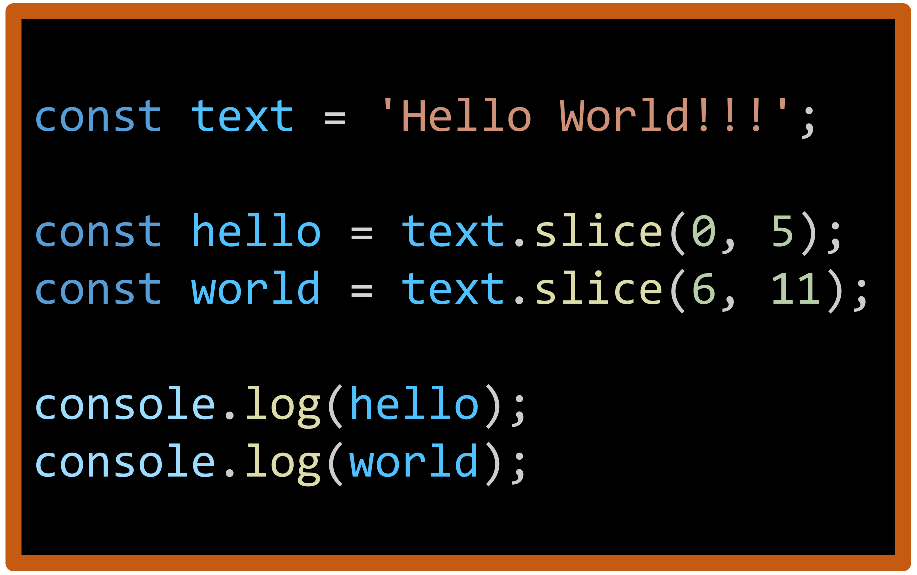
```
const text = 'Hello World!!!';

const firstLetter = text.charAt(0);
const secondLetter = text.charAt(1);

console.log(firstLetter);
console.log(secondLetter);
```



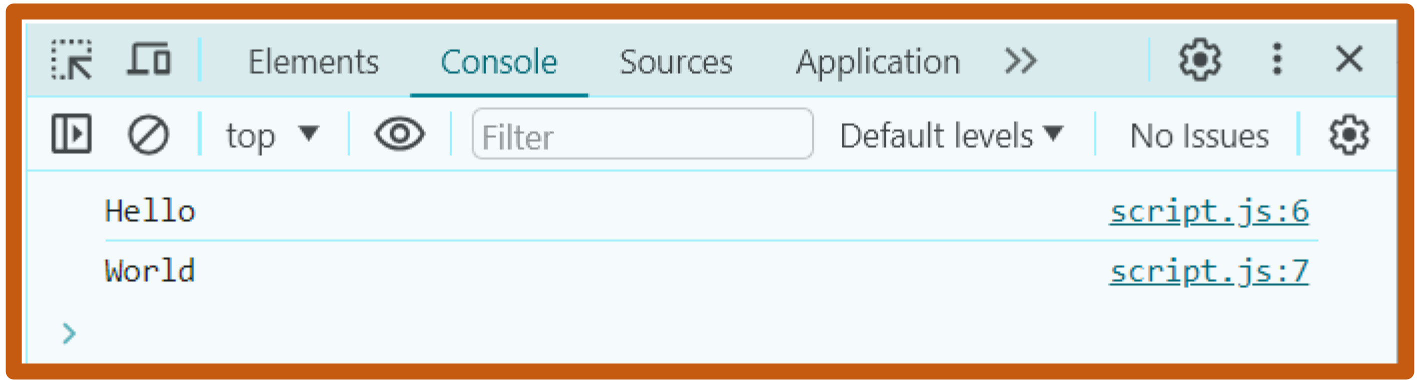
```
H                                                    script.js:6
e                                                    script.js:7
>
```

String - slice

- You can use the slice() method to extract part of a string
  - Slice(start index position, end index position)



```
const text = 'Hello World!!!';

const hello = text.slice(0, 5);
const world = text.slice(6, 11);

console.log(hello);
console.log(world);
```

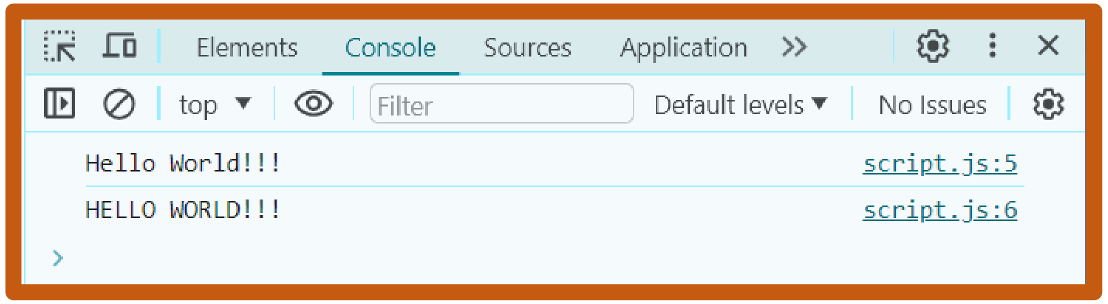


```
Hello                                                    script.js:6
World                                                    script.js:7
>
```

String - toUpperCase

- The toUpperCase() method converts a string to uppercase letters
  - ‘hello’ to ‘HELLO’

```
const text = 'Hello World!!!';

const upperCaseText = text.toUpperCase();

console.log(text);
console.log(upperCaseText);
```

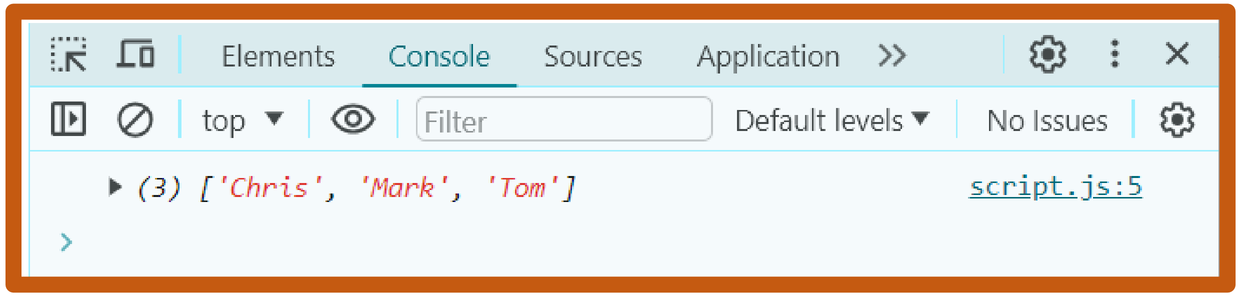


```
Hello World!!!                                    script.js:5
HELLO WORLD!!!                                    script.js:6
```

String - split

- split() can be used to split a string into an array
  - You need to decide where to split the string
  - ‘Chris_Mark_Tom’ could be split by the ‘_’ to become [‘Chris’, ’Mark’, ’Tom’]

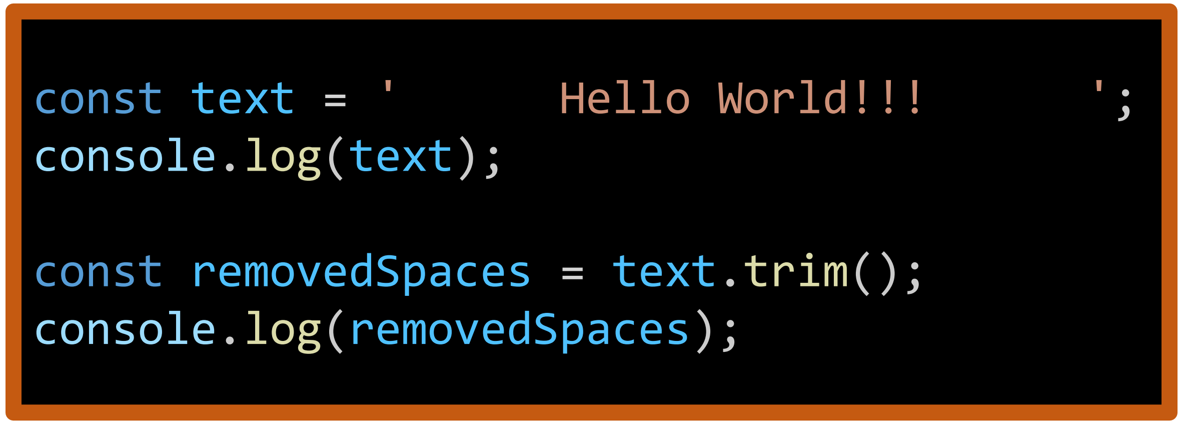
```
const names = 'Chris_Mark_Tom';

const namesArray = names.split("_");

console.log(namesArray);
```



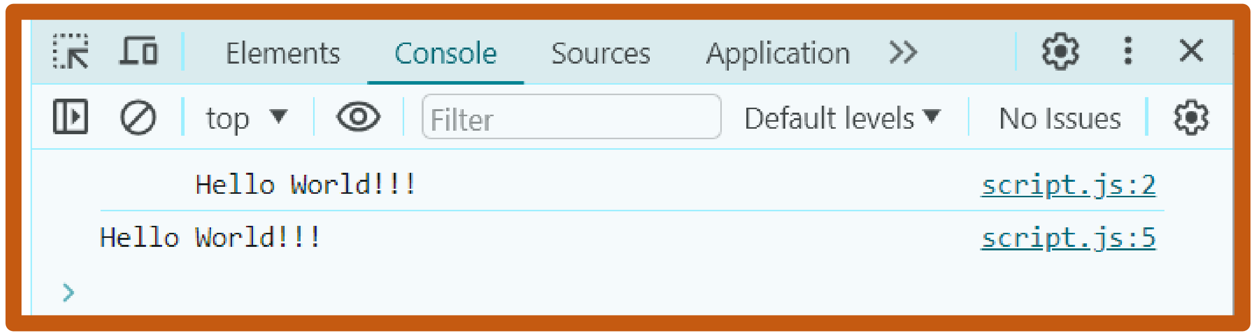
```
▶ (3) ['Chris', 'Mark', 'Tom']
>
```

String - trim

- trim() can be used to remove any whitespace from the start and end of a string



```
const text = '        Hello World!!!        ';
console.log(text);

const removedSpaces = text.trim();
console.log(removedSpaces);
```



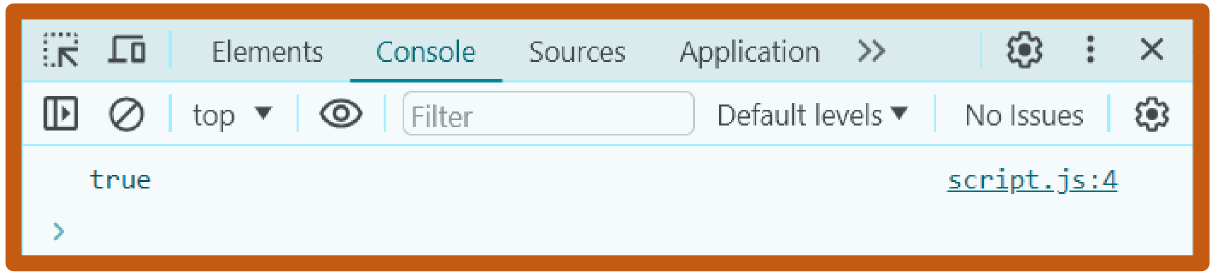
```
      Hello World!!!                                   script.js:2
Hello World!!!                                         script.js:5
```

- trim() does not change the original string

String - includes

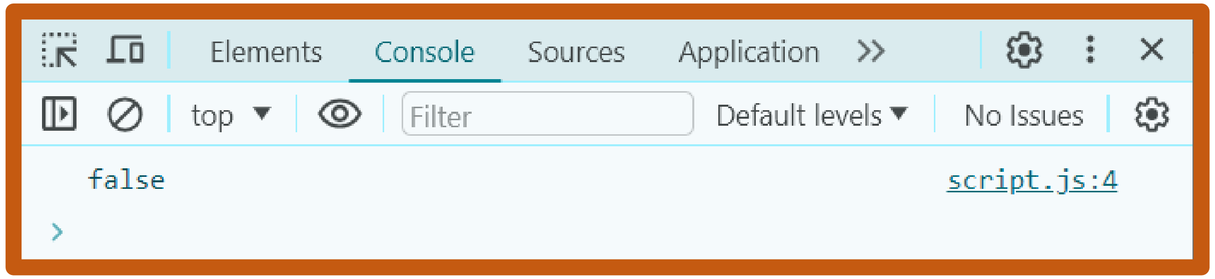
- includes() will return either true or false if a string contains a specified string

```
const text = 'Hello World!!!';
const check = text.includes('Hello');

console.log(check);
```



```
true                                                    script.js:4
```

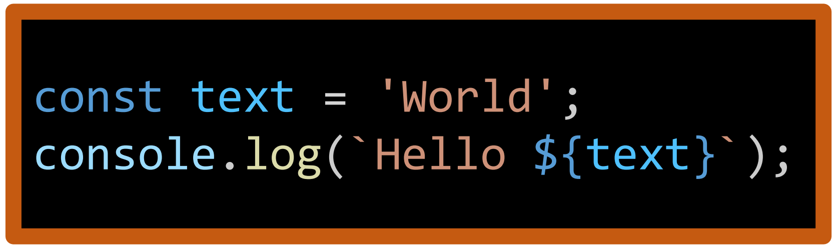
```
const text = 'Hello World!!!';
const check = text.includes('Chris');

console.log(check);
```



