Table of Contents

JavaScript-ObjectOriented

Js ES5 ES6 | Data Structure

- JavaScript Object-Oriented | Data Structure

https://novemberfall.github.io/JavaScript-ObjectOriented/

- Algorithm with java c++ javascript

https://novemberfall.github.io/Algorithm-FullStack/

- front-end

https://novemberfall.github.io/Front-End_fullstack/

- back-end

https://novemberfall.github.io/Java-backEnd-fullstack/

- Java Object-Oriented

https://novemberfall.github.io/Java-ObjectOriented/

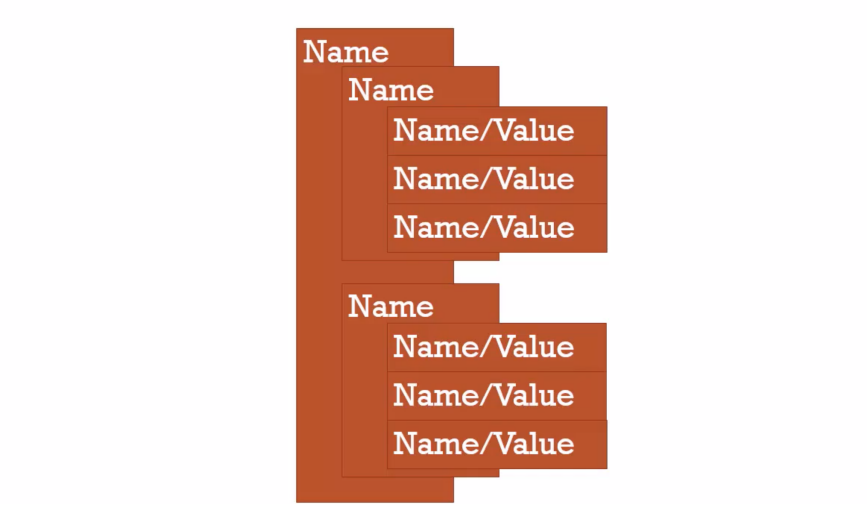
- LeetCode-Algorithm

https://novemberfall.github.io/LeetCode-Algorithm/

Chapter1： Object

1. Object Properties

Objects and Dot



remember this picture

- Object can hava Primitive types, called Property
- An object could have another object connected to it as a child
- An object can also contains functions
- Object hava properties and methods

```
var person = new Object();

person["firstname"] = "Tony";
person["lastname"] = "Alicea";

var firstNameProperty = "firstname";

console.log(person)
console.log(person[firstNameProperty])

console.log(person.firstname);
console.log(person.lastname)

person.address = new Object();
person.address.street = "111 Main St.";
person.address.city = "New York";
person.address.state = "NY";

console.log(person.address.street)  //111 Main St.
console.log(person.address.city)    //New York
console.log(person["address"]["state"]) //NY
```

Objects and properties

```
var john = {
     firstName: 'John',
     lastName: 'Smith',
     birthYear: 1990,
     family: ['Jane', 'Mark', 'Bob', 'Emily'],
     job: 'teacher',
     isMarried: false
 };
 console.log(john.firstName);
 console.log(john['lastName']);
 var x = 'birthYear';
 console.log(john[x]);

 john.job = 'designer';
 john['isMarried'] = true;
 console.log(john);

 //new Object syntax
 var jane = new Object();
 jane.firstName = 'Jane';
 jane.birthYear = 1969;
 jane['lastname'] = 'Smith';
 console.log(jane);
```

Objects and methods:

```
var john = {
    firstName: 'John',
    lastName: 'Smith',
    birthYear: 1990,
    family: ['Jane', 'Mark', 'Bob', 'Emily'],
    job: 'teacher',
    isMarried: false,
    calcAge: function(birthYear){
        this.age = 2019 - birthYear;
    }
};

john.calcAge(1990);
console.log(john);
```

Object property

```
var a = {};
var b = new Object();     // equal to 'var a = {};'
//console.log(a);
console.log(a.constructor === b.constructor); //true

//var a = Object.create();
//Error since you didn't pass any parameters into the proto

var b = Object.create(
    {
        a: 1,
        b: 2
    }
);
console.log('b:', b);
```

2. Prototype

factory function

```
function user(name, age, gender){
    var person = {};
    person.name = name;
    person.age = age;
    person.gender = gender;
    return person;
}

var Tim = user('Tim', 20, 'male');
console.log('Tim:', Tim);
```

Constructor; this way should use new keyword

```
function user(name, age, gender){
     this.name = name;
     this.age = age;
     this.gender = gender;
 }

 var pTim = new user('Tim', 18, 'male');
 console.log('Tim: ', pTim);

 function Dog(){
     this.age = 5;
     this.gender = 'male';
     this.speak = function(){
         console.log('Woof...')
     }
 }
 var dog = new Dog();
 console.log('dog: ', dog);
```

JavaScript Object Prototypes

- All JavaScript objects inherit properties and methods from a prototype.

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}

var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

- We also learned that you can not add a new property to an existing object constructor:

```
Person.nationality = "English";    // Error:
console.log(myFather.nationality);  //undefined
```

Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

- **Date** objects inherit from **Date.prototype**
- **Array** objects inherit from **Array.prototype**
- **Person** objects inherit from **Person.prototype**

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

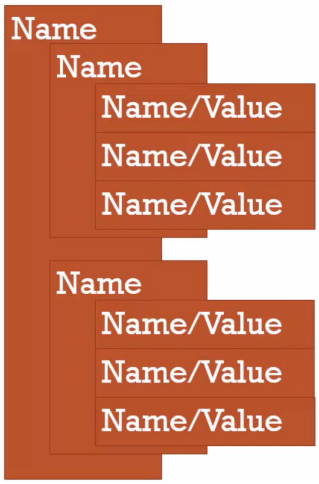
- The JavaScript prototype property allows you to add new properties to object constructors:

```
// The correct way:
Person.prototype.nationality = "English";
console.log(myFather.nationality);   //output: English
console.log(myMother.nationality);   //output: English
```

```
// We also can change the prototype.property
Person.prototype.nationality = "Chinese";
console.log(myFather.nationality);   //output: Chinese
console.log(myMother.nationality);   //output: Chinese

//We can check the _proto_ if they are equal
console.log(myFather._proto_ === myMother._proto_); //True



//We also can check the instance's constructor
console.log(myFather.constructor);   //Point to function Pe
console.log('\n')

//We can define a new object
var myBrother = new myFather.constructor('Tom', 'Zheng', 30
console.log(myBrother);
Person.prototype.nationality = "English";
console.log(myBrother.nationality);   // English
```

3. Name Value Pairs Object

Name/Value pair

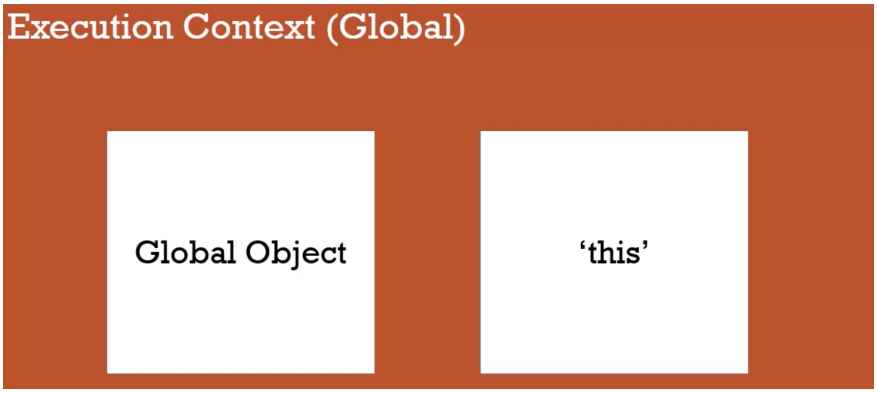


Object:

