前端網絡開發人員課程
（二）進階網絡程式設計

10. ES6 JS III: Syntax III

Presented by Krystal Institute

Learning Objective

- Understand Template literals, array destructuring, arrow functions and importing / exporting custom modules
- Know how to use the new functions and methods effectively

Content

10.1
Revise on the previous lesson

10.2
Template Literals

10.3
Array Destructuring

10.4
Arrow Function

10.5
ES6 Modules

10.1 Revise on the previous lesson

Rest Parameter

- The rest parameter represents any number of arguments after the prefix of the parameter (...)
- The arguments are put after the rest parameter

```
function add(...num) {
    let sum = 0
    for (let i = 0; i < num.length; i++) {
        sum += num[i]
    }
    console.log(sum);
};
add(10, 20, 30, 40, 50);
// same as add([10, 20, 30, 40, 50])
```

Spread Operator

- The spread operator unpacks arguments inside an array
- It uses the same prefix as the rest parameter - ...

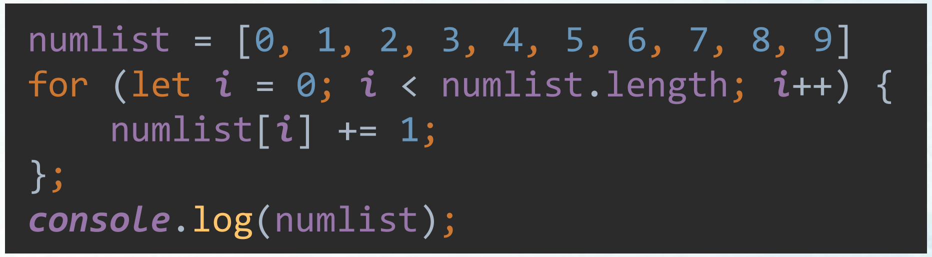
```
let num = [4, 5, 6]
let numbers = [1, ...num, 2, 3]
console.log(numbers) // [1, 4, 5, 6, 2, 3]
```

Object Literals

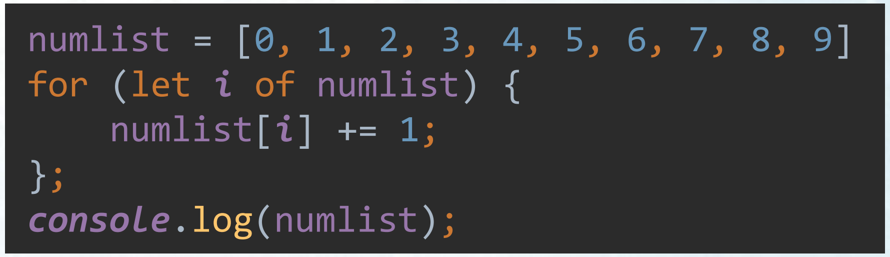
- Before ES6, an object literal is a collection of name-value pairs
- Duplication of name-value pair names can be removed in ES6 by only using the parameter name
- Before ES6, using square brackets will let users use string literals and variables
- In ES6, everything put inside the square brackets will be counted as a string
- Before ES6, property names need to be specified before the value, and it happens even for functions inside an object literal
- In ES6, property name don't have to be included for functions if they have the same name

For…of

- For…of is a new way of iterating over iterable object
- Works very similar to a regular for loop
- array.entries can be used to get the index number and the value of items inside the iterable object



```
numlist = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for (let i = 0; i < numlist.length; i++) {
    numlist[i] += 1;
};
console.log(numlist);
```



```
numlist = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for (let i of numlist) {
    numlist[i] += 1;
};
console.log(numlist);
```

Octal and Binary Literals

- ES6 supports octal literals with the prefix 0o, followed by a octal number

```
let a = 0o23;
console.log(a)
```

- ES6 also supports binary literals with the prefix 0b, followed by a binary number

```
let a = 0b10110;
console.log(a)
```

- Both literals will return a converted decimal number

10.2 Template Literals

Template Literals

- Before ES6, you can use single quotes and double quotes for wrapping strings

```
let single = 'This is a single quoted string';
let double = 'This is a double quoted string';
```

- However, they are limited in their functionality, often needing to write more code to compensate

Template Literals

- A few inconveniences forms when you use the single and double quotes:

- How do you wrap a string that spans multiple lines?
- How do you add variables into the strings?