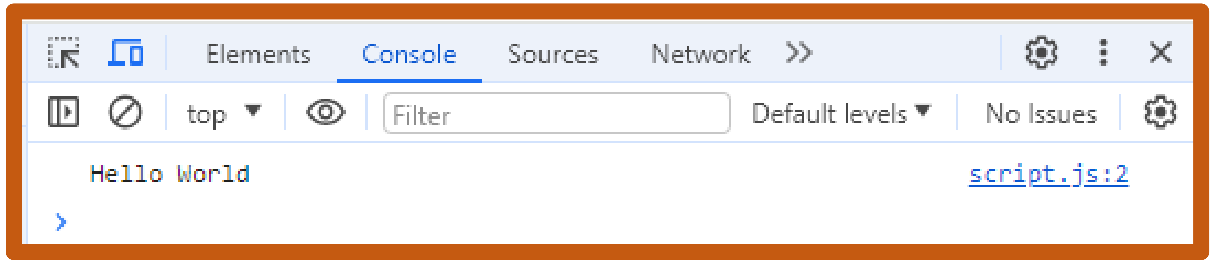
```
false                                                   script.js:4
```

Template strings

- You can concatenate variables in strings by using a template string instead of the ‘+’ operator
- Template strings use ` (backtick) and ${variable}
  - Usually under the esc key



```
const text = 'World';
console.log(`Hello ${text}`);
```



```
Hello World                                                script.js:2
```

Template strings example

```
const circle = document.querySelector('#circle');

const random1 = Math.ceil(Math.random() * 255);
const random2 = Math.ceil(Math.random() * 255);
const random3 = Math.ceil(Math.random() * 255);

circle.style.backgroundColor = 'rgb(' + random1 + ',' + random2 + ',' + random3 + ')';
```

```
const circle = document.querySelector('#circle');

const random1 = Math.ceil(Math.random() * 255);
const random2 = Math.ceil(Math.random() * 255);
const random3 = Math.ceil(Math.random() * 255);

circle.style.backgroundColor = `rgb(${random1},${random2},${random3})`;
```

Functions

- Long functions that do a lot are very difficult to follow
- And difficult to write/understand
- Breaking code up into smaller chunks makes it a lot easier to write and understand

Functions pt2

- So far we’ve used functions to respond to events
  - Page load
  - Button clicks
  - Timers

- It’s also possible to write a function and call it yourself
  - Every function has a name
  - You can explicitly call a function by using its name followed by brackets

Functions pt3

- Functions can be used to reduce repeated code and break the code into smaller chunks
- By putting code inside functions it can be a lot more manageable and easier to read/follow

Arguments

- When you call a function, the code inside the function is run
- It’s also possible to send values to a function that are unique each time it is called
- These are called arguments
- You have already used arguments when using the in-built JavaScript function

Arguments pt2

- When querySelector() is called, you send the function the id/class of the element you want to find
- When you use alert you send the function the string you want to print to the screen
- These are arguments

```
const heading = document.querySelector('#heading');
const points = document.querySelectorAll('.point');

alert('Hello World!');
```

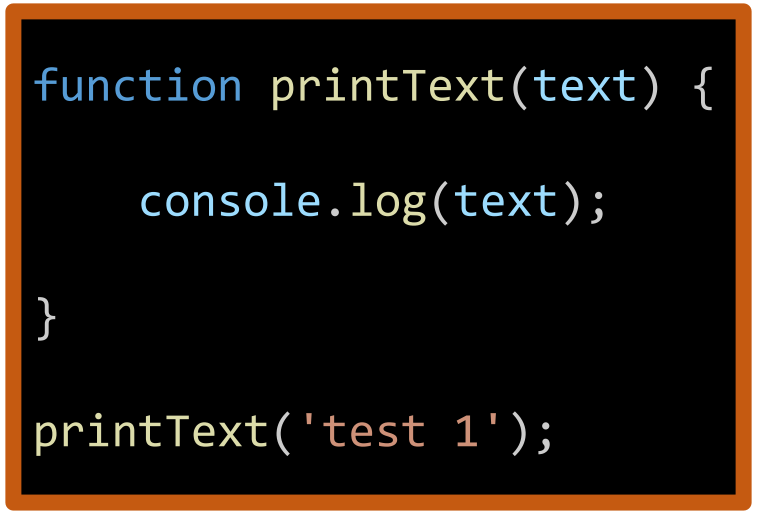
Setting arguments

- You can create a function that takes arguments in this way
- This is done by putting a variable name inside the brackets when the function is defined

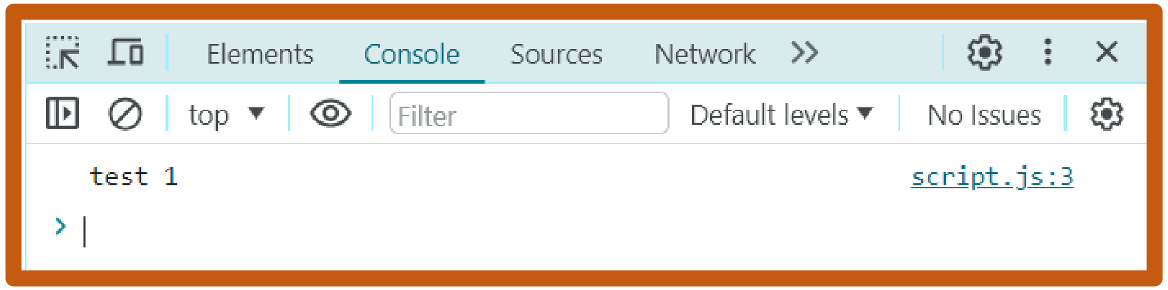
```
function printText(text) {

    console.log(text);

}
```

Using arguments

- When the function is called, the argument must be provided (like with alert/querySelector/etc)
- When printText is run, the variable text will be set to 'test 1'



```
function printText(text) {
    console.log(text);
}
printText('test 1');
```

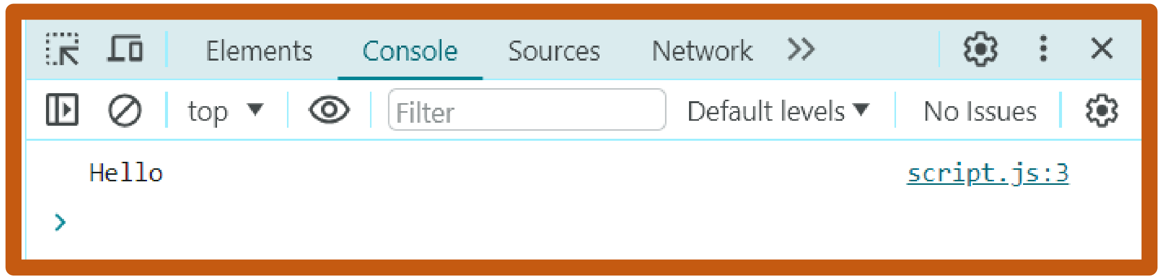


```
test 1                                   script.js:3
```

Arguments example

- The text will now be set to ‘Hello’

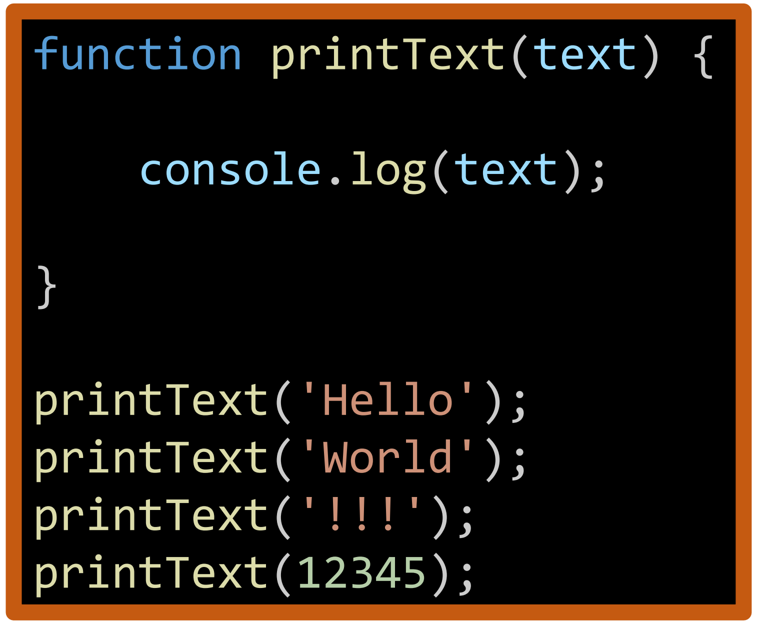
```
function printText(text) {

    console.log(text);

}

printText('Hello');
```



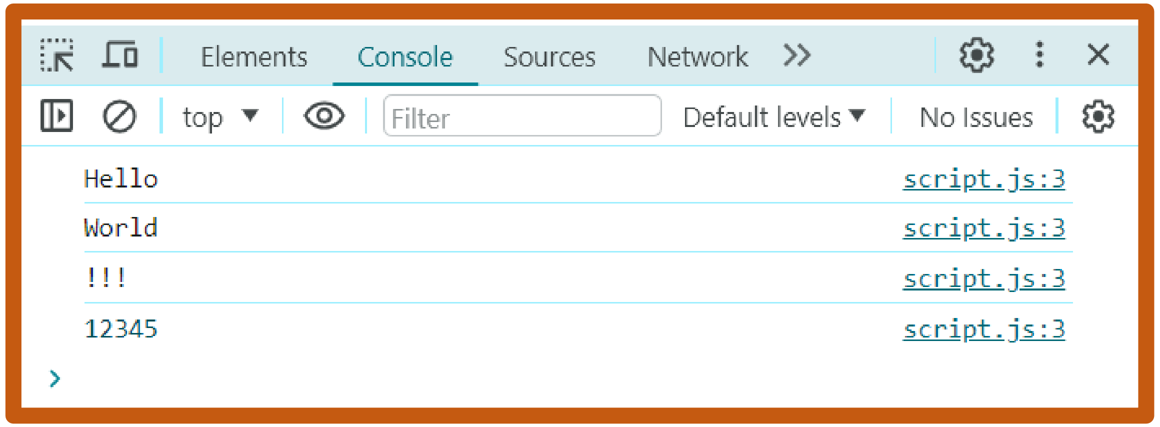
```
Hello                                          script.js:3
>
```

Different arguments

- Functions can be called multiple times with different argument values



```
function printText(text) {

    console.log(text);

}

printText('Hello');
printText('World');
printText('!!!');
printText(12345);
```

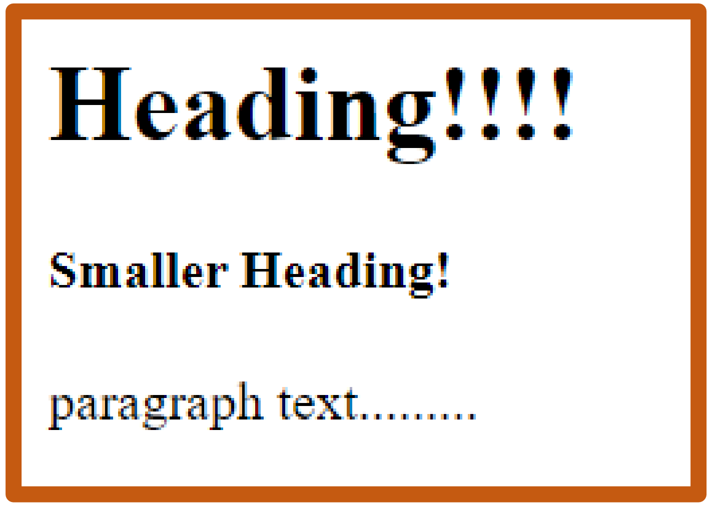


```
Hello                                      script.js:3
World                                      script.js:3
!!!                                        script.js:3
12345                                      script.js:3
>
```

Multiple arguments

- You can create a function that takes multiple arguments by separating them with a comma

```
function createElement(tag, text) {
    const element = document.createElement(tag);
    const textNode = document.createTextNode(text);

    element.appendChild(textNode);
    document.body.appendChild(element);
}

createElement('h1', 'Heading!!!!');
createElement('h4', 'Smaller Heading!');
createElement('p', 'paragraph text.........');
```



```
Heading!!!!

Smaller Heading!

paragraph text.........
```

Return

- As well as arguments, functions can return a value back to the place they were called
- You have already seen this behaviour with the inbuilt functions

```
const element = document.querySelector('p');

const enemies = document.querySelectorAll('.enemy');

const randomNumber = Math.random();
```

Return values

- Each of these functions performs a task and then sends a value back to where it was called

```
const randomNumber = Math.random();
```

