Important Concepts of ES6

Arrow Functions

- ES6 arrow functions provide you with an alternative way to write a shorter syntax compared to the function expression.

let add = function (x, y) {
        return x + y;
};
 	
console.log(add(10, 20)); // 30
	let add = (x, y) => x + y;
	
	console.log(add(10, 20)); // 30;

  - The arrow function has one expression x + y so it returns the result of the expression.
  - let add = (x, y) => { return x + y; }; //return requires block

Arrow functions with multiple parameters

- Syntax
  - (p1, p2, ..., pn) => expression;
- Example: Sort numbers

```
let numbers = [4,2,6];
numbers.sort(function(a,b){
   return b - a;
});
console.log(numbers); // [6,4,2]
```

```
let numbers = [4,2,6];
numbers.sort((a,b) => b - a);
console.log(numbers); // [6,4,2]
```

Arrow functions with a single parameter

- Syntax
  - (p1) => { statements }   **OR**   p => { statements }
- Arrow function as an argument.

```
let names = ['John', 'Mac', 'Peter'];
let lengths = names.map(name => name.length);

console.log(lengths);
```

Arrow functions with no parameter

- Syntax
  - () => { statements }
-

arrow functions and object literal

```
let setColor = function (color) {
    return {value: color}
};

let backgroundColor = setColor('Red');
console.log(backgroundColor.value); // "Red"
```

vs.

```
let setColor = color => {value: color };
```

arrow functions and this value

- Normal function

```
function Car() {
    this.speed = 0;

    this.speedUp = function (speed) {
        this.speed = speed;
        setTimeout(function () {
            console.log(this.speed); // undefined
        }, 1000);

    };
}

let car = new Car();
car.speedUp(50);
```

this of the anonymous function shadows the this of the speedUp() method.

Solution:

```
let self = this;
setTimeout(function () {
console.log(self.speed); },
1000);
```

arrow functions and this value

```
function Car() {
   this.speed = 0;

   this.speedUp = function (speed) {
      this.speed = speed;
      setTimeout(
         () => console.log(this.speed),
         1000);

   };
}

let car = new Car();
car.speedUp(50); // 50;
```

(...) rest operator

allows you to represent an indefinite number of arguments as an array

```
const combine = (...args) => {
  return args.reduce(function (prev, curr) {
    return prev + ' ' + curr;
  });
};

let message = combine('JavaScript', 'Rest', 'Parameters'); // =>
console.log(message); // JavaScript Rest Parameters
```

(...). The spread operator

allows you to spread out elements of an iterable object such as an array

```
const odd = [1,3,5];
const combined = [2,4,6, ...odd];
console.log(combined);
```

destructuring assignment

allows you to destructure properties of an object or elements of an array
into individual variables.

```
function getScores()
{ return [70, 80, 90]; }
let scores = getScores();
```

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

```
let [x, y, z] = getScores();
```

for … of loop

```
let scores = [80, 90, 70];

for (let score of scores) {
    score = score + 5;
    console.log(score);
}
```

```
let scores = [80, 90, 70];

for (const score of scores) {
    console.log(score);
}
```

```
let colors = ['Red', 'Green', 'Blue'];
for (const [index, color] of colors.entries())
{ console.log(`${color} is at index ${index}`); }
```

JavaScript classes

- JavaScript Classes are templates for JavaScript Objects.
  - keyword class
  - Add method constructor()
  - Methods and properties

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

JavaScript classes

- getter and setter

```
class Person {
    constructor(name) {
        this.name = name;
    }
    get name() {
        return this._name;
    }
    set name(newName) {
        newName = newName.trim();
        if (newName === '') {
            throw 'The name cannot be empty';
        }
        this._name = newName;
    }
}
```

JavaScript classes

- getter in objects

```
let meeting = {
    attendees: [],
    add(attendee) {
        console.log(`${attendee} joined the meeting.`);
        this.attendees.push(attendee);
        return this;
    },
    get latest() {
        let count = this.attendees.length;
        return count == 0 ? undefined :
this.attendees[count - 1];
    }
};

meeting.add('John').add('Jane').add('Peter');
console.log(`The latest attendee is
${meeting.latest}.`);
```

JavaScript classes expression

```
let Person = class {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}
```

Template literals

- Before ES6, single quotes (') or double quotes (")
- In ES6, create a template literal by wrapping text in backticks (`)
- Features
  - Multiline string
  - String formatting
  - Html escaping

```
let simple = `This is a template literal`;
```

```
${variable_name}
```