Template Literals

- A template literal can solve all those issues from normal JS with one simple change:

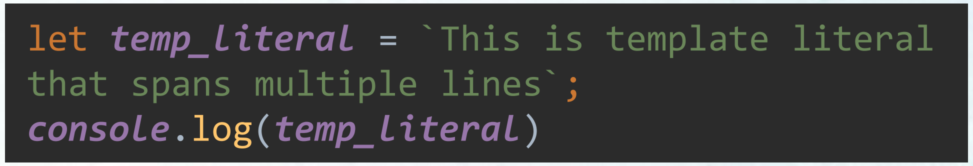
- Instead of wrapping a string with single or double quotes
- Wrap the string with backticks **(`)**
- Backticks is the button to the left of your “1” key

```
let temp_literal = `This is template literal`;
```

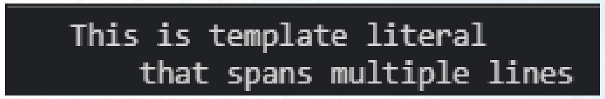
Template Literals

- Template literals can have strings with multiple lines, and will record the newlines inside the string

- You can also use single and double quotes without escaping



```
let temp_literal = `This is template literal
that spans multiple lines`;
console.log(temp_literal)
```



```
This is template literal
   that spans multiple lines
```

Template Literals

- The biggest difference is the substitution of variables into the strings

- A special block can be used inside the template literal (indicated by ${variable}) to assign variables into the template literal

Template Literals

- For instance, when you want to include variables inside a string before ES6, you have to use operators

```
let firstname = “Justin";
let lastname = “Chan";
msg = "Hi" + firstname + " " + lastname + ".";
console.log(msg)
```

- The string will be very long and might sometimes be very confusing if lots of variables are used

Template Literals

- Using the special block, template literals can turn that same message into a much more readable and short version

```
let firstname = “Justin";
let lastname = “Chan";
msg = "Hi" + firstname + " " + lastname + ".";
console.log(msg);
```

- Expressions can also be added for calculations inside the template literal

```
let firstname = “Justin";
let lastname = “Chan";
msg = `Hi ${firstname} ${lastname}.`;
console.log(msg);
```

Template Literals Exercise

- Create and display a template literal that contains a substitution block, as well as a Boolean variable
- The substitution block must contain a Boolean variable, and a if statement should be used to modify the outcome of the template literal

- If the Boolean is true, the template literal should be “This statement is true”
- Otherwise, the template literal should be “This statement is false”

Template Literals Solution

- Using template literal's substitution, we can add an if statement inside the template literal, further shortening the code

```
let bool = true;
let msg = `This statement is ${bool ? "true" : "false"}`;
console.log(msg);
```

10.3 Array Destructuring

Array Destructuring

- ES6 adds a new feature called destructing assignment
- It allows you to destructure…
    - o Properties of an object
    - o Elements of an array
    - o Into individual variables

Array Destructuring

- On the example on right, the getAge function returns an array of different ages
- This is how you assign variables before ES6
- You can assign a variable to the function to get the array
- To get each element out of the array, you need to use indexes

```
function getAge(...ages) {
    return ages;
}

let age = getAge(10,20,30);

let x = age[0];
let y = age[1];
let z = age[2];
```

Array Destructuring

- It could be very lengthy when lots of variables are involved
- To shorten the lines, ES6 allows you to use destructuring assignment on the array, assigning all 3 variables in one line
- Example on right assigns age to the 3 variables in order

```
function getAge(...ages) {
    return ages;
}

let [x, y, z] = getAge(10,20,30);
console.log(x); //10
```

Array Destructuring

- If the function only returns 2 numbers, the third variable (z) will be undefined

```
let [x, y, z] = getAge(10,20);
console.log(x); // 10
console.log(y); // 20
console.log(z); // undefined
```

- Likewise, only the first 3 numbers will be assigned to x, y and z if the function returns more than 3 numbers

```
let [x, y, z] = getAge(10,20, 30, 40);
console.log(x); // 10
console.log(y); // 20
console.log(z); // 30
```