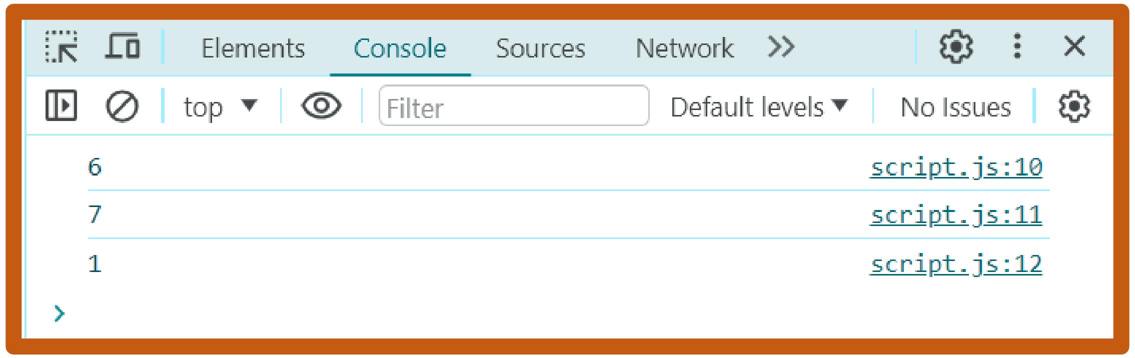
- The random function generates a random number then sends that number back to the place it was called
- The value sent back (the “return value”) can then be stored inside a variable
- The same thing happens with other functions

```
const element = document.querySelector('p');
const enemies = document.querySelectorAll('.enemy');
```

Adding a return

- Your own functions can also return values
- This can be use to reduce repeated code and simplify the calling code
- To return a value use the return keyword followed by a value which will be returned

```
function randomNumberGenerator() {
    let number = Math.ceil(Math.random() * 10);
    return number;
}

const random1 = randomNumberGenerator();
const random2 = randomNumberGenerator();
const random3 = randomNumberGenerator();
```



```
6                                                    script.js:10
7                                                    script.js:11
1                                                    script.js:12
>
```

Arguments & returns

- Return values can be combined with arguments

```
function randomNumberGenerator(max) {
    let number = Math.ceil(Math.random() * max);
    return number;
}

//Generate a number between 1 and 10
const random1 = randomNumberGenerator(10);

//Generate a number between 1 and 50
const random2 = randomNumberGenerator(50);

//Generate a number between 1 and 100
const random3 = randomNumberGenerator(100);
```

Exercise 2

- Download ex2.zip
- It contains the HTML and CSS for a to do list
- Add a click event to the Add Task button
- Create a function that returns the value the user has entered in the text box
- Try creating another function that accepts the string as an argument, inside this function add the code from last week to add a new <li> to the <ul> (remember to add the delete button too)
- Add the functionality for the delete buttons, they should delete the correct task.

Arrow functions

- Another function you can use is an arrow function
- These are a much more concise way to write functions in JavaScript

```
const multiply = (a, b) => a * b;

console.log(multiply(10, 2));
```

- Arrow functions use => and automatically return the value without needing the return keyword.
  - Multiply 10 and 2 then return the result

Arrow functions comparison

- Arrow functions allow us to create a function and return a value with significantly less code
  - You can pass in arguments (a, b)

```
function multiply(a, b) {
    const total = a * b;
    return total;
}
const total = multiply(10, 2);

console.log(total);
```

```
const multiply = (a, b) => a * b;
const total = multiply(10,2);

console.log(total);
```

Arrow functions pt2

- Arrow functions can be used anywhere you would usually call a function

```
const startButton = document.querySelector('.start');

startButton.addEventListener('click', () => startButton.style.backgroundColor = 'red');
```

- Keydown and keyup could be implemented using an arrow function

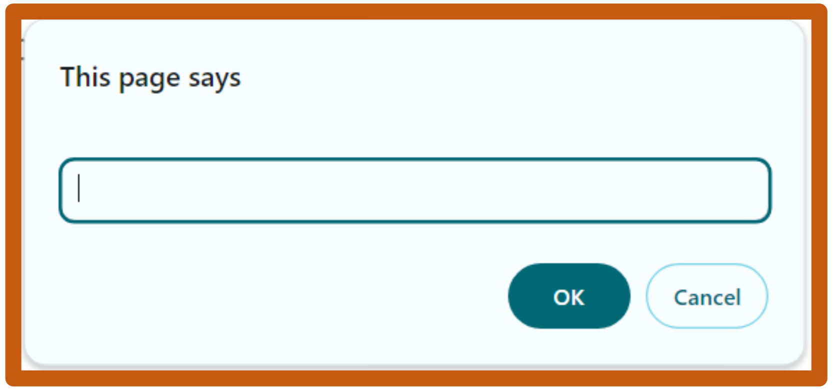
```
document.addEventListener('keydown', (event) => {
    if (event.key === 'ArrowUp') {
        upPressed = true;
    } else if (event.key === 'ArrowDown') {
        downPressed = true;
    } else if (event.key === 'ArrowLeft') {
        leftPressed = true;
    } else if (event.key === 'ArrowRight') {
        rightPressed = true;
    }
});
```

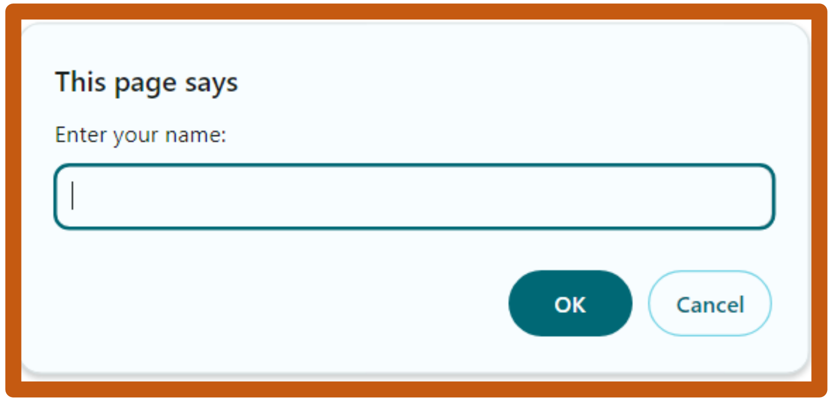
```
document.addEventListener('keyup', (event) => {
    if (event.key === 'ArrowUp') {
        upPressed = false;
    } else if (event.key === 'ArrowDown') {
        downPressed = false;
    } else if (event.key === 'ArrowLeft') {
        leftPressed = false;
    } else if (event.key === 'ArrowRight') {
        rightPressed = false;
    }
});
```

Prompt

- You can use prompt() to create a dialog box the user can enter text into

```
prompt();

prompt('Enter your name:');
```


This page says

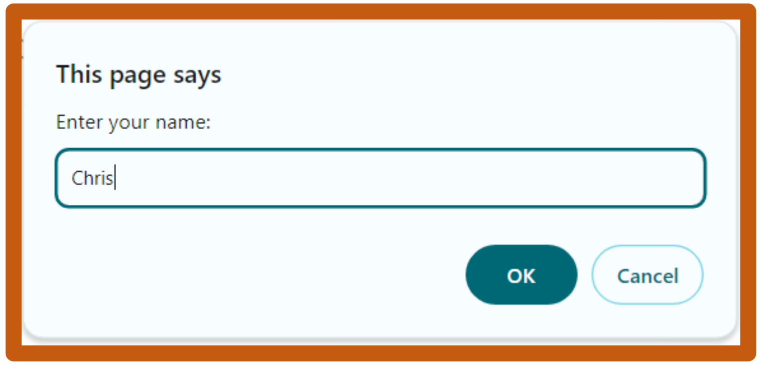

This page says

Enter your name:

Prompt pt2

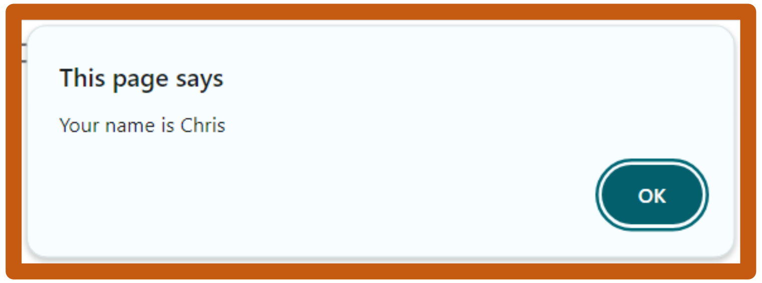
- You can save the input of a prompt by creating a variable and storing the return of prompt
  - The value the user entered will then be displayed in the alert()

```
const myName = prompt('Enter your name:');

alert('Your name is ' + myName);
```


This page says

Enter your name:

Chris

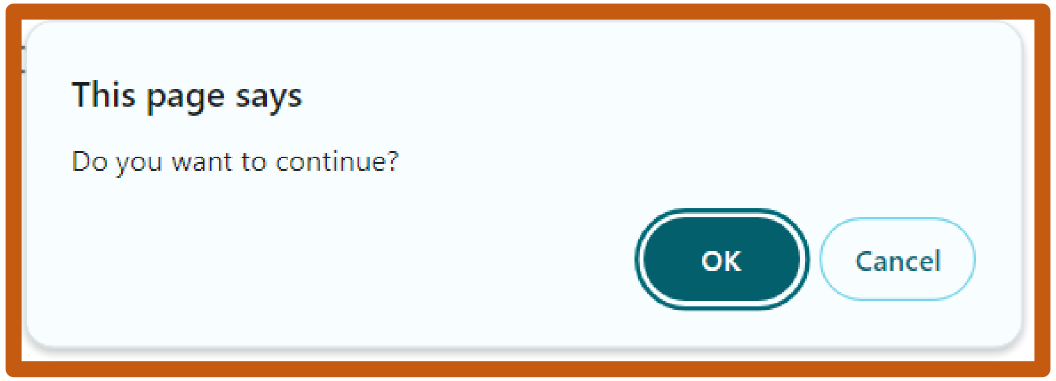

This page says

Your name is Chris

Confirm

- Confirm() is like alert however you can store whether the user clicked “Ok” or “Cancel”
  - Boolean is returned (true/false)

```
const check = confirm('Do you want to continue?');

if(check == true) {
    alert('User clicked continue');
}
else {
    alert('User did not continue');
}
```


This page says

Do you want to continue?

Local storage - setItem

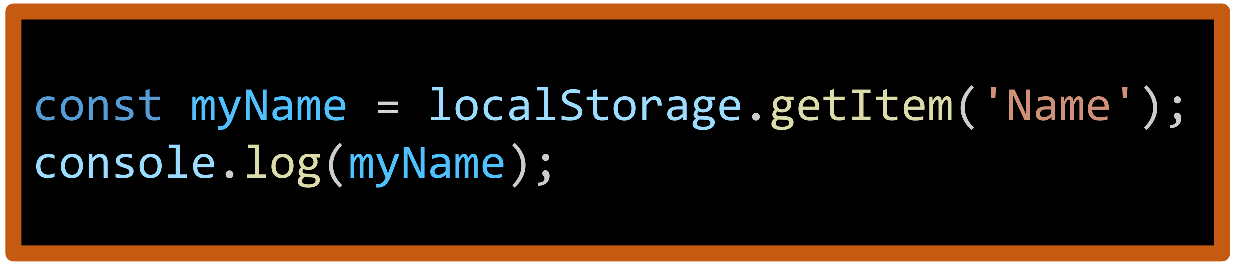
- Local storage allows you to save values in the browser
- Even if you close the browser those values will remain

- To add values to local storage you need to use the setItem() function
  - You need a key (name) for the data
  - Value to be saved

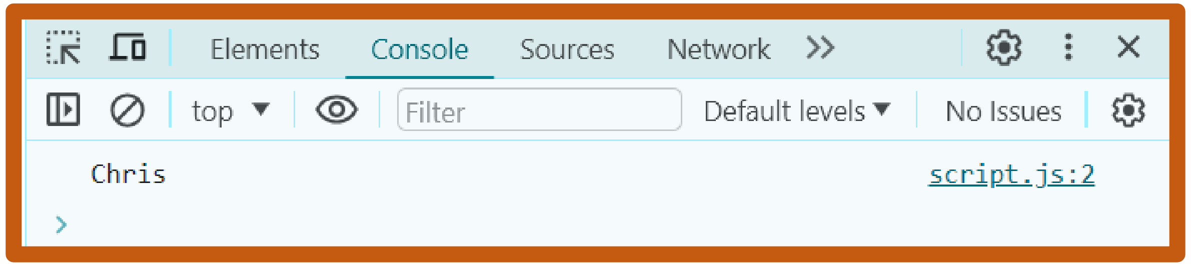
```
localStorage.setItem('Name', 'Chris');
```

Local storage - getItem

- To retrieve something from local storage use the getItem() function and the key (name) of the value you want to retrieve



```
const myName = localStorage.getItem('Name');
console.log(myName);
```