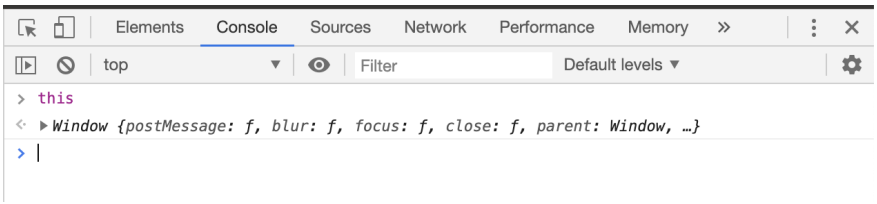
- A collection of Name Value Pairs

```
Address:
{
    Street: 'Main',
    Number: 100
    Apartment:
    {
        Floor: 3,
        Number: 301
    }
}
```

Global Object



- enter this:



When we say GLOBAL

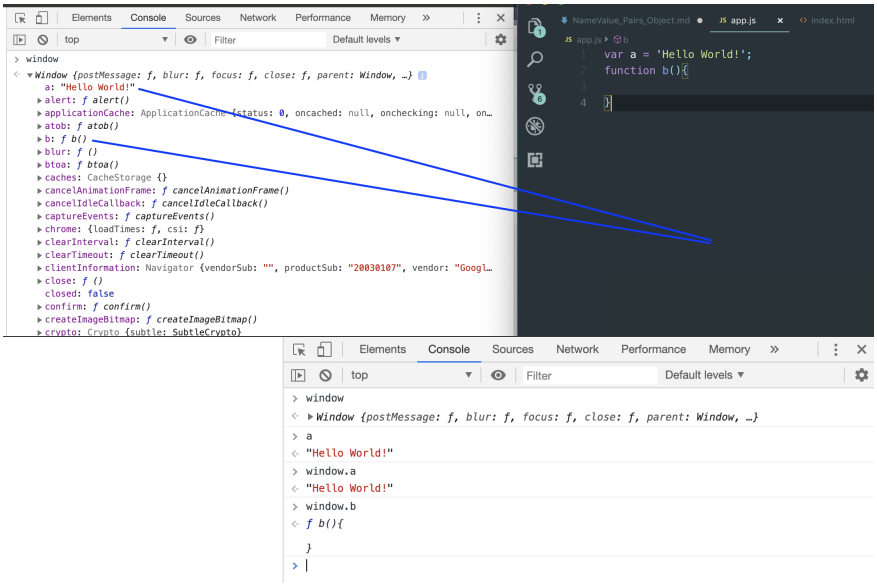
- in javascript "Not Inside a Function"

```
var a = 'Hello World!';
function b(){

}
```

```
<html>
    <head>
    </head>
    <body>
        <script src="app.js"></script>
    </body>
</html>
```

- SINCE window = "this"
- Now, we can see:

1. Object Properties



4. Closures

- Example:

```
function user(name) {
    var age, sex;
    return {
      getName: function() {
        return name;
      },
      setName: function(newName) {
        name = newName;
      },
      getAge: function() {
        return age;
      },
      setAge: function(newAge) {
        age = newAge;
      },
      getSex: function() {
        return sex;
      },
      setSex: function(newSex) {
        sex = newSex;
      }
    }
  }

  var whh = user('王花花');
  whh.setSex('女');
  whh.setAge(22);
  var name = whh.getName();
  var sex = whh.getSex();
  var age = whh.getAge();
  console.log(name, sex, age);//王花花 女 22
```

- Summary:**An inner function has always access to the variables and parameters of its outer function, even after the outer function has returned.**

```
function retirement(retirementAge){
    var a = ' years left until retirement.';
    return function(yearOfBirth){
        var age = 2019-yearOfBirth;
        console.log((retirementAge - age) + a);
    }
}

var retirementUS = retirement(66);
/* 1. pass parameter 66 into function retirement, which, (1
 (2) return the anonymous function with parameter 'yearOfBi
 and its execution context gets popped off the stack
 */
retirementUS(1990); //Output: 37 years left until retiremer
/* 2. In this function, we use the retirementAge paramente
and also a variable 'a' that is also declared outside of th
When we run this anonymous function, this works, so somehow
even after retirement function, which declares these variab
 */
/**
 * Overall: Our inner function here is able to use the reti
 * function here that is already gone. It has already retur
 * there, and this is the closure.
 * /

// equal:
//retirement(66)(1990);
```

convert to Clsures functioning

```
/*
function interviewQuestion(job){
    if(job === 'designer'){
        return function(name){
            console.log(name + ', can you please explain wh
            'UX design is?');
        }
    }else if(job === 'teacher'){
        return function(name){
            console.log('What subject do you teach, '+name-
        }
    }else{
        return function(name){
            console.log('Hello '+name+', what do you do?');
        }
    }
}
*/

function interviewQuestion(job){
    return function(name){
        if(job === 'designer'){
            console.log(name + ', can you please explain wh
            'UX design is?');
        }else if (job === 'teacher'){
            console.log('What subject do you teach, '+name-
        }else{
            console.log('Hello '+name+', what do you do?');
        }
    }
}
interviewQuestion('teacher')('John'); //What subject do you
```

5. Everything is an object

- Every JavaScript object has a **prototype property**, which makes inheritance possible in JavaScript;
- The prototype property of an object is where we put methods and properties that we want **other objects to inherit**;
- The Constructor's prototype property is NOT the prototype of the Constructor itself, it's the prototype of **ALL** instances that are created through it;
- When a certain method(or property) is called, the search starts in the object itself, and if it cannot be found, the search moves on to the object's prototype. This continues until the method is found: **prototype chain**

define a function:

```
var Person = function(name, yearOfBirth, job){
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.job = job;
    this.calculateAge = function(){
        console.log(2019- this.yearOfBirth);
    }
}
```

- we can use prototype

```
var Person = function(name, yearOfBirth, job){
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.job = job;
}

Person.prototype.calculateAge = function(){
    console.log(2019- this.yearOfBirth);
};
```

```
var Person = function(name, yearOfBirth, job){
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.job = job;
}

Person.prototype.calculateAge = function(){
    console.log(2019- this.yearOfBirth);
};

Person.prototype.lastName = 'Smith';

var john = new Person('John', 1990, 'teacher');
var jane = new Person('Jane', 1969, 'designer');
var mark = new Person('Mark', 1948, 'retired');

john.calculateAge();
jane.calculateAge();
mark.calculateAge();

console.log(john.lastName)
console.log(jane.lastName)
console.log(mark.lastName)
```

- the above codes,
- because it is the prototype proterty of the function constructor.
- All John, jane and Mark inherit this property.

```
//Object.Create
var personProto = {
    calculateAge: function(){
        console.log(2019 - this.yearOfBirth);
    }
};

var john = Object.create(personProto);
john.name = 'John';
john.yearOfBirth = 1990;
john.job = 'teacher';

var jane = Object.create(personProto, {
    name:{ value: 'Jane'},
    yearOfBirth:{ value: 1969},
    job: {value: 'designer'}
});
```