Array Destructuring: Rest Syntax

- The rest syntax (...) can be used in array destructuring, where it takes all the remaining numbers as an array and assign it to the designated name

```
let [x, y, z, ...others] = getAge(10, 20, 30, 40, 50);
console.log(x); // 10
console.log(y); // 20
console.log(z); // 30
console.log(others); // [40, 50]
```

Array Destructuring: Default Values

- The default parameter can also be set for array destructuring, if it is undefined, it will be defaulted into some prefix number

```
let [x, y, z = 30] = getAge(10, 20);
console.log(x); // 10
console.log(y); // 20
console.log(z); // 30
```

Array Destructuring: Nested Arrays

- If the function returns a nested array
- Array destructuring can have nested array as well

```
let [x, y, [color1, color2, color3]] =
getAge(10, 20, ["Red", "Green", "Blue"]);
console.log(x); // 10
console.log(y); // 20
console.log(color1); // "Red"
console.log(color2); // "Green"
console.log(color3); // "Blue"
```

- If nested array is not used when assigning variables, the nested array will be assigned to a value instead

```
let [x, y, z] = getAge(10, 20, ["Red", "Green", "Blue"]);
console.log(x); // 10
console.log(y); // 20
console.log(z); // ["Red", "Green", "Blue"]
```

Array Destructuring

- In practice, array destructuring has many uses
- One of which is swapping values
- Before ES6, you have to add a temporary variable to swap between 2 values

```
let a = 10;
let b = 20;
let temp = a;
a = b;
b = temp;
console.log(a); // 20
console.log(b); // 10
```

Array Destructuring

- With array destructuring in ES6
- You can swap between any number of variables easily
- The example on the right shows the swapping of 4 variables, ordering them manually

```
let a = 2;
let b = 4;
let c = 1;
let d = 3;
[a, b, c, d] = [c, a, d, b];
console.log([a, b, c, d]); // [1, 2, 3, 4]
```

10.4 Arrow Function

Arrow Functions

- Arrow functions are a welcome addition to ES6
- It speeds up the coding process and shortens code
- See the example on right that adds and returns two numbers together
- It is how you create a function before ES6

```
let add = function(x,y) {
  return x + y;
}
console.log(add(1,2)); // 3
```

Arrow Functions

- With arrow functions introduced, the code should be shortened

```
let add = function(x, y) {
  return x + y;
};
console.log(add(1,2)); // 3
```

- The example on the right uses a prefix of an arrow =>, indicating the add function

```
let add = (x, y) => x + y;
console.log(add(1, 4)); // 5
```

Arrow Functions

- The arrow function indicates a function that will return a value, based on the given arguments
- Even if no variables are used, the brackets are still needed
- If block syntax are used, the return keyword needed to be used as well

```
let function = (variables) => return_value;
```

```
let function = (variables) => {return return_value};
```

Arrow Functions Activity

- Activity: Sort an array of numbers

- Use the sort function – will be explained in the next slide
- Add a compare function
- Display the number array

```
let numlist = [60, 2, 100, 4];
numlist.sort((a, b) => a - b);
console.log(numlist);
```

Arrow Functions Activity Explanation

- The sort method sorts arrays in alphabetical order
- It sees array items as strings, so the first letter / number will be compared

```
let numlist = [2, 100];
numlist.sort();
console.log(numlist); // [100, 2]
```

Arrow Functions Activity Explanation

- If only sort() is used, comparing 2 and 100, it sees 2 is bigger than 1 in 100, so the array will be [100, 2] in ascending order

```
let numlist = [2, 100];
numlist.sort((a, b) => a - b);
console.log(numlist); // [2, 100]
```

- By adding a compare function, in each comparison (e.g. 2 and 100), the result will be calculated (2 - 100 = -98)

Arrow Functions Activity Explanation

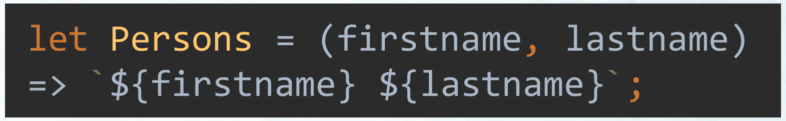
- For the sort method, a negative number means 2 is less than 100, so It will not change the order of the array