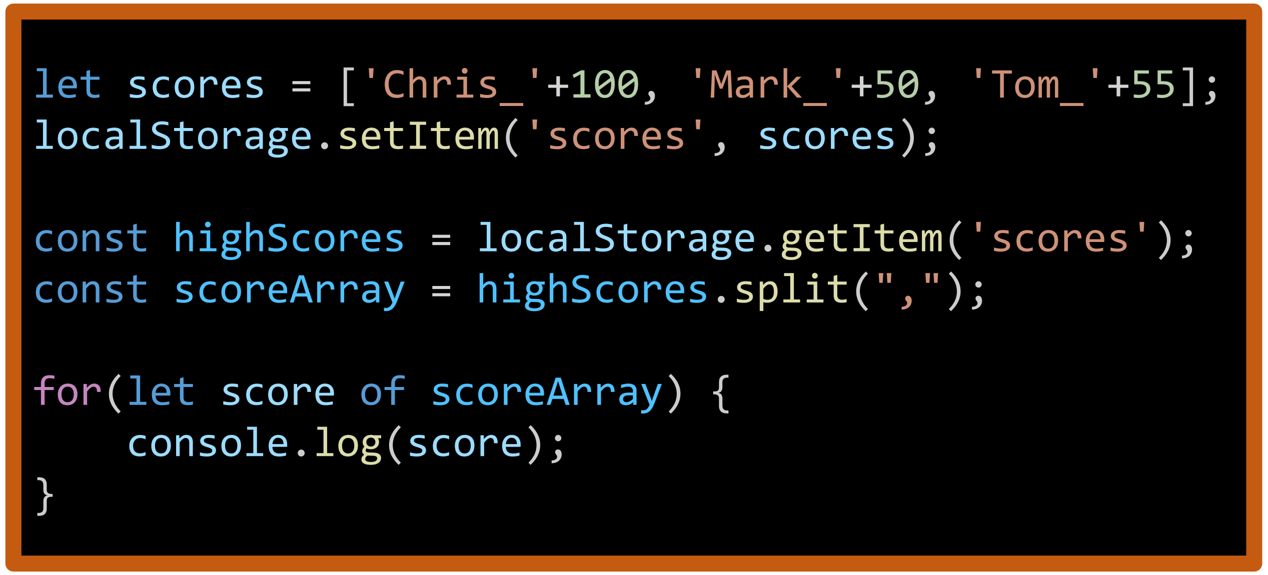


```
Chris                                                    script.js:2
>
```

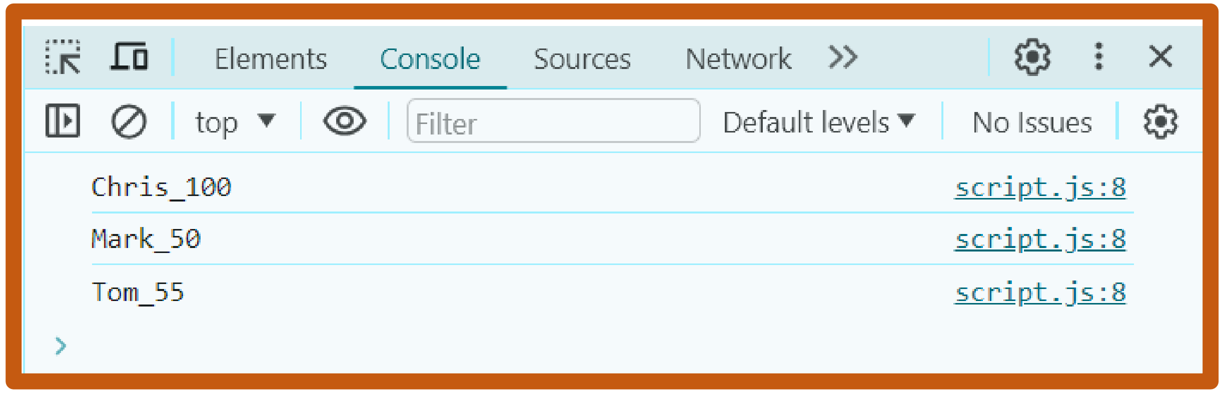
- Even if the browser is closed “Chris” could still be retrieved from local storage

Local storage – saving an array

- When you save something to local storage it will save as a string
- This is a problem if you want to save an array
- You could save the array to local storage and then retrieve it as a string, using the split() function then separate it back into an array



```
let scores = ['Chris_'+100, 'Mark_'+50, 'Tom_'+55];
localStorage.setItem('scores', scores);

const highScores = localStorage.getItem('scores');
const scoreArray = highScores.split(",");

for(let score of scoreArray) {
    console.log(score);
}
```



```
Chris_100                                         script.js:8
Mark_50                                           script.js:8
Tom_55                                            script.js:8
```

Local storage - removeItem

- To remove something from local storage use the removeItem() function
  - You will need the key (name) of the item to remove

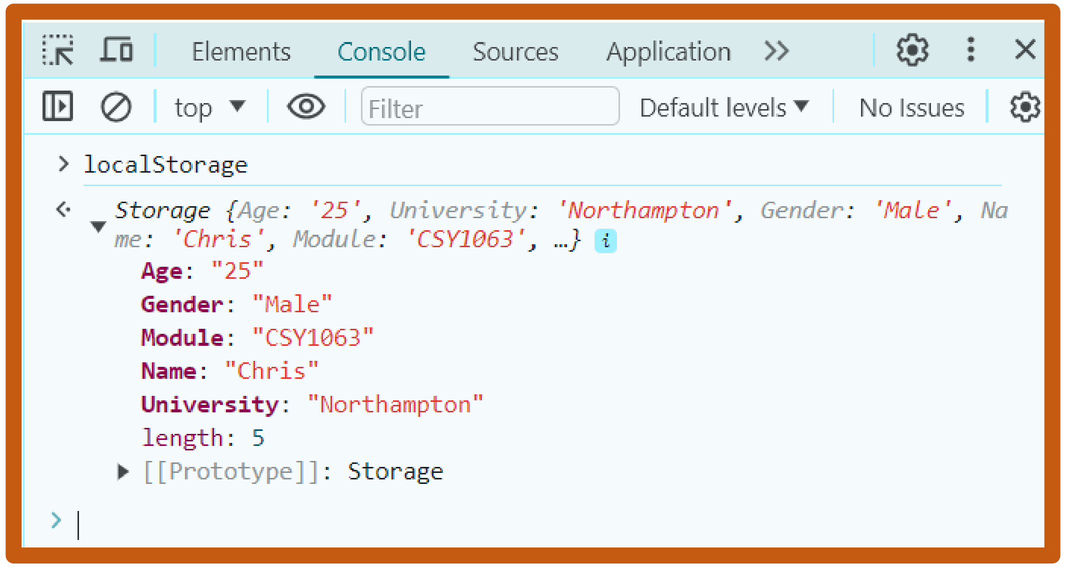
```
localStorage.removeItem('Name');
```

- To completely clear all items from local storage use the .clear() function

```
localStorage.clear();
```

Local storage - see all items

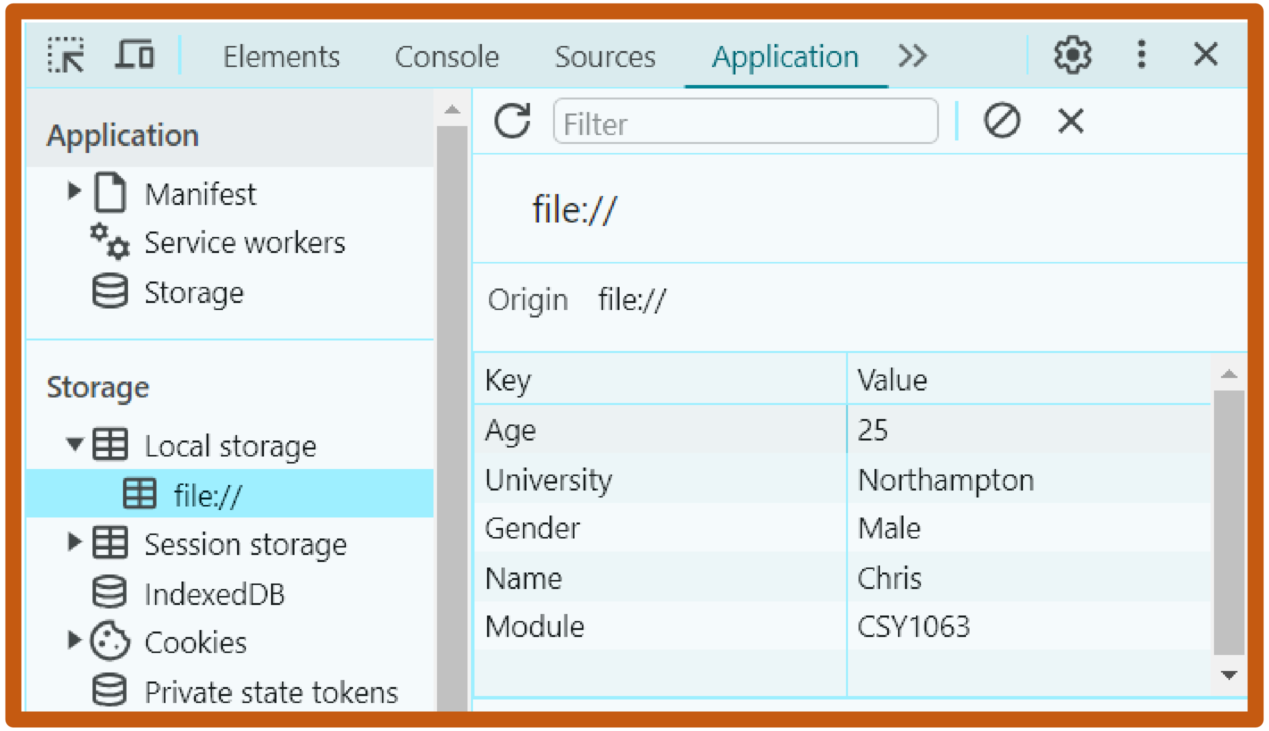
- To see everything in local storage you can use developer tools
- You can either type localStorage into the console



```
> localStorage
<  Storage {Age: '25', University: 'Northampton', Gender: 'Male', Na
   me: 'Chris', Module: 'CSY1063', …}
     Age: "25"
     Gender: "Male"
     Module: "CSY1063"
     Name: "Chris"
     University: "Northampton"
     length: 5
   ▸ [[Prototype]]: Storage
>
```

Local storage - see all items pt2

- Go to Application and then Local storage under Storage



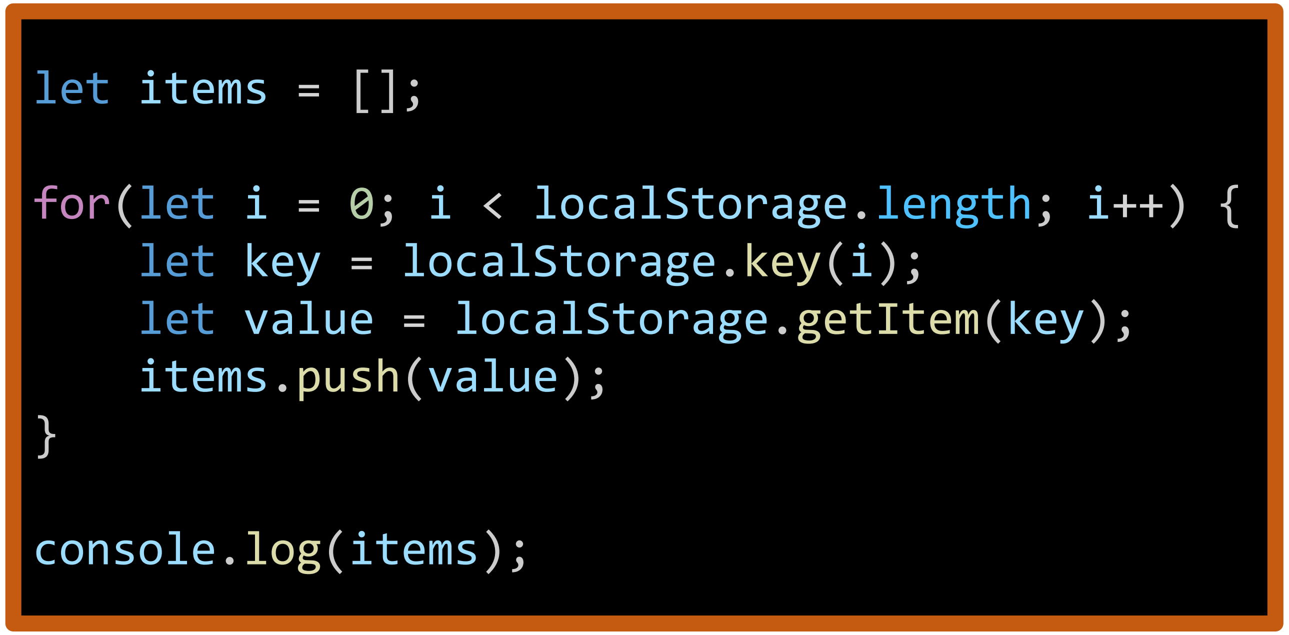
file://

Origin file://

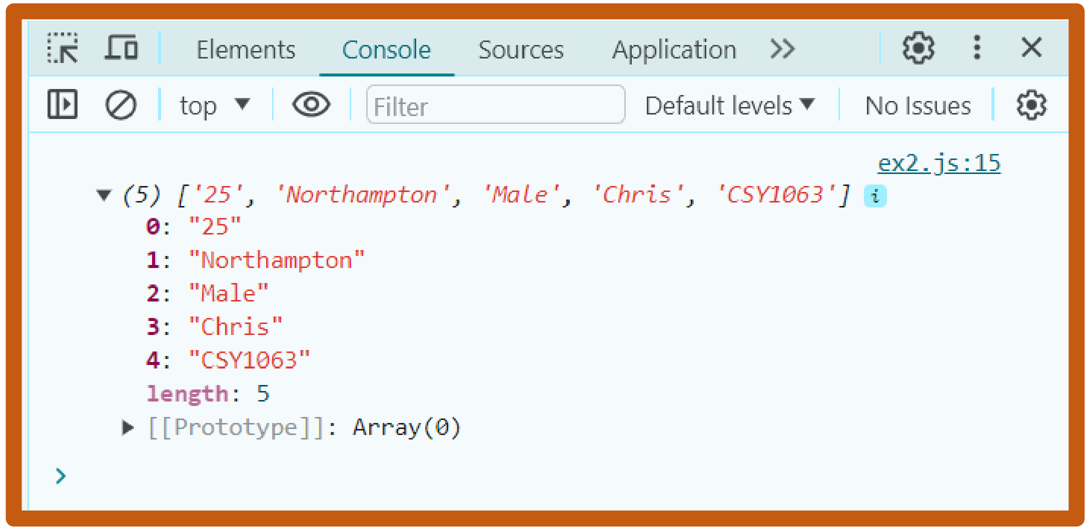
Key	Value
Age	25
University	Northampton
Gender	Male
Name	Chris
Module	CSY1063