Primitives vs objects

```
// primitives
var a = 23;
var b = a;
a = 46;
console.log(a);     //46
console.log(b);     //23

// Objects
var obj1 = {
    name: 'John',
    age: 26;
};
var obj2 = obj1;
obj1.age = 30;
console.log(obj1.age);  //30
console.log(obj2.age);  //30
```

- we didn't create a new object; only create a reference to object1

Chapter2：

1. for loop

for loop with continue and break;

```
/**
 * Loops and iteration
 */

var john = ['John', 'Smith', 1990, 'designer', false, 'blue

for(var i=0; i<john.length; i++){
    if(typeof john[i] !== 'string'){
        continue;
    }
    console.log(john[i]);
}

console.log('\n')
for(var i=0; i<john.length; i++){
    if(typeof john[i] !== 'string'){
        break;
    }
    console.log(john[i]);
}

console.log('\n')
// Looping backwards
for(var i = john.length - 1; i>=0; i--){
    console.log(john[i]);
}
```

2. Array ES5

```
/*****************************
* Arrays
*/

// Initialize new array
var names = ['John', 'Mark', 'Jane'];
var years = new Array(1990, 1969, 1948);

console.log(names[2]);
console.log(names.length);

// Mutate array data
names[1] = 'Ben';
names[names.length] = 'Mary';
console.log(names); //["John", "Ben", "Jane", "Mary"]

// Different data types
var john = ['John', 'Smith', 1990, 'designer', false];

john.push('blue');//[John", "Smith", 1990, "designer", fals
john.unshift('Mr.');
console.log(john);//["Mr.", "John", "Smith", 1990, "designe

john.pop();
john.pop();
john.shift();
console.log(john); //["John", "Smith", 1990, "designer"]

console.log(john.indexOf(23));//-1

var isDesigner = john.indexOf('designer') === -1 ? 'John is
console.log(isDesigner);//John IS a designer
```

Array.filter() mthod

```
const scores = [10, 30, 15, 25, 50, 40, 5];

const filteredScores = scores.filter((score) => {
    return score > 20;
});
console.log(filteredScores);

//[30, 25, 50, 40]
```

```
const users = [
    { name: 'shaun', permium: true },
    { name: 'yoshi', permium: false },
    { name: 'mario', permium: false },
    { name: 'chun-li', permium: true },
];

const permiumUsers = users.filter((user) => {
    return user.permium;
})
console.log(permiumUsers);

/*
[
  { name: 'shaun', permium: true },
  { name: 'chun-li', permium: true }
]
*/
```

- we can alter

```
const users = [
    { name: 'shaun', permium: true },
    { name: 'yoshi', permium: false },
    { name: 'mario', permium: false },
    { name: 'chun-li', permium: true },
];

const permiumUsers = users.filter((user) => user.permium);
console.log(permiumUsers);
/*
[
  { name: 'shaun', permium: true },
  { name: 'chun-li', permium: true }
]
*/
```

```
const products = [
    { name: 'gold star', price: 20 },
    { name: 'mushroom', price: 40 },
    { name: 'green shells', price: 30 },
    { name: 'banana skin', price: 10 },
    { name: 'red shells', price: 50 },
];
const saleProducts = products.map((product) => {
    if (product.price > 30) {
        product.price = product.price / 2;
        return product;
    } else {
        return product;
    }
})
console.log(saleProducts, products);

//output:
[
  { name: 'gold star', price: 20 },
  { name: 'mushroom', price: 20 },
  { name: 'green shells', price: 30 },
  { name: 'banana skin', price: 10 },
  { name: 'red shells', price: 25 }
]
[
  { name: 'gold star', price: 20 },
  { name: 'mushroom', price: 20 },
  { name: 'green shells', price: 30 },
  { name: 'banana skin', price: 10 },
  { name: 'red shells', price: 25 }
]
```

- when we `console.log(saleProducts, products);` , we get the same results
- they're going to be exactly the same because it's directly edited the original one because when we take in an item from an array it's an object and we edited over here it's editing it directly.
- This isn't a copy of the object that we pass in it's the object directly that we're passing it. So then when we edit it it's going to edit the original value and we don't want to do that.

`To avoid this, we need to overwrite the above codes:`

```
const products = [
    { name: 'gold star', price: 20 },
    { name: 'mushroom', price: 40 },
    { name: 'green shells', price: 30 },
    { name: 'banana skin', price: 10 },
    { name: 'red shells', price: 50 },
];

const saleProducts = products.map((product) => {
    if (product.price > 30) {
        return {
            name: product.name,
            price: product.price / 2
        }
    } else {
        return product;
    }
})

console.log(saleProducts, products);

//output:
[
  { name: 'gold star', price: 20 },
  { name: 'mushroom', price: 20 },
  { name: 'green shells', price: 30 },
  { name: 'banana skin', price: 10 },
  { name: 'red shells', price: 25 }
] [
  { name: 'gold star', price: 20 },
  { name: 'mushroom', price: 40 },
  { name: 'green shells', price: 30 },
  { name: 'banana skin', price: 10 },
  { name: 'red shells', price: 50 }
```

- So that's why we create a new object over here because then that doesn't edit the original value because we're crating a new object and we're setting that equal to the product price of it too.
- We're just saying the new price property on our new object is equal to this.

Reduce method

```
//reduce method
const scores = [10, 30, 15, 25, 70, 90, 30];

const result = scores.reduce((accumulator, currentValue) =>
    if (currentValue > 50) {
        accumulator++;
    }
    return accumulator;
}, 0);

console.log(result);

//output:
2
```

```
const scores = [
    { player: 'mario', score: 50 },
    { player: 'yoshi', score: 30 },
    { player: 'mario', score: 70 },
    { player: 'crystal', score: 60 }
];

const marioTotal = scores.reduce((acc, curr) => {
    if (curr.player === 'mario') {
        acc += curr.score;
    }
    return acc;
}, 0);

console.log(marioTotal);
//120
```

Find method

- The find() method returns the value of the first element in the array that satisfies the provided testing function. Otherwise undefined is returned.

```
// Find method
const scores = [10, 5, 0, 40, 60, 10, 20, 70];

const firstHighScore = scores.find((score) => {
    return score > 50;
});
console.log(firstHighScore);

//60
```

sort method

```
//sort method

//example 1 - sorting strings
const names = ['mario', 'shaun', 'chun-li', 'yoshi', 'luig

names.sort();
console.log(names);
// [ 'chun-li', 'luigi', 'mario', 'shaun', 'yoshi' ]


//example 2 - sorting numbers
const scores = [10, 50, 20, 5, 35, 70, 45];
scores.sort();
console.log(scores);
/*
[
  10, 20, 35, 45,
   5, 50, 70
]
*/



//example 3 - sorting objects
const players = [
    { name: 'mario', score: 20 },
    { name: 'luigi', score: 10 },
    { name: 'chun-li', score: 50 },
    { name: 'yoshi', score: 30 },
    { name: 'shaun', score: 70 }
];

players.sort((a, b) => {
    if (a.score > b.score) {
        return -1;
    } else if (b.score > a.score) {
        return 1;
    } else {
        return 0;
    }
});

console.log(players)
/*
[
  { name: 'shaun', score: 70 },
  { name: 'chun-li', score: 50 },
  { name: 'yoshi', score: 30 },
```

```
  { name: 'mario', score: 20 },
  { name: 'luigi', score: 10 }
]
*/
```