- For descending arrays, use b - a instead

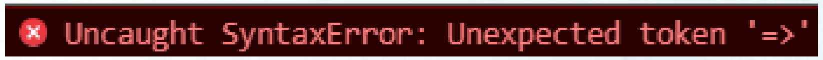
```
let numlist = [2, 100];
numlist.sort((a, b) => a - b);
console.log(numlist); // [2, 100]
```

Arrow Functions: Syntax

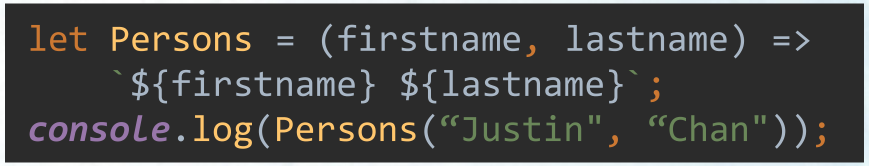
- Line breaks are not allowed with conditions
- The arrow cannot be used after the line break, it will result in SyntaxError
- It will still work if the arrow is used before the line break



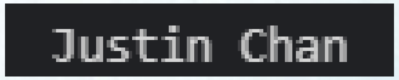
```
let Persons = (firstname, lastname)
=> `${firstname} ${lastname}`;
```



```
Uncaught SyntaxError: Unexpected token '=>'
```



```
let Persons = (firstname, lastname) =>
    `${firstname} ${lastname}`;
console.log(Persons(“Justin", “Chan"));
```



```
Justin Chan
```

10.5 ES6 Modules

ES6 Modules

- Modules are extensions of the script, allowing you to use functions and methods from other js files

- It will be explored deeper on backend JS, for now, modules are not very useful without backend JS

```
    <div id="display"></div>
<script>
    // Assuming this is on the message.js file
    export let message = "This is a message"

    import { message } from './message.js'
    let div = document.querySelector("#display");
    div.textContent = message; // message will be shown on
    // div despite inside another file
```

Exercise

- Create an website with 4 buttons and a empty <div>…

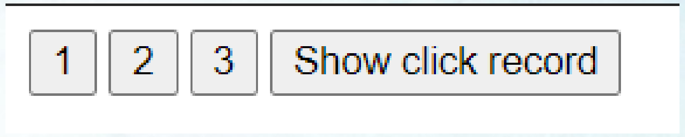
- 3 buttons should have 1, 2, 3 as its names
- The remaining button is used to show all the past record of button clicking

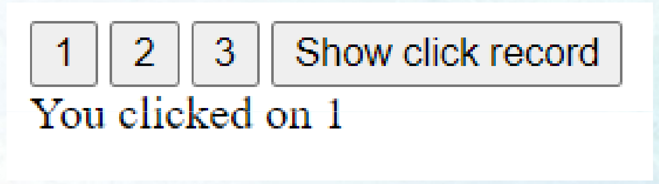
Exercise

- Create an website with 4 buttons and a empty <div>…
- Using arrows functions, display text with numbers associated to the button number on the empty <div> on button click
- Clicking on the show past record button will show an array of clicked button numbers on the empty <div>
- Finish this exercise by the end of this lesson

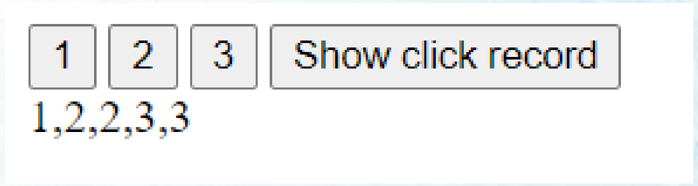
Exercise Example

- Clicking on the 1 button shows a text that you clicked on 1
- The number buttons all work the same, and will record the button clicks
- Clicking on show click record shows all the button the user clicked, in order





You clicked on 1



1,2,2,3,3

References

- Use these if you need more explanations!
- https://www.javascripttutorial.net/es6/
- https://javascript.info/

- Use this if you need more specific answers!
- https://stackoverflow.com/