Local storage - get all items

- You can use a for loop to get every item saved in local storage, this will store it in an array



```
let items = [];

for(let i = 0; i < localStorage.length; i++) {
    let key = localStorage.key(i);
    let value = localStorage.getItem(key);
    items.push(value);
}

console.log(items);
```



```
                                                              ex2.js:15
▼ (5) ['25', 'Northampton', 'Male', 'Chris', 'CSY1063']
    0: "25"
    1: "Northampton"
    2: "Male"
    3: "Chris"
    4: "CSY1063"
    length: 5
  ▶ [[Prototype]]: Array(0)
>
```

Math.max

- The max() method on Math will return the largest number in a set of numbers

```
const largestNumber = Math.max(10, 100, 50, 20);
console.log(largestNumber);
```

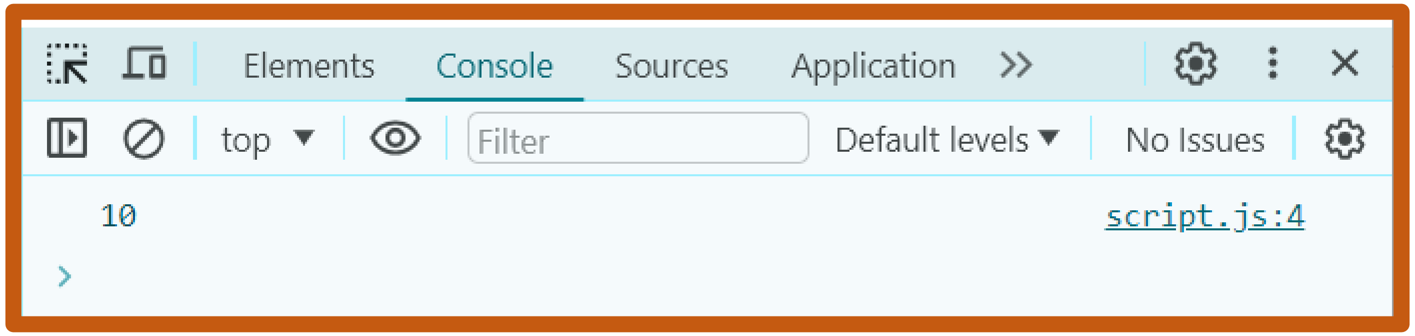
- To use max() on an array you need to use the spread operator (...)
  - Allows you to expand iterable objects like arrays into individual elements

```
const numbers = [10, 100, 20, 50];

const largestNumber = Math.max(...numbers);
console.log(largestNumber);
```

Math.min

- min() can be used to return the smallest in a set of numbers

```
const numbers = [10, 100, 20, 50];

const smallestNumber = Math.min(...numbers);
console.log(smallestNumber);
```



```
10                                                   script.js:4
>
```

Exercise 3

- Amend exercise 2 to save the tasks in local storage and retrieve them
- If you close the browser or click refresh all the tasks should still be there

- Add a clear button and add the code to clear local storage

Arrays methods

- We looked at arrays last lecture
- Arrays let you store more than one value in a single variable

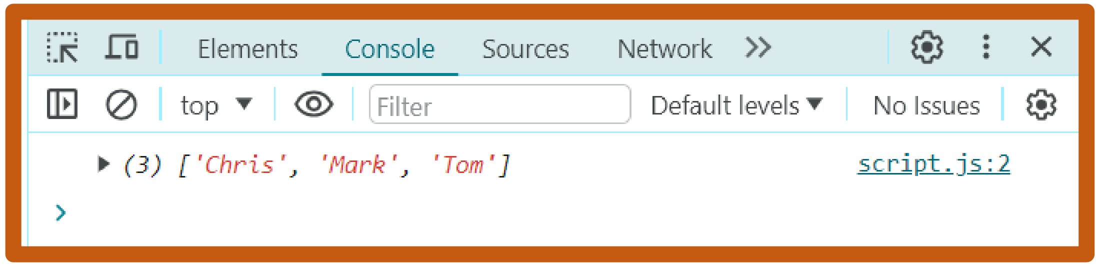
```
let myArray = [];
myArray[0] = 'Red';
myArray[1] = 'Green';
myArray[2] = 'Blue';
```

- There are a variety of methods available when using arrays that can make using them easier
  - Removing/adding elements
  - Sorting an array
  - Etc.

Array - push

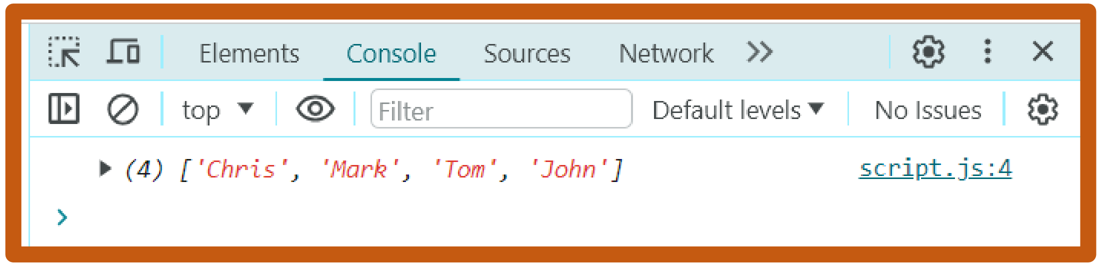
- The push() method of an array adds an element to the end of an array
  - Useful for adding to an array

```
const names = ['Chris', 'Mark', 'Tom'];
console.log(names);
```



```
▸ (3) ['Chris', 'Mark', 'Tom']                script.js:2
>
```

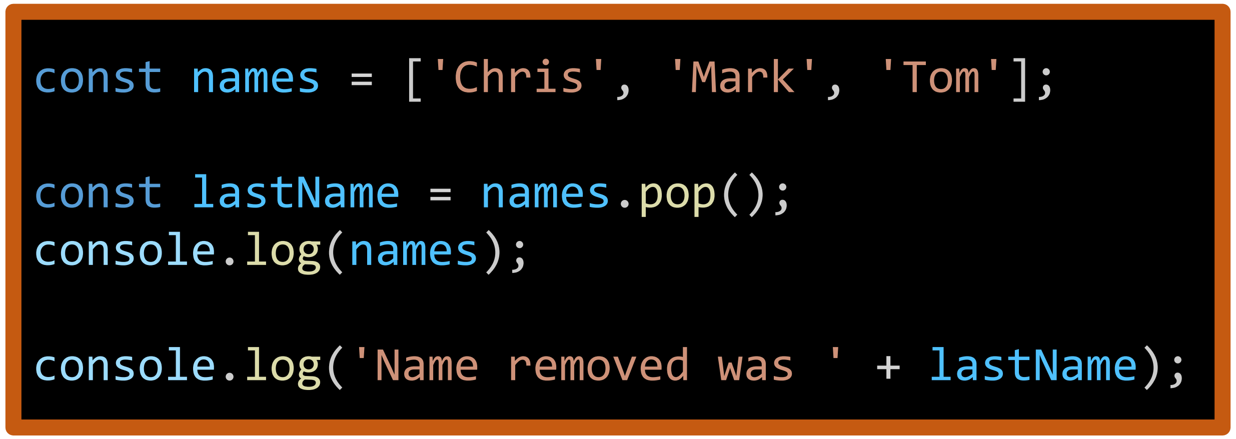
```
const names = ['Chris', 'Mark', 'Tom'];

names.push('John');
console.log(names);
```



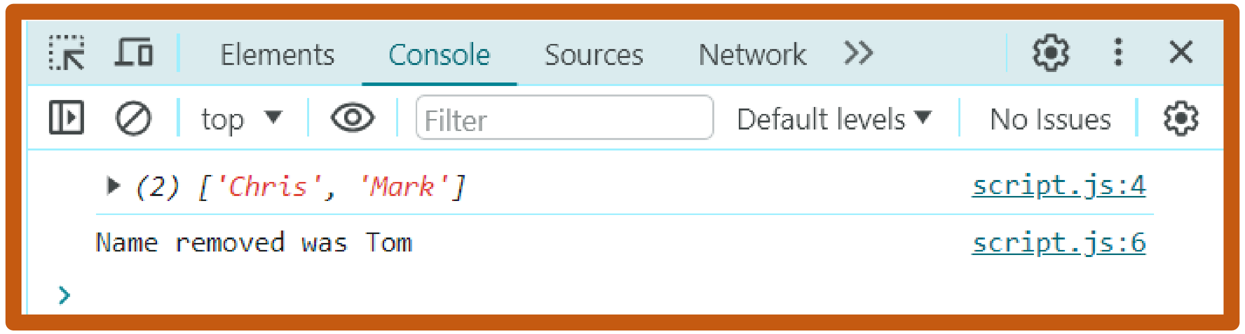
```
▸ (4) ['Chris', 'Mark', 'Tom', 'John']        script.js:4
>
```

Array - pop

- pop() removes the last element of an array and returns that element
  - Tom was removed from the array and saved in the lastName variable



```
const names = ['Chris', 'Mark', 'Tom'];

const lastName = names.pop();
console.log(names);

console.log('Name removed was ' + lastName);
```

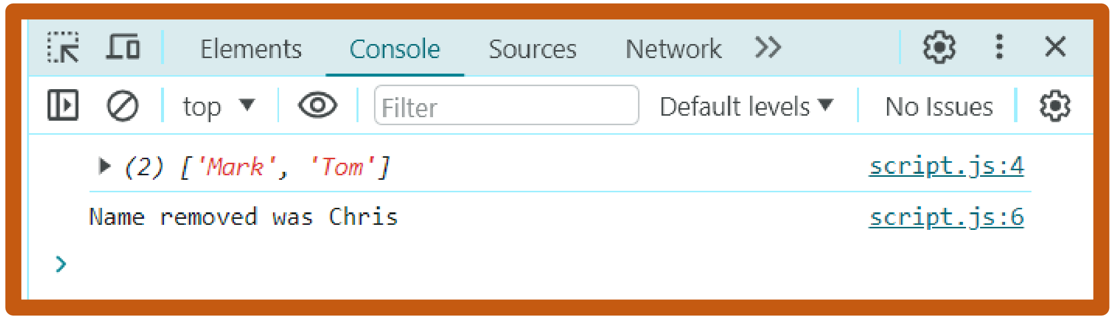


```
▶ (2) ['Chris', 'Mark']                                   script.js:4
Name removed was Tom                                      script.js:6
>
```

Array - shift

- The shift method is similar to pop except it is for the first element instead of the last

```
const names = ['Chris', 'Mark', 'Tom'];

const firstName = names.shift();
console.log(names);

console.log('Name removed was ' + firstName);
```

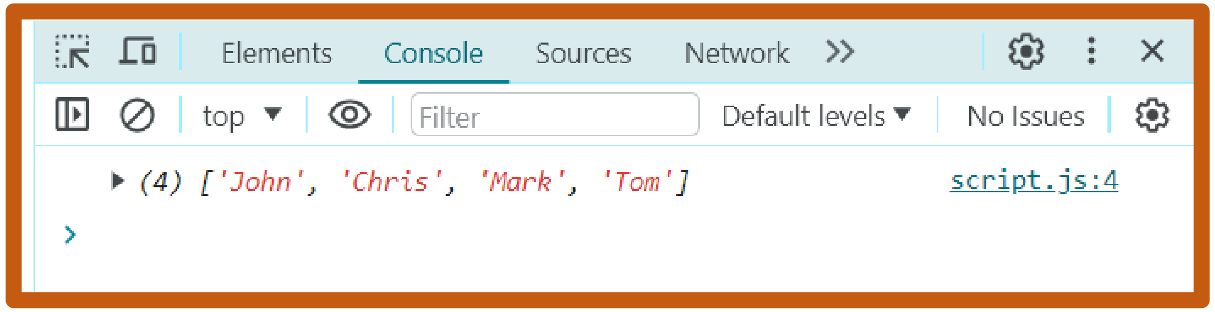


```
▶ (2) ['Mark', 'Tom']          script.js:4
Name removed was Chris         script.js:6
```