- we can alter players.sort()

```
//example 3 - sorting objects
const players = [
    { name: 'mario', score: 20 },
    { name: 'luigi', score: 10 },
    { name: 'chun-li', score: 50 },
    { name: 'yoshi', score: 30 },
    { name: 'shaun', score: 70 }
];

players.sort((a, b) => b.score - a.score);
console.log(players)


//output:
[
  { name: 'shaun', score: 70 },
  { name: 'chun-li', score: 50 },
  { name: 'yoshi', score: 30 },
  { name: 'mario', score: 20 },
  { name: 'luigi', score: 10 }
]




//example 2 - sorting numbers
const scores = [10, 50, 20, 5, 35, 70, 45];
scores.sort((a, b) => b - a)
console.log(scores)
```

chaining Array methods

```
//chaining array methods
const products = [
    { name: 'gold star', price: 30 },
    { name: 'green shell', price: 10 },
    { name: 'red shells', price: 40 },
    { name: 'banana skin', price: 5 },
    { name: 'mushroom', price: 50 },
];

const filtered = products.filter(product => product.price >
const promos = filtered.map(product => {
    return `the ${product.name} is ${product.price / 2} pou
});
console.log(promos)

//output:
[
  'the gold star is 15 pounds',
  'the red shells is 20 pounds',
  'the mushroom is 25 pounds'
]
```

- the other way:

```
//chaining array methods
const products = [
    { name: 'gold star', price: 30 },
    { name: 'green shell', price: 10 },
    { name: 'red shells', price: 40 },
    { name: 'banana skin', price: 5 },
    { name: 'mushroom', price: 50 },
];

const promos = products
    .filter(product => product.price > 20)
    .map(product => `the ${product.name} is ${product.price
console.log(promos);

//output:
[
  'the gold star is 15 pounds',
  'the red shells is 20 pounds',
  'the mushroom is 25 pounds'
]
```

3. Array ES6

ES5

index.html

```
<html lang="en">
    <head>
        <title>Section 7: Get Ready for the Future: ES6 / E
        <style>
            .box {
                width: 200px;
                padding: 60px;
                text-align: center;
                font-size: 30px;
                margin-top: 50px;
            }
            .green { background-color: green; }
            .blue { background-color: dodgerblue; }
            .orange { background-color: orangered; }
        </style>

    </head>
    <body>
        <h1>Section 7: Get Ready for the Future: ES6 / ES20
        <div class="box green">I'm green!</div>
        <div class="box blue">I'm blue!</div>
        <div class="box orange">I'm orange!</div>

        <script src="script.js"></script>
    </body>
</html>
```

```
const boxes = document.querySelectorAll('.box');

//ES5
var boxesArr5 = Array.prototype.slice.call(boxes);
boxesArr5.forEach(function(cur){
    cur.style.backgroundColor = 'dodgerblue';
});
```

ES6

```
const boxesArr6 = Array.from(boxes);
boxesArr6.forEach(cur => cur.style.backgroundColor = 'dodge
```

- Or: (combine two statement)

```
Array.from(boxes).forEach(cur => cur.style.backgroundColor
```

ES5

```
for(var i=0; i<boxesArr5.length; i++){
    if(boxesArr5[i].className === 'box blue'){
        continue;
    }
    boxesArr5[i].textContent = 'I changed to blue!';
}
```

Section 7: Get Ready for the Future: ES6 / ES2015

The same result of ES6:

```
//ES6
for(const cur of boxesArr6){
    if(cur.className.includes('blue')){
        continue;
    }
    cur.textContent = 'I changed to blue';
}
```

ES5 map && ES6

```
//ES5
var ages = [12, 17, 8, 21, 14, 11];

var fullAge = ages.map(function(cur){
    return cur >= 18;
});
console.log(fullAge); //[false, false, false, true, false,
console.log(fullAge.indexOf(true)); //3
console.log(ages[fullAge.indexOf(true)]); //21

//ES6
console.log(ages.findIndex(cur => cur >= 18));  //3
console.log(ages.find(cur => cur>= 18)); //21
```

Spread Operator

```
function addFourAges (a, b, c, d){
    return a + b + c + d;
}
var sum1 = addFourAges(18, 30,12, 21);
console.log(sum1);//81

//ES5
var ages =[18, 30, 12, 21];
var sum2 = addFourAges.apply(null, ages);
console.log(sum2);//81

//ES6
const sum3 = addFourAges(...ages);
console.log(sum3);//81

const familySmith = ['John', 'Jane', 'Mark'];
const familyMiller = ['Mary', 'Bob', 'Ann'];
let bigFamily = [...familySmith, ...familyMiller];
console.log(bigFamily);//["John", "Jane", "Mark", "Mary", "
bigFamily = [...familySmith, 'Lily', ...familyMiller];
console.log(bigFamily);//["John", "Jane", "Mark", "Lily", "
```

html implementation

```
<body>
        <h1>Section 7: Get Ready for the Future: ES6 / ES2

        <div class="box green">I'm green!</div>
        <div class="box blue">I'm blue!</div>
        <div class="box orange">I'm orange!</div>

        <script src="script.js"></script>
</body>
```

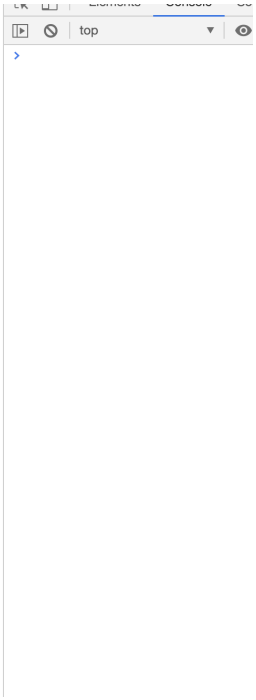
```
const h = document.querySelector('h1');
const boxes = document.querySelectorAll('.box');
const all = [h, ...boxes];
Array.from(all).forEach(cur => cur.style.color = 'purple');

//The same effect
const h = document.getElementsByTagName('h1')[0];
const boxes = document.getElementsByTagName('div');
const all = [h, ...boxes];
Array.from(all).forEach(cur => cur.style.color = 'purple');
```

Section 7: Get Ready for the Future: ES6 / ES2015

I'm green!

I'm blue!

I'm orange!



Chapter3：Date Time Library

1. Date | Time

Date & Time

```
//dates & times
const now = new Date();
console.log(now);
// Fri Jul 26 2019 00:19:51 GMT-0700 (Pacific Daylight Time

console.log(typeof now); //object

console.log('getFullYear:', now.getFullYear());
console.log('getMonth:', now.getMonth());
console.log('getDate:', now.getDate());
console.log('getDay:', now.getDay());
console.log('getHours:', now.getHours());
console.log('getMinutes:', now.getMinutes());
console.log('getSeconds:', now.getSeconds());

/*
getFullYear: 2019
getMonth: 6
getDate: 26
getDay: 5
getHours: 0
getMinutes: 23
getSeconds: 57
*/




//tiemstamps
console.log('timestamp', now.getTime());  //timestamp 1564

//date strings
console.log(now.toDateString());
console.log(now.toTimeString());
console.log(now.toLocaleString());

/*
Fri Jul 26 2019
00:26:40 GMT-0700 (Pacific Daylight Time)
7/26/2019, 12:26:40 AM
*/
```

Chapter4：Class

1. Class

=======

- **Classes, unlike functions are not hoisted. You need to declare a class before you can construct an instance.**
- A class can have at most one constructor. After all, class is just syntactic sugar for defining a constructor function. If you declare a class without a constructor, it automatically gets a constructor with an empty body.
- Unlike in an object literal, in a class declaration, you do not use commas to separate the method definitions.
- The body of a class is executed in strict mode.