Array - unshift

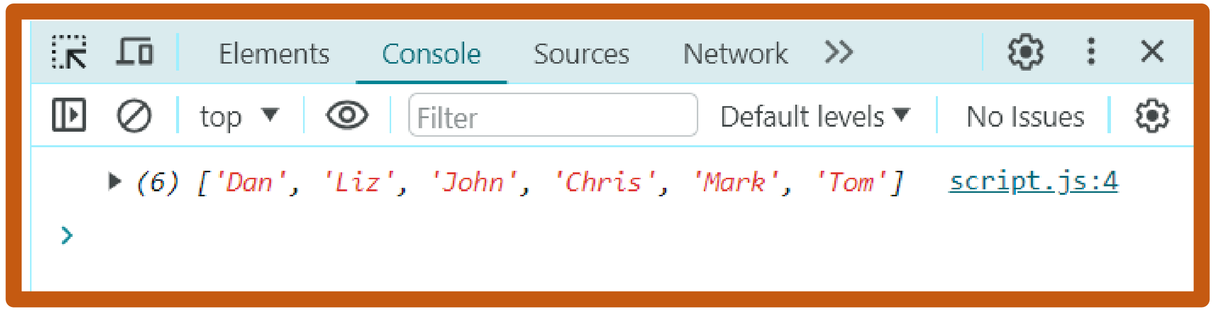
- unshift() inserts a value to the beginning of an array
  - Multiple values can be added to the beginning of an array using a comma to sperate them

```
const names = ['Chris', 'Mark', 'Tom'];

names.unshift('John');
console.log(names);
```



```
▸ (4) ['John', 'Chris', 'Mark', 'Tom']                script.js:4
>
```

```
const names = ['Chris', 'Mark', 'Tom'];

names.unshift('Dan', 'Liz', 'John');
console.log(names);
```



```
▸ (6) ['Dan', 'Liz', 'John', 'Chris', 'Mark', 'Tom']  script.js:4
>
```

Array - splice

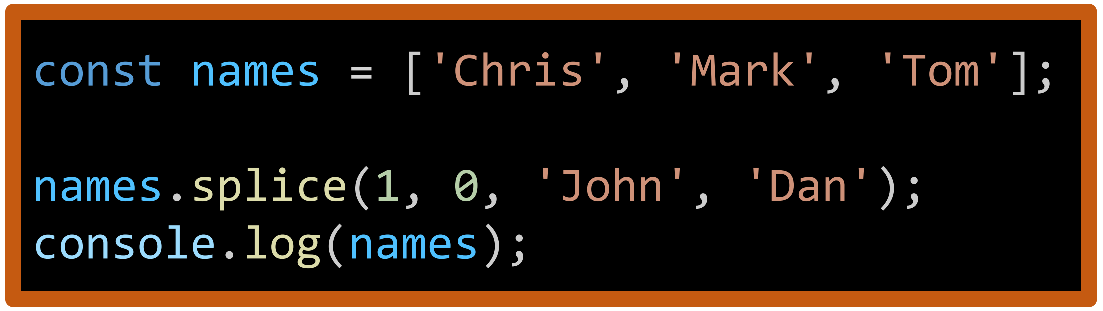
- Splice can be used to add or remove elements of an array
- The syntax for splice

```
names.splice(
    /* Index position (where to add/remove) */ ,
    /* How many to remove (optional) */ ,
    /* New elements to be added (optional) */
);
```

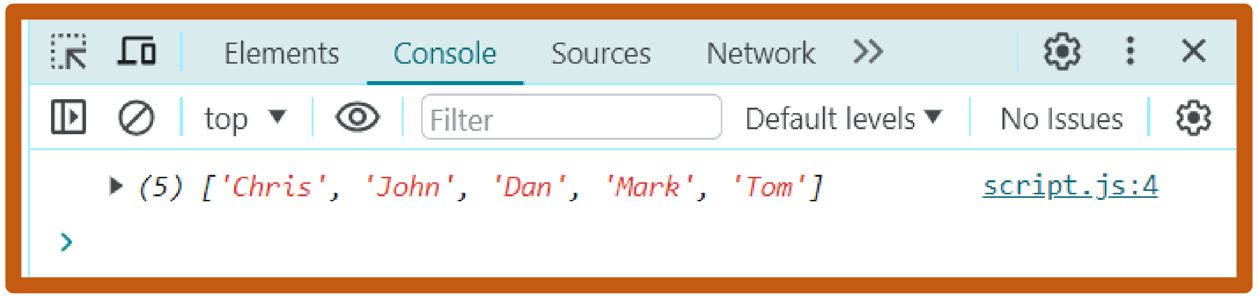
- Splice will overwrite the original array

Array - splice (add)

- You can add new elements to a specific position in an array using splice
  - At index position 1 add “John” and “Dan”



```
const names = ['Chris', 'Mark', 'Tom'];

names.splice(1, 0, 'John', 'Dan');
console.log(names);
```

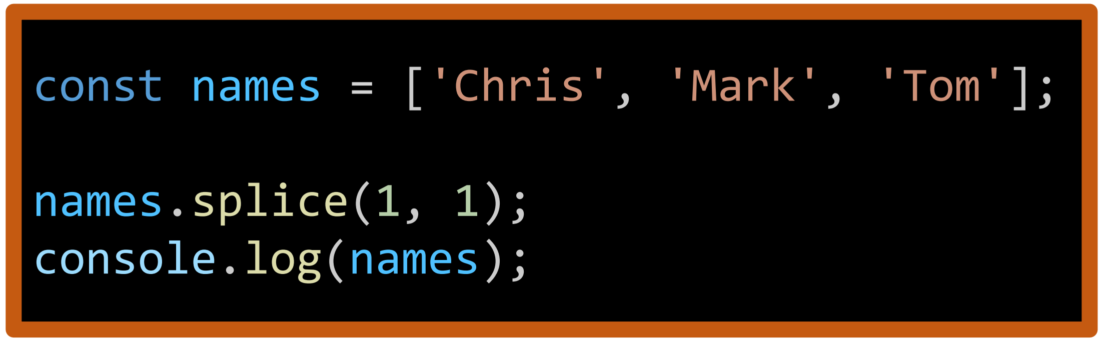


```
▶ (5) ['Chris', 'John', 'Dan', 'Mark', 'Tom']          script.js:4
>
```

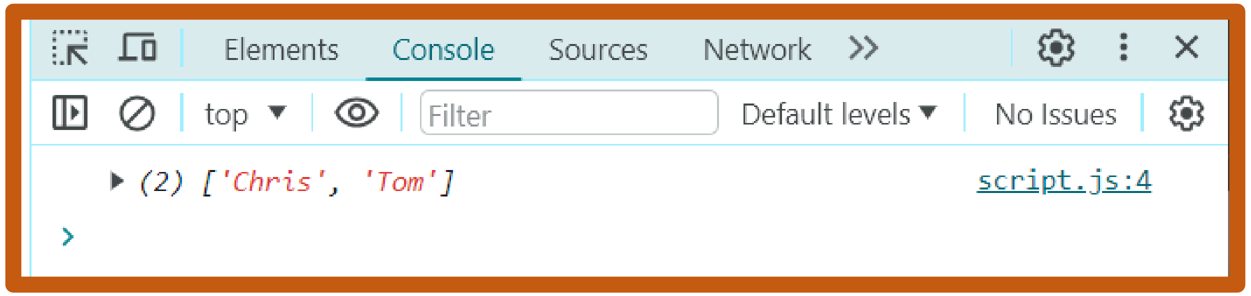
- Multiple elements can be added to the array using splice

Array - splice (remove)

- Splice can be used to remove elements at a specific index position
  - At index position 1 remove 1 element
  - ‘Mark’ was removed from the array



```
const names = ['Chris', 'Mark', 'Tom'];

names.splice(1, 1);
console.log(names);
```



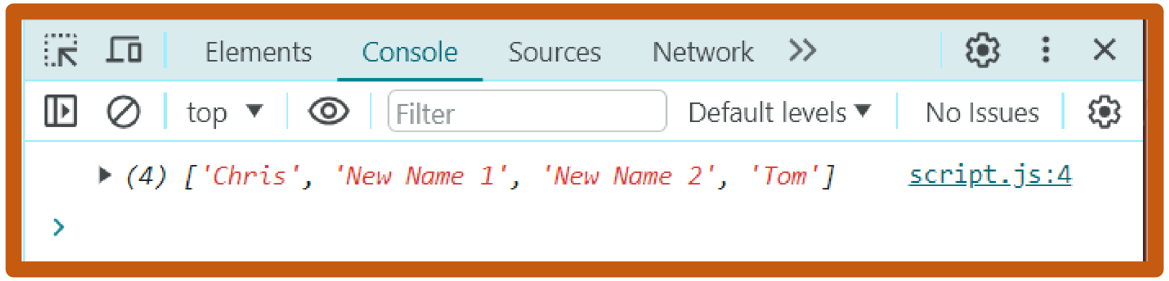
```
▸ (2) ['Chris', 'Tom']                    script.js:4
>
```

- Multiple elements can be removed at a time using splice
  - names.splice(1,2) would remove ‘Mark’ and ‘Tom’

Array - splice (remove & add)

- Splice can be used to remove and add a new element to a specific position in an array
  - At index position 1 remove 1 element and add ‘New name 1’, ‘New name 2’
  - Replace ‘Mark’ with ‘New name 1’ and ‘New name 2’

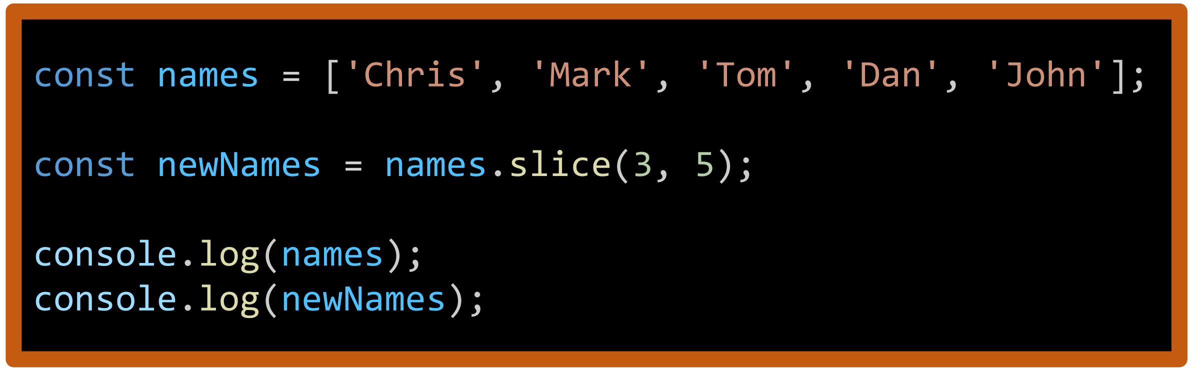
```
const names = ['Chris', 'Mark', 'Tom'];

names.splice(1, 1, 'New Name 1', 'New Name 2');
console.log(names);
```



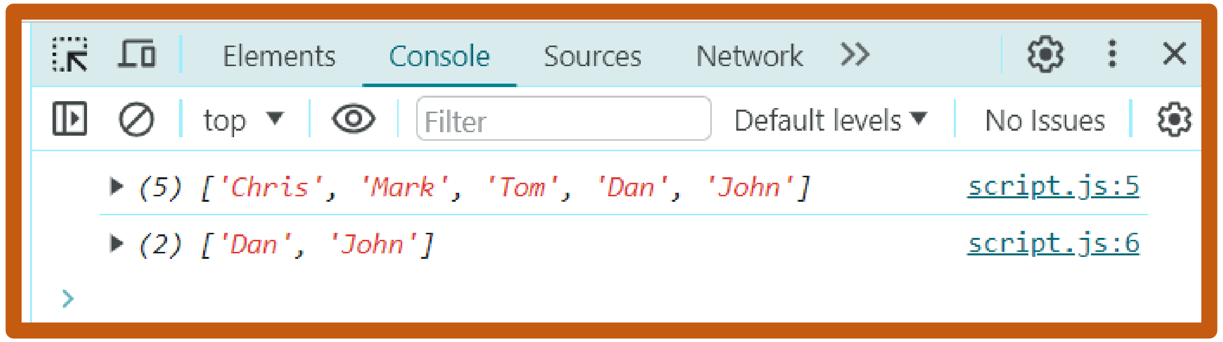
```
▸ (4) ['Chris', 'New Name 1', 'New Name 2', 'Tom']     script.js:4
>
```

Array - slice

- The slice() method is used to return selected elements in a new array
  - names.slice(start index, end index)
  - Select from ‘Dan’ to ‘John’ from names



```
const names = ['Chris', 'Mark', 'Tom', 'Dan', 'John'];

const newNames = names.slice(3, 5);

console.log(names);
console.log(newNames);
```

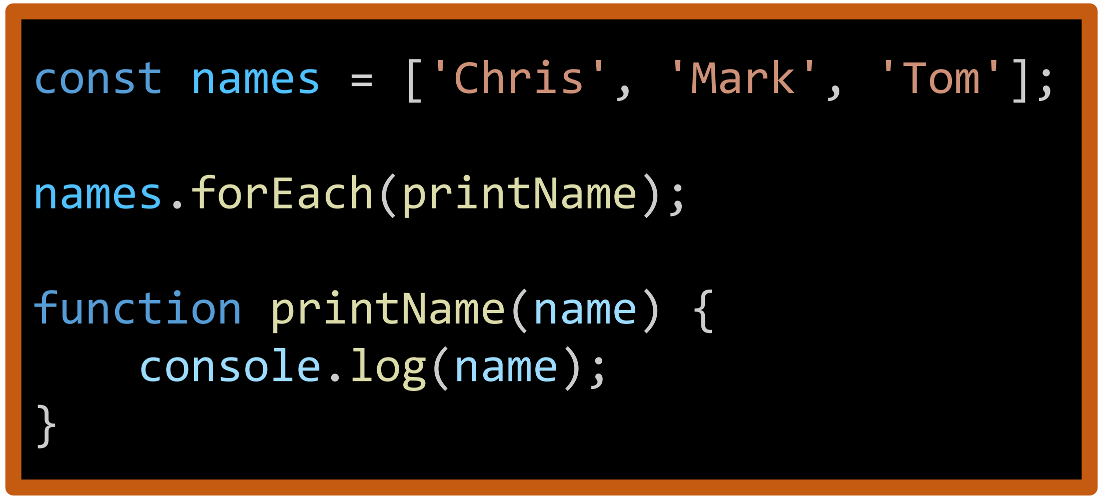


```
▸ (5) ['Chris', 'Mark', 'Tom', 'Dan', 'John']     script.js:5
▸ (2) ['Dan', 'John']                             script.js:6
>
```

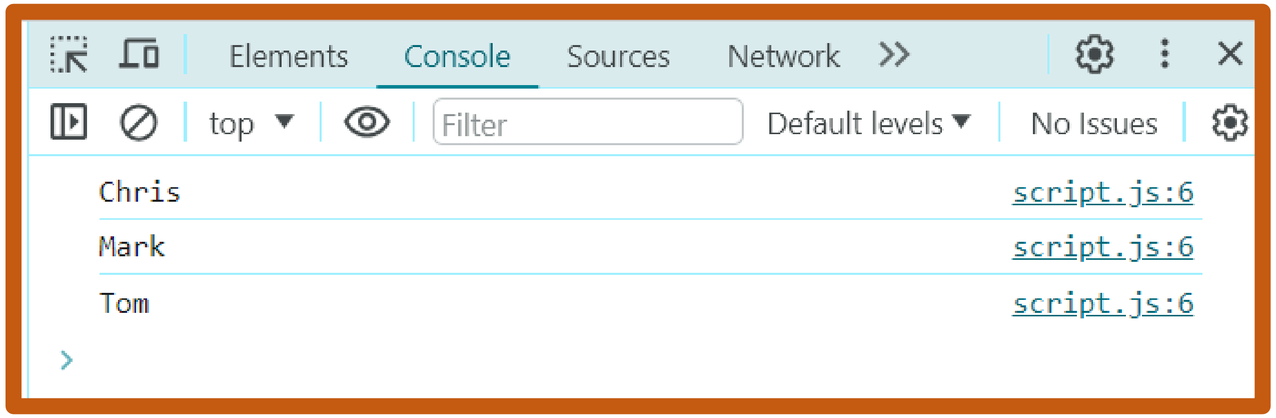
- slice() does not change the original array

Array - forEach

- forEach() calls a function for each element in an array, another way of looping through an array
  - The name variable becomes each element of the array



```
const names = ['Chris', 'Mark', 'Tom'];

names.forEach(printName);

function printName(name) {
    console.log(name);
}
```

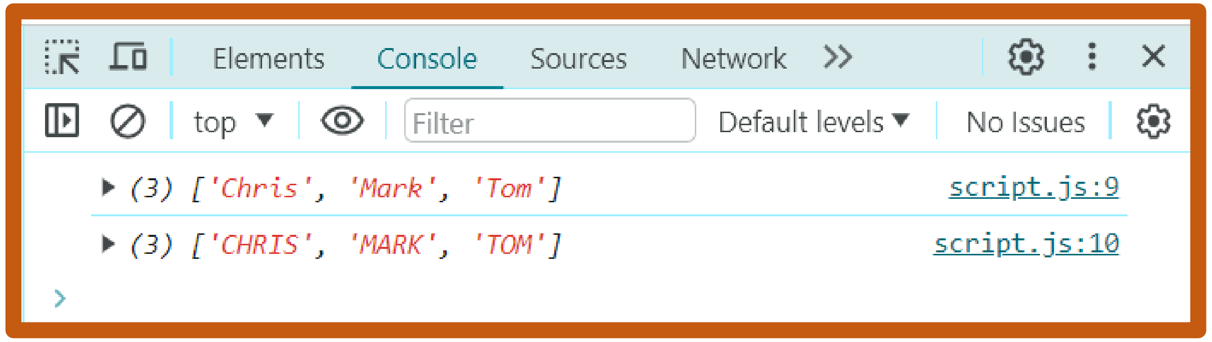


```
Chris                                                  script.js:6
Mark                                                   script.js:6
Tom                                                    script.js:6
>
```

Array - map

- map() is like forEach() except it creates a new array leaving the original array unchanged
  - Create a new array which contains the uppercase values of the names array
  - return and toUpperCase (covered later)

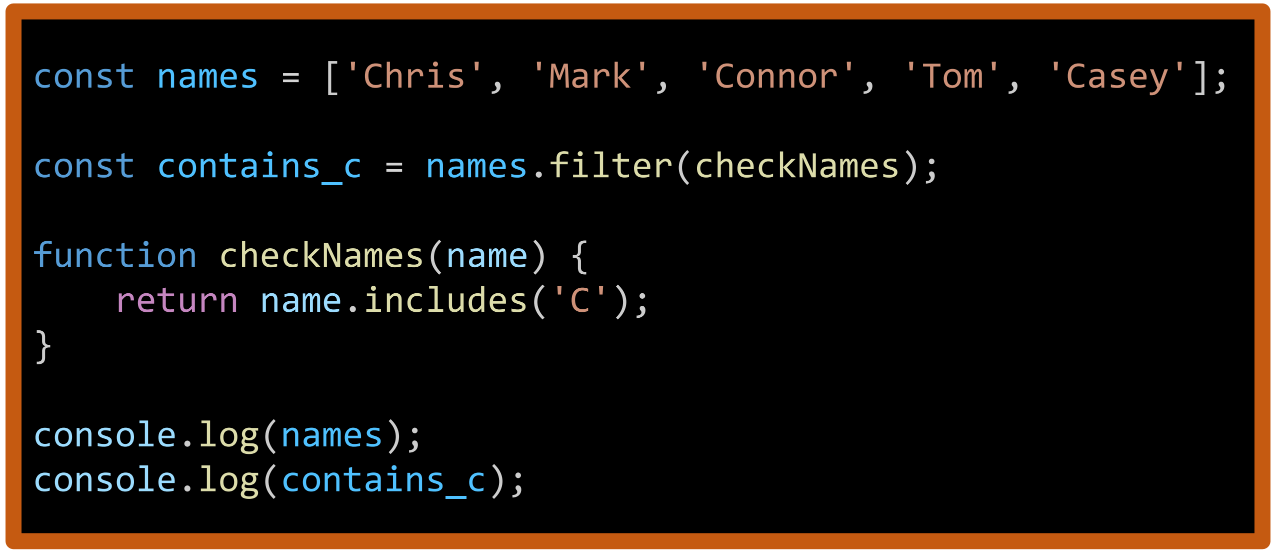
```javascript
const names = ['Chris', 'Mark', 'Tom'];

const upperCaseNames = names.map(capitalizeNames);

function capitalizeNames(name) {
    return name.toUpperCase();
}

console.log(names);
console.log(upperCaseNames);
```



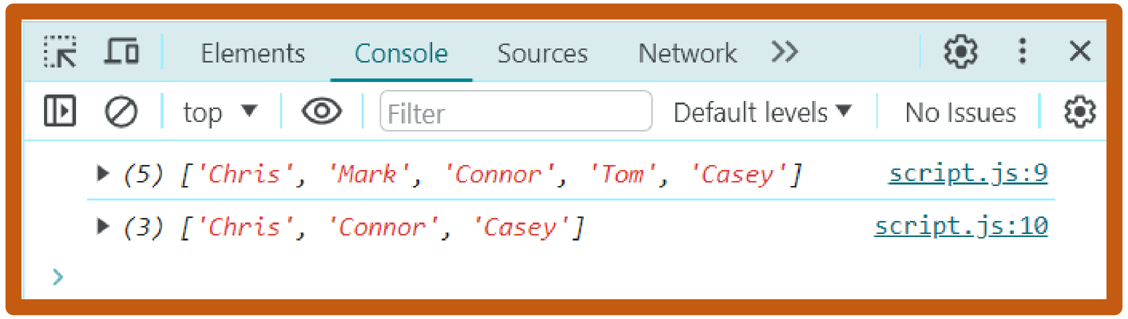
```
▶ (3) ['Chris', 'Mark', 'Tom']        script.js:9
▶ (3) ['CHRIS', 'MARK', 'TOM']        script.js:10
>
```

Array - filter

- filter() is used to create a new array which only contain elements that pass a condition by a function
  - Only names containing a ‘C’ are returned



```
const names = ['Chris', 'Mark', 'Connor', 'Tom', 'Casey'];

const contains_c = names.filter(checkNames);

function checkNames(name) {
    return name.includes('C');
}

console.log(names);
console.log(contains_c);
```

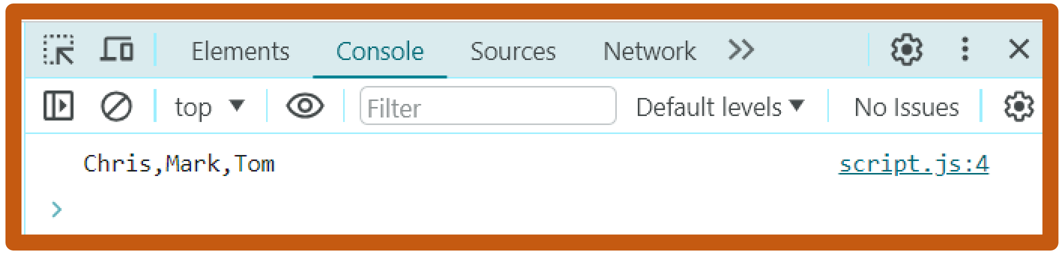


```
▶ (5) ['Chris', 'Mark', 'Connor', 'Tom', 'Casey']        script.js:9
▶ (3) ['Chris', 'Connor', 'Casey']                        script.js:10
>
```

Array - join

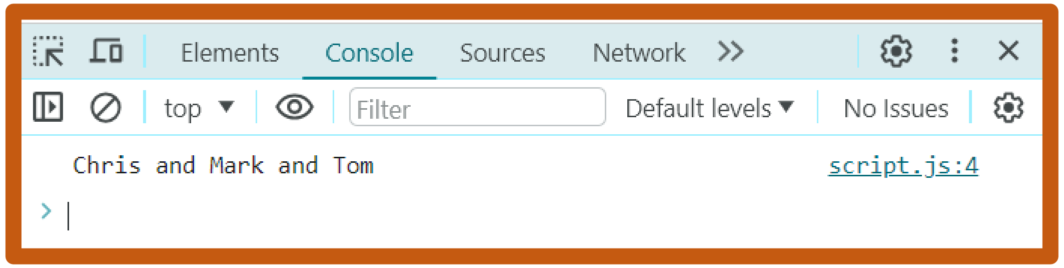
- join() can be used to return an array as a string
  - You can specify what separator is used, the default is a comma

```
const names = ['Chris', 'Mark', 'Tom'];
const stringName = names.join();

console.log(stringName);
```



```
Chris,Mark,Tom                                    script.js:4
```

```
const names = ['Chris', 'Mark', 'Tom'];
const stringName = names.join(' and ');

console.log(stringName);
```

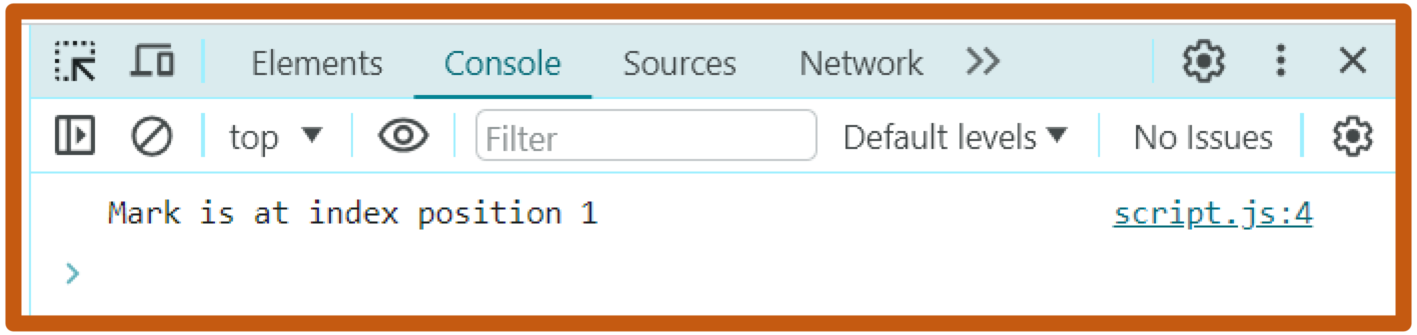


```
Chris and Mark and Tom                            script.js:4
```

Array - indexOf

- You can use indexOf() to find the first index position of a value
  - The first instance of ‘Mark’ is names[1]

```
const names = ['Chris', 'Mark', 'Tom' , 'Mark'];
const indexOfMark = names.indexOf('Mark');

console.log('Mark is at index position ' + indexOfMark);
```