ES5

```
var Person5 = function(name, yearOfBirth, job){
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.job = job;

}

Person5.prototype.calculateAge = function(){
    var age = new Date().getFullYear - this.yearOfBirth;
    console.log(age);
}

var john5 = new Person5('John', 1990, 'teacher');
/*
Person5 {name: "John", yearOfBirth: 1990, job: "teacher"}
job: "teacher"
name: "John"
yearOfBirth: 1990
__proto__: Object
*/
```

ES6

```
class Person6{
    constructor(name, yearOfBirth, job){
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job;
    }

    calculateAge(){
        var age = new Date().getFullYear - this.yearOfBirth
        console.log(age);
    }
}

const john6 = new Person6('John', 1990, 'teacher');
/*
Person6 {name: "John", yearOfBirth: 1990, job: "teacher"}
job: "teacher"
name: "John"
yearOfBirth: 1990
__proto__: Object
*/
```

adding a static method to a class

```
class Person6{
    constructor(name, yearOfBirth, job){
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job;
    }

    calculateAge(){
        var age = new Date().getFullYear - this.yearOfBirth
        console.log(age);
    }

    static greeting(){
        console.log('Hi there!');
    }
}

Person6.greeting(); // Hi there
```

class contructor

```
class User{
    constructor(){
        //set up properties
        this.username = 'mario';
    }
}

const userOne = new User();
// the 'new' keyword
// 1 - it creates a new empty object{}
// 2 - it binds the value of 'this' to the new empty object
// 3 - it calls the constructor function to 'build' the obj
```

2. Sub Class

classes and subclasses

ES5

```
var Person5 = function(name, yearOfBirth, job){
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.job = job;
}

Person5.prototype.calculateAge = function(){
    var age = new Date().getFullYear() - this.yearOfBirth;
    console.log(age);
}

var john5 = new Person5('John', 1990, 'teacher');
//Person5 {name: "John", yearOfBirth: 1990, job: "teacher"}

var Athletes5 = function(name, yearOfBirth, job, olymicGame
    Person5.call(this, name, yearOfBirth, job);
    this.olymicGames = olymicGames;
    this.medals = medals;
}

Athletes5.prototype = Object.create(Person5.prototype);

Athletes5.prototype.wonMedal = function(){
    this.medals++;
    console.log(this.medals);
}

var johnAthlete5 = new Athletes5('John', 1990, 'swimmer', 3
/*
Person5 {
  name: 'John',
  yearOfBirth: 1990,
  job: 'swimmer',
  olymicGames: 3,
  medals: 10
}
*/

johnAthlete5.calculateAge();//29
johnAthlete5.wonMedal();//11
```

ES6

```
//ES6
class Person6{
    constructor(name, yearOfBirth, job){
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job;
    }

    calculateAge(){
        var age = new Date().getFullYear() - this.yearOfBi
        console.log(age);
    }

    static greeting(){
        console.log('Hi there!');
    }
}

class Athletes6 extends Person6{
    constructor(name, yearOfBirth, job, olympicGame, medals
        super(name, yearOfBirth, job);
        this.olympicGame = olympicGame;
        this.medals = medals;
    }

    wonMedal(){
        this.medals++;
        console.log(this.medals);
    }
}

const johnAthlete6 = new Athletes6('John', 1990, 'swimmer',
/*
Athletes6 {
  name: 'John',
  yearOfBirth: 1990,
  job: 'swimmer',
  olympicGame: 3,
  medals: 10
}
*/
johnAthlete6.wonMedal();//11
johnAthlete6.calculateAge();//29
```

Chapter5：Call back

1. callback

What is a Callback?

- **Simply put**: A callback is a function that is to be executed **after** another function has finished executing — hence the name ‘call back’.
- **More complexly put**: In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called **higher-order functions**. Any function that is passed as an argument is called a **callback function**.

Why do we need Callbacks?

- For one very important reason — JavaScript is an event driven language. This means that instead of waiting for a response before moving on, JavaScript will keep executing while listening for other events. Lets look at a basic example:

```
function first() {
    console.log(1);
}

function second() {
    console.log(2);
}

first();
second();

// 1
// 2
```

- But what if function `first` contains some sort of code that can’t be executed immediately? For example, an API request where we have to send the request then wait for a response? To simulate this action, were going to use `setTimeout` which is a JavaScript function that calls a function after a set amount of time. We’ll delay our function for 500 milliseconds to simulate an API request. Our new code will look like this:

```
const first = () => {
    setTimeout(() => {
        console.log(1);
    }, 500);
}

const second = () => {
    console.log(2);
}

first();
second();


// 2
// 1
```

- Even though we invoked the `first()` function first, we logged out the result of that function after the `second()` function.
- It's not that JavaScript didn't execute our functions in the order we wanted it to, it's instead that **JavaScript didn't wait for a response from `first()` before moving on to execute `second()`**.
- Because you can't just call one function after another and hope they execute in the right order. Callbacks are a way to make sure certain code doesn't execute until other code has already finished execution.

Create a Callback

```
function doHomework(subject) {
  alert(`Starting my ${subject} homework.`);
}



doHomework('math');
// Alerts: Starting my math homework.
```

- Now lets add in our callback — as our last parameter in the `doHomework()` function we can pass in `callback`. The callback function is then defined in the second argument of our call to `doHomework()`.

```
function doHomework(subject, callback) {
  alert(`Starting my ${subject} homework.`);
  callback();
}

doHomework('math', function() {
  alert('Finished my homework');
});
```

- As you'll see, if you type the above code into your console you will get two alerts back to back: Your 'starting homework' alert, followed by your 'finished homework' alert.

- But callback functions don't always have to be defined in our function call. They can be defined elsewhere in our code like this:

```
function doHomework(subject, callback) {
  alert(`Starting my ${subject} homework.`);
  callback();
}
function alertFinished(){
  alert('Finished my homework');
}
doHomework('math', alertFinished);
```

- This result of this example is exactly the same as the previous example, but the setup is a little different. As you can see, we've passed the alertFinished function definition as an argument during our `doHomework()` function call!

A real world example:

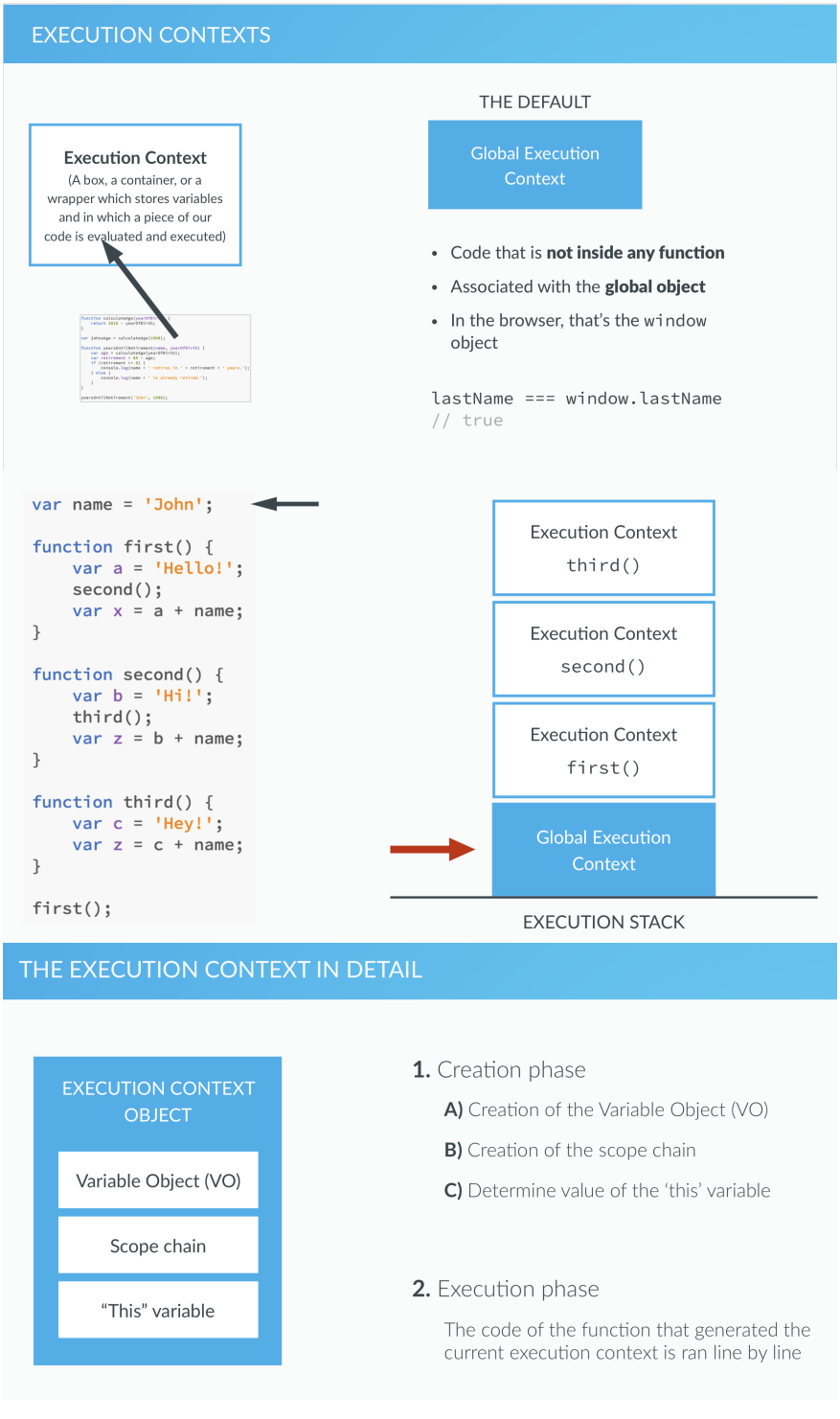
- When you make requests to an **Twitters** API, you have to wait for the response before you can act on that response. This is a wonderful example of a real-world callback. Here's what the request looks like:

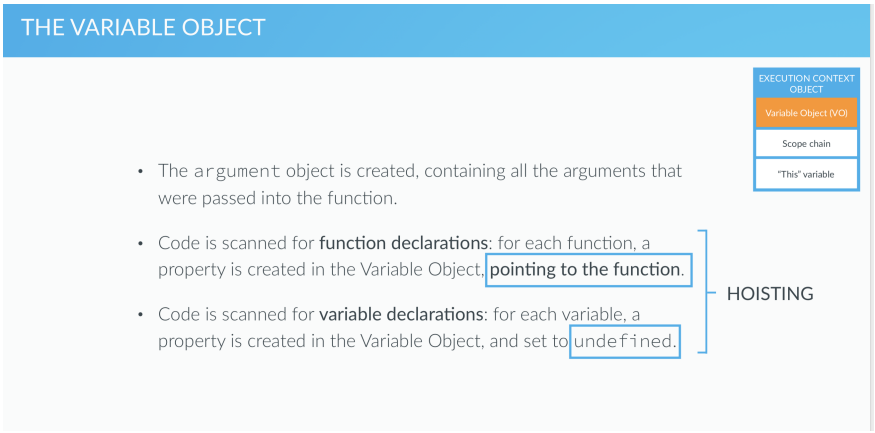
```
T.get('search/tweets', params, function(err, data, response
  if(!err){
    // This is where the magic will happen
  } else {
    console.log(err);
  }
})
```

- `T.get` simply means we are making a get request to Twitter
- There are three parameters in this request: `'search/tweets'`, which is the route of our request, params which are our search parameters, and then an anonymous function which is our callback.
- A callback is important here because we need to wait for a response from the server before we can move forward in our code. We don't know if our API request is going to be successful or not so after sending our parameters to search/tweets via a get request, we wait. Once Twitter responds, our callback function is invoked. Twitter will either send an `err` (error) object or a `response` object back to us. In our callback function we can use an `if()` statement to determine if our request was successful or not, and then act upon the new data accordingly.

2. Execution Context Stack

How javascript works behind the scenes





Hoisting

- **calculateAge which will work, since it is function declaration**

```
calculateAge(1965); //54
function calculateAge(year){
    console.log(2019 - year);
}
```

- **the below, retirement(1956) not working, since retirement is not function declarations**

```
retirement(1956); //not working
var retirement = function(year){
    console.log(65 - (2019 - year));
}
```

Chapter6：call apply bind

1. call apply bind

```
function yo(){
    console.log('Yo, '+ name + ' 我是 '+this.name);
}

var whh = {
    name: '王花花'
}

var lsd = {
    name: '李双蛋'
}
yo.call(whh) //Yo,  我是 王花花
yo.call(lsd) //Yo,  我是 李双蛋
console.log('\n')
```

- version2

```
function yo(name, a, b, c){
    console.log('Yo, '+ name + ' 我是 '+this.name);
}

var whh = {
    name: '王花花'
}

var lsd = {
    name: '李双蛋'
}
yo.call(whh, '招可爽', 1, 2, 3) //Yo, 招可爽 我是 王花花
//yo.call(lsd, '刘贝贝', 3, 2, 1) //Yo, 刘贝贝 我是 李双蛋
yo.apply(whh, ['招可爽', 1, 2, 3]) //Yo, 招可爽 我是 王花花

yobind = yo.bind(whh);
yobind('招可爽')
```

call

```
function foo(){
    console.log('.call() executing...');
}
foo();
foo.call(); //this statement is equal to foo()
console.log('\n')

// Bind, call and apply
var john = {
    name: 'John',
    age: 25,
    job: 'teacher',
    presentation: function(style, timeOfDay){
        if(style === 'formal'){
            console.log('Good '+timeOfDay+' ,Ladies '+
            'gentlemen! I\'m '+ this.name +
            ', I\'m a '+this.job+' and I\'m '+
            this.age+ ' years old.');
        }else if(style === 'friendly'){
            console.log('Hey! What\'s up? I\'m '+ this.name
            ', I\'m a '+this.job+' and I\'m '+
            this.age+ ' years old. Have a nice '+
            timeOfDay + '.' );
        }
    }
};

var emily = {
    name: 'Emily',
    age: 35,
    job: 'designer'
};

john.presentation('formal', 'morning');
/*  output:
Good morning ,Ladies gentlemen! I'm John, I'm a teacher and
 */
john.presentation.call(emily, 'friendly','afternoon');
/* output:
Hey! What's up? I'm Emily, I'm a designer and I'm 35 years
 */
```