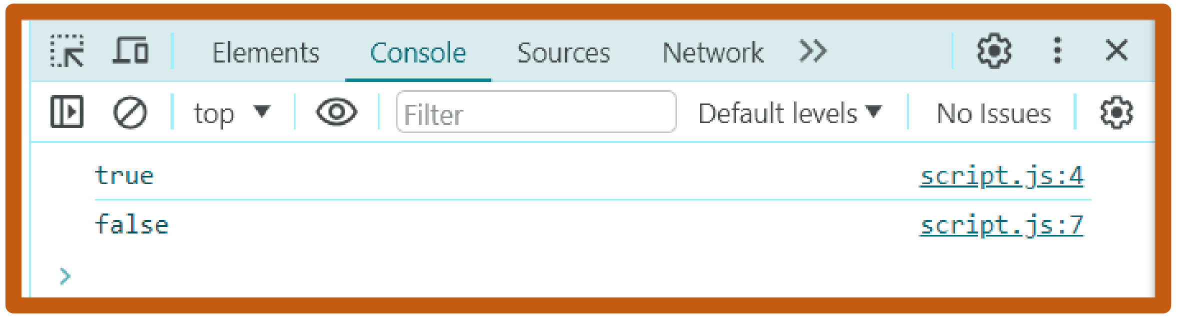


```
Mark is at index position 1
```

Array - includes

- includes() will return true or false is an array contains a specific value

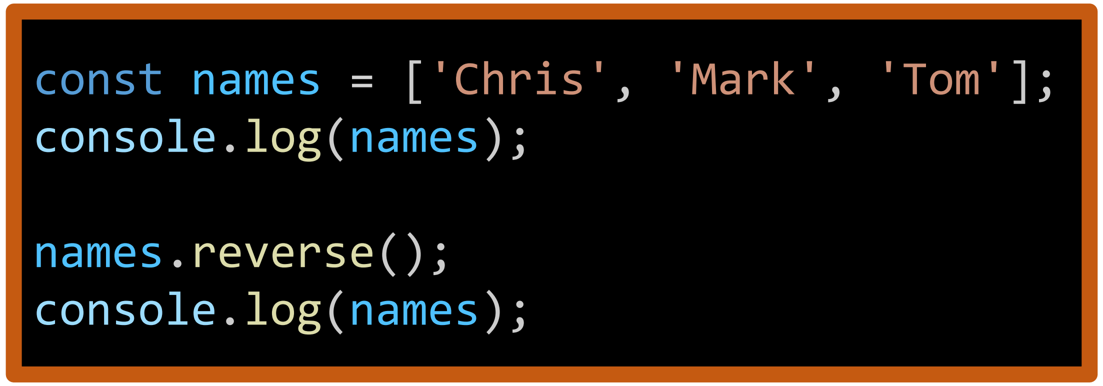
```
const names = ['Chris', 'Mark', 'Tom'];

//Return true
console.log(names.includes('Chris'));

//Return false
console.log(names.includes('John'));
```



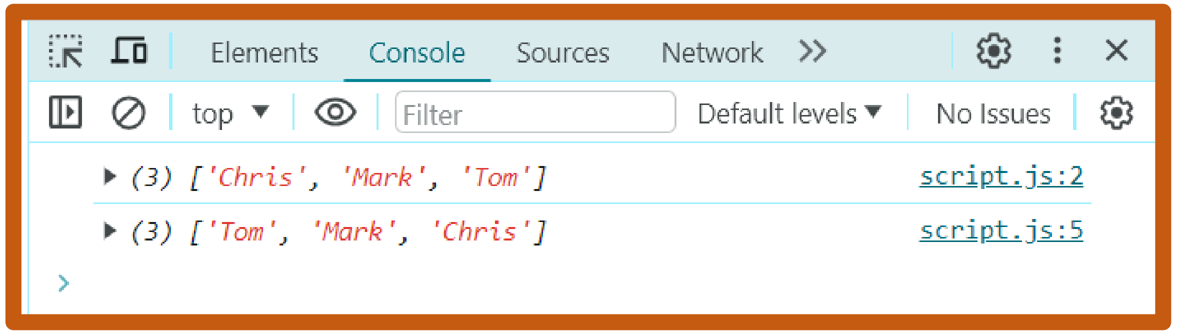
```
true                                                    script.js:4
false                                                   script.js:7
>
```

Array - reverse

- reverse() will reverse the order of elements in an array



```
const names = ['Chris', 'Mark', 'Tom'];
console.log(names);

names.reverse();
console.log(names);
```



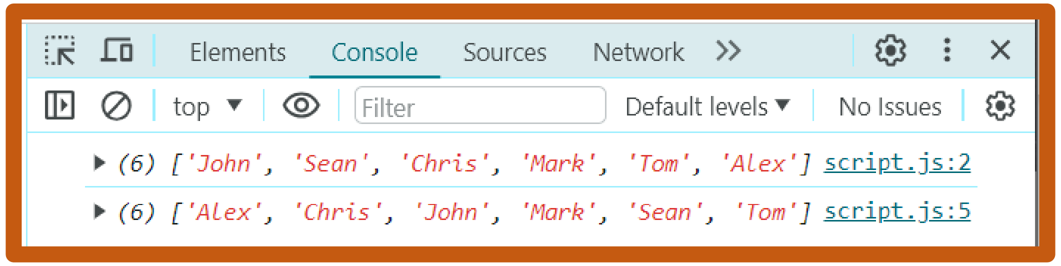
```
▶ (3) ['Chris', 'Mark', 'Tom']          script.js:2
▶ (3) ['Tom', 'Mark', 'Chris']          script.js:5
>
```

- This will overwrite the original array

Array - sort

- sort() can be used to sort an array alphabetically
  - a,b,c,d,e ect

```
const names = ['John', 'Sean', 'Chris', 'Mark', 'Tom', 'Alex'];
console.log(names);

names.sort();
console.log(names);
```



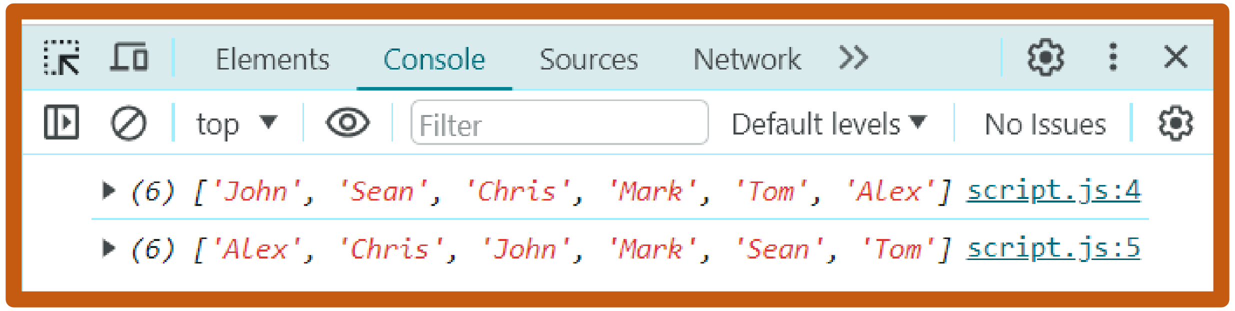
```
▶ (6) ['John', 'Sean', 'Chris', 'Mark', 'Tom', 'Alex'] script.js:2
▶ (6) ['Alex', 'Chris', 'John', 'Mark', 'Sean', 'Tom'] script.js:5
```

- This will overwrite the array

Array - toSorted

- You can use toSorted() to sort an array without overwriting the original array
  - toSorted() returns a new array sorted alphabetically from the original array

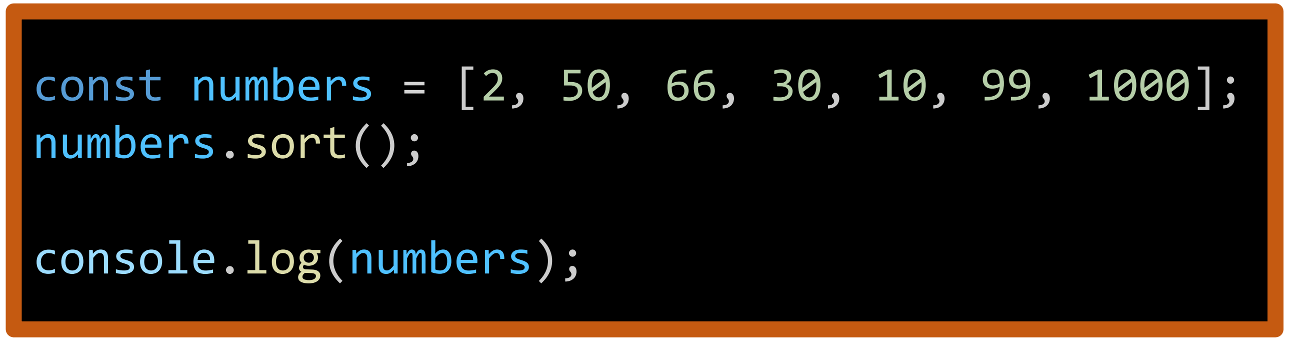
```
const names = ['John', 'Sean', 'Chris', 'Mark', 'Tom', 'Alex'];
const sortedNames = names.toSorted();

console.log(names);
console.log(sortedNames);
```



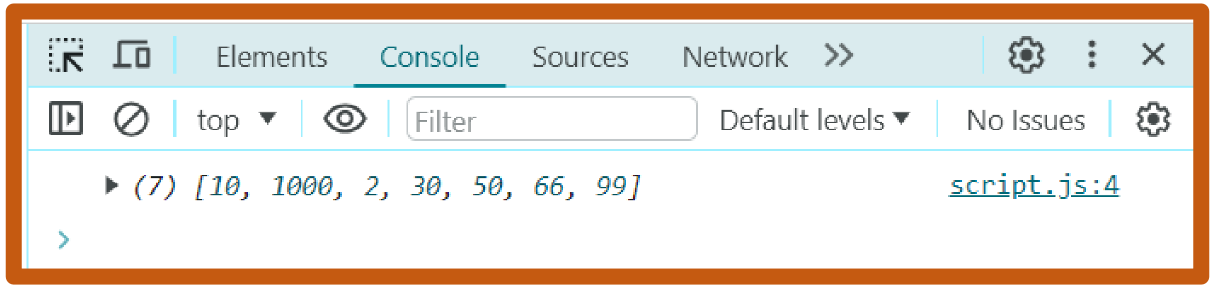
```
▶ (6) ['John', 'Sean', 'Chris', 'Mark', 'Tom', 'Alex'] script.js:4
▶ (6) ['Alex', 'Chris', 'John', 'Mark', 'Sean', 'Tom'] script.js:5
```

Array - sort number problem

- If you try and sort an array based on you will encounter an issue



```
const numbers = [2, 50, 66, 30, 10, 99, 1000];
numbers.sort();

console.log(numbers);
```



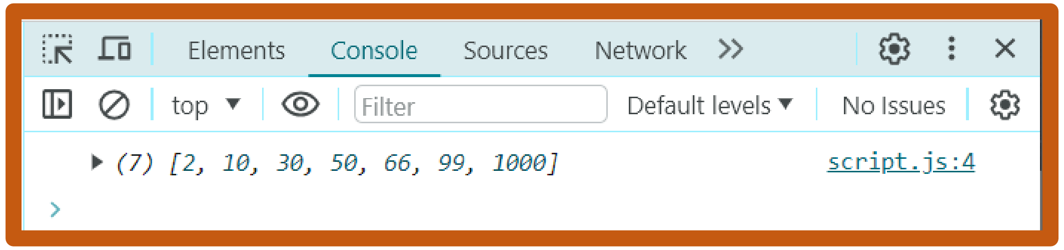
```
▸ (7) [10, 1000, 2, 30, 50, 66, 99]                script.js:4
```

- This is because sort() is used for sorting strings
  - “50” is larger than “1000” because “5” is larger than “1”
  - It is based on the first number rather than the whole number

Array - sort numbers

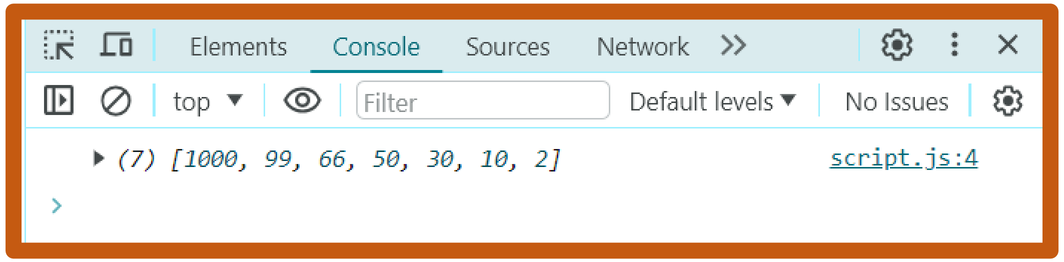
- To sort numbers in an array accurately you need to provide a comparison function that defines the sorting order
  - ((a, b) => a - b) Ascending order (smallest to largest)
  - ((a, b) => a - b) Descending order (largest to smallest)

```
const numbers = [2, 50, 66, 30, 10, 99, 1000];
numbers.sort((a, b) => a - b);

console.log(numbers);
```



```
▸ (7) [2, 10, 30, 50, 66, 99, 1000]                script.js:4
>
```

```
const numbers = [2, 50, 66, 30, 10, 99, 1000];
numbers.sort((a, b) => b - a);

console.log(numbers);
```



```
▸ (7) [1000, 99, 66, 50, 30, 10, 2]                script.js:4
>
```

Exercise 4

- Download and extract ex4.zip
- Follow the instructions in the comments and edit the arrays using the methods
- Print the array to the console

- **Hint**: Some of these methods can make the leaderboard functionality a lot easier

Useful links

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String (String methods)
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- https://developer.mozilla.org/en-US/docs/Web/API/Window/prompt
- https://developer.mozilla.org/en-US/docs/Web/API/Window/confirm
- https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array (Array methods)