- the above piece of codes:
- we called it———— **method borrowing, because we actually borrowed the method from John , to use it here on the Emily object. Since we use John's presentation method, but setting**

the this variable to "emily"

the other method is called Apply method

- the only different way is that this one accepts the arguments as an array so that's only two arguments, first this variable, and then an array where all the other arguments go.

```
john.presentation.apply(emily, ['friendly', 'afternoon']);
/*  output:
Hey! What's up? I'm Emily, I'm a designer and I'm 35 years
 */
```

bind method

- It is very similar to the call method as well, so it also allows us to set the this variable explicitly. However, the difference here is that bind doesn't immediately call the function, but instead it generates a copy of the function so that we can store it somewwhere

```
var johnFriendly = john.presentation.bind(john, 'friendly'
johnFriendly('morning');
johnFriendly('night');
/* output:
Hey! What's up? I'm John, I'm a teacher and I'm 25 years o
 */

 var emilyFormal = john.presentation.bind(emily, 'formal');
 emilyFormal('afternoon');
/* output
Good afternoon ,Ladies gentlemen! I'm Emily, I'm a designe
*/



/* function as arguments; bind method */
var years = [1990, 1965, 1937, 2005, 1998];
function arrayCalc(arr, fn){
    var arrRes = [];
    for(var i=0; i<arr.length; i++){
            arrRes.push(fn(arr[i]));
    }
    return arrRes;
}

function calculateAge(el){
    return 2019 - el;
}

function isFullAge(limit, el){
    return el >= limit;
}

var ages = arrayCalc(years, calculateAge);
var fullJapan = arrayCalc(ages, isFullAge.bind(this, 20));
console.log(ages);
console.log(fullJapan);
```

Chapter7： Function | IIFE

1. function es5

Passing functions as Arguments

- A function is an instance of the Object types;
- A function behaves like any other object;
- We can store functions in a variable;
- We can pass a function ase an argument to another function;
- We can return a function from a function

```
var years = [1990, 1965, 1937, 2005, 1998];
function arrayCalc(arr, fn){
    var arrRes = [];
    for(var i=0; i<arr.length; i++){
            arrRes.push(fn(arr[i]));
    }
    return arrRes;
}

function calculateAge(el){
    return 2019 - el;
}

function isFullAge(el){
    return el >= 18;
}

function maxHeartRate(el){
    if(el >= 18 && el <= 81){
        return Math.round(206.9 - (0.67 * el));
    }else{
        return -1;
    }
}

var ages = arrayCalc(years, calculateAge);
var fullAge = arrayCalc(ages, isFullAge);
var rates = arrayCalc(ages, maxHeartRate);
console.log(ages);
console.log(fullAge);
console.log(rates);
```

Functions returning functions

```
function interviewQuestion(job){
    if(job === 'designer'){
        return function(name){
            console.log(name + ', can you please explain wh
            'UX design is?');
        }
    }else if(job === 'teacher'){
        return function(name){
            console.log('What subject do you teach, '+name-
        }
    }else{
        return function(name){
            console.log('Hello '+name+', what do you do?');
        }
    }
}

var teacherQuestion = interviewQuestion('teacher');
var designerQuestion = interviewQuestion('designer');

teacherQuestion('John');
designerQuestion('John');
//Generic function
designerQuestion('Jane');
designerQuestion('Jim');
designerQuestion('Mike');

interviewQuestion('teacher')('Jane');
//left part [interviewQuestion('teacher')] return a functio
//passing parameter 'Jane' into the returned function
```

Immediately Invoked Function Expressions

```
/*
Immediately Invoked Function Expressions
        IIFE  (pattern)
*/

function game(){
    var score = Math.random() * 10;
    console.log(score >= 5);
}
game();

(function(){
    var score = Math.random() * 10;
    console.log(score >= 5);
})();

//console.log(score);  //here is error, since it cannot acc

(function(goodLuck){
    var score = Math.random() * 10;
    console.log(score >= 5 - goodLuck);
})(5);
```

2. function example

javaScript function

```
/**
 * Functions
 */
function calculateAge(birthYear){
    return 2019 - birthYear;
}
var ageJohn = calculateAge(1990);
var ageMike = calculateAge(1948);
var ageJane = calculateAge(1969);
console.log(ageJohn, ageMike, ageJane);

function yearsUntilRetirement(year, firstName){
    var age = calculateAge(year);
    var retirement = 65 - age;

    if(retirement > 0){
        console.log(firstName + ' retires in '+retirement+
    }else{
        console.log(firstName + ' is already retired.');
    }

}
yearsUntilRetirement(1990, 'John');
yearsUntilRetirement(1948, 'Mike');
yearsUntilRetirement(1990, 'Jane');
console.log("\n")
/**
 * Function Statements and Expressions
 */
var whatDoYouDo = function(job, firstName){
    switch(job){
        case 'teacher':
            return firstName + 'teachs kids how to code';
        case 'driver':
            return firstName + 'drives a cab in Lisbon';
        case 'designer':
            return firstName + 'designs beautiful websites
        default:
            return firstName + ' does something else';
    }
}
console.log(whatDoYouDo('teacher','John'));
console.log(whatDoYouDo('designer','John'));
console.log(whatDoYouDo('retired','John'));
console.log('\n')
```

3. Default Parameters

Default Parameters

```
function SmithPerson(firstName, yearOfBirth, lastName, nat
    this.firstName = firstName;
    this.lastName = lastName;
    this.yearOfBirth = yearOfBirth;
    this.nationality = nationality;
}

var john = new SmithPerson('John', 1990);
console.log(john)



//output:
SmithPerson {
  firstName: 'John',
  lastName: undefined,
  yearOfBirth: 1990,
  nationality: undefined
}
```

```
function SmithPerson(firstName, yearOfBirth, lastName, nat
    lastName === undefined ? lastName = 'Smith' : lastName

    nationality === undefined ? nationality = 'american' :
    this.firstName = firstName;
    this.lastName = lastName;
    this.yearOfBirth = yearOfBirth;
    this.nationality = nationality;
}

var john = new SmithPerson('John', 1990);
console.log(john)

//output:
SmithPerson {
  firstName: 'John',
  lastName: 'Smith',
  yearOfBirth: 1990,
  nationality: 'american'
}
```

ES6

```
//ES6
function SmithPerson(firstName, yearOfBirth, lastName = 'Sm
    this.firstName = firstName;
    this.lastName = lastName;
    this.yearOfBirth = yearOfBirth;
    this.nationality = nationality;
};
var john = new SmithPerson('John', 1990);
var emily = new SmithPerson('Emily', 1983, 'Diaz', 'spanish

console.log(john)
console.log(emily)


//output:
SmithPerson {
  firstName: 'John',
  lastName: 'Smith',
  yearOfBirth: 1990,
  nationality: 'american'
}
SmithPerson {
  firstName: 'Emily',
  lastName: 'Diaz',
  yearOfBirth: 1983,
  nationality: 'spanish'
}
```

4. coding challenging 1

```
/**
 * Coding Challenge 2:
 *
 * John and his family went on a holiday and went to 3 dif
 * The bills were $124, $48 and $268
 *
 * To tip the waiter a fair amount, John create a simple t
 * He likes to tip 20% of the bill when the bill is less th
 * 15% when the bill is between $50 and $200, and 10% if th
 *
 * In the end, John would like to have 2 arrays:
 * 1. Containing all three tips (one for ecah bill)
 * 2. Containing all three final paid amount (bill + tip).
 *
 * (NOTE: To calculate 20% of a value, simply mulitply if w
 */

 function tipCalculator(bill){
     var percentage;
     if(bill < 50){
         percentage = 0.2;
     }else if(bill >= 50 && bill < 200){
         percentage = 0.15;
     }else{
         percentage = 0.1;
     }
     return percentage * bill;
 }

 console.log(tipCalculator(100));

 var bills = [124, 48, 268];
 var tip = [tipCalculator(bills[0]),
            tipCalculator(bills[1]),
            tipCalculator(bills[2])];
var finalValues = [bills[0] + tips[0],
                   bills[1] + tips[1],
                   bills[2] + tips[2],]
console.log(tips);
```

4. coding challenging 2

```
/**
 * coding challenge 5
 *
 * Remember the tip calculator challenger? Let's create a r
 * This time, John and his family went to 5 different resta
 * The bills were $124, $48, $268, $180 and $42.
 * John likes to tip 20% of the bill when the bill is less
 * $50 and $200, and 10% if the bill is more than $200.
 *
 * Implement a tip calculator using objects and loops:
 * 1. Create an object with an array for the bill values
 * 2. Add a method to calculate the tip
 * 3. This method should include a loop to iterate over al
 * 4. As an output, create 1) a new array containing all t
 * and 2) an array containing final paid amounts(bill + tip
 * Hint: Start with two empty arrays [] as properties and t
 *
 * EXTRA AFTER FINISHING: Mark's family also went on a holi
 * The bills were $77, $375, $110, and $45.
 * Mark likes to tip 20% of the bill when the bill is less
 * $300 (different than John).
 *
 * 5. Implement the same functionality as before, this time
 * 6. Create a function (not a method) to calculate the ave
 * Hint: loop over the array, and in each iteration store t
 * After you have the sum of the array, divide it by the nu
 * (that's how you calculate the average)
 * 7. Calculate the average tip for each family
 * 8. Log to the console which family paid the highest tips
 */

var john = {
    fullName: 'John Smith',
    bills: [124, 48, 268, 42],
    calcTips: function(){
       this.tips = [];
       this.finalValues = [];

       for(var i = 0; i<this.bills.length; i++){
           // Determin percentage based on tipping rules
           var percentage;
           var bill = this.bills[i];

           if(this.bills[i] < 50){
               percentage = 0.2;
           }else if(this.bills[i] >= 50 && this.bills[i] <
               percentage = 0.15;
```

```
            }else{
                percentage = 0.1;
            }

            //Add results to the corresponding arrays
            this.tips[i] = bill * percentage;
            this.finalValues[i] = bill + bill * percentage;
        }
    }
}

var mark = {
    fullName: 'JMark Miller',
    bills: [77, 375, 110, 45],
    calcTips: function(){
        this.tips = [];
        this.finalValues = [];

        for(var i = 0; i<this.bills.length; i++){
            // Determin percentage based on tipping rules
            var percentage;
            var bill = this.bills[i];

            if(bill < 100){
                percentage = 0.2;
            }else if(bill >= 50 && bill < 300){
                percentage = 0.1;
            }else{
                percentage = 0.25;
            }

            //Add results to the corresponding arrays
            this.tips[i] = bill * percentage;
            this.finalValues[i] = bill + bill * percentage;
        }
    }
}

function calcAverage(tips){
    var sum = 0;
    for(var i = 0; i<tips.length; i++){
        sum = sum + tips[i];
    }
    return sum / tips.length;
}

// Do the calculations
john.calcTips();
```

```
mark.calcTips();


john.average = calcAverage(john.tips);
mark.average = calcAverage(mark.tips);
console.log(john, mark);

if(john.average > mark.average){
    console.log(john.fullName + '\'s family pays higher t
    + john.average);
}else if(mark.average > john.average){
   console.log(mark.fullName + '\'s family pays higher tip
   + mark.average);
}
```

3. Arrow function

```
// Arrow functions
const years = [1990, 1965, 1982, 1937];

// ES5
var ages5 = years.map(function(el){
    return 2019 - el;
});
console.log(ages5); //[29, 54, 37, 82]

// ES6
let ages6 = years.map(el => 2019 - el);
console.log(ages6); //[29, 54, 37, 82]

ages6 = years.map((el, index) => `Age elment ${index + 1}:
console.log(ages6);
//["Age elment 1: 29.", "Age elment 2: 54.", "Age elment 3

ages6 = years.map((el, index) => {
    const now = new Date().getFullYear();
    const age = now - el;
    return `Age elment ${index + 1}: ${2019-el}.`;
});
console.log(ages6);
//["Age elment 1: 29.", "Age elment 2: 54.", "Age elment 3
```

arraow function 2

```
var box5 = {
    color: 'green',
    position: 1,
    clickMe: function(){
        document.querySelector('.green').addEventListener(
            var str = 'This is box number '+ this.position
            this.color;
            alert(str);
        });
    }
}
box5.clickMe();
/* console:
This is box number undefined and it is undefined
 */
```

- The reason is that 'this' keyword, actually points to that object; but in the regular function call the this keyword will always point to the global object, which, in the case of the brower, is the window object
- However, a very common pattern to avoid this is to simply create a new variable in here and store the this variable in that new variable.

```
// ES5
var box5 = {
    color: 'green',
    position: 1,
    clickMe: function(){
        var self = this;
        document.querySelector('.green').addEventListener(
            var str = 'This is box number '+ self.position
            self.color;
            alert(str);
        });
    }
}
box5.clickMe(); //This is box number 1 and it is green
```

- we also can using Arrow function to avoid this:

```
// ES6
const box6 = {
    color: 'green',
    position: 1,
    clickMe: function(){
        document.querySelector('.green').addEventListener(
        () => {
            var str = 'This is box number '+ this.position
            this.color;
            alert(str);
        });
    }
}
box6.clickMe(); //This is box number 1 and it is green
```

ES5 this keyword

```
function Person(name){
    this.name = name;
}

// ES5
Person.prototype.myFriends5 = function(friends){
    var arr = friends.map(function(el)
    {
        return this.name + ' is friends with '+el;
    });
    console.log(arr);
}

var friends = ['Bob', 'Jane', 'Mark'];
new Person('John').myFriends5(friends);
/*  output:
0: " is friends with Bob"
1: " is friends with Jane"
2: " is friends with Mark"
 */
```

We have passed the 'John' into the Person constructor, but in ES5 .map function,

which, reference to the gloabl window object.

- To avoid this, outside .map function,

1. we can add **var self = this**
2. return **self.name + ...**

The second way to avoid this; but it sounds a little bit weird

```
// ES5
function Person(name){
    this.name = name;
}

Person.prototype.myFriends5 = function(friends){
    var arr = friends.map(function(el)
    {
        return this.name + ' is friends with '+el;
    }.bind(this));    //it sounds or looks a little bit wei
    console.log(arr);
}

var friends = ['Bob', 'Jane', 'Mark'];
new Person('John').myFriends5(friends);
/*  output:
0: "John is friends with Bob"
1: "John is friends with Jane"
2: "John is friends with Mark"
 */
```

ES6 Arrow function to avoid this:

```
//ES6
function Person(name){
    this.name = name;
}

Person.prototype.myFriends5 = function(friends){
    var arr = friends.map( el => {
        return this.name + ' is friends with '+el;
    });
    console.log(arr);
}
/*  output:
0: "John is friends with Bob"
1: "John is friends with Jane"
2: "John is friends with Mark"
 */
```

- OR:

```
Person.prototype.myFriends5 = function(friends){
    var arr = friends.map( el => `${this.name} is friends w
    console.log(arr);
}
```

4. block scope IIFE

block-scope

```
{
    const a = 1;
    let b = 2;
}
console.log(a + b); //error since ES6 block-scope

//ES5
(function(){
    var c = 3;
})();

console.log(c); //error, since IIFES also block-scoped
```

function-scope

```
{
    const a = 1;
    let b = 2;
    var c = 3;  //since var is function-scope
}
console.log(c);   //3
```

JS IIEF

https://www.cnblogs.com/shiyou00/p/10631013.html

```
function outputNumbers(count) {
    for (var i = 0; i < count; i++) {
        console.log(i);
    }

    console.log("**********total:");
    console.log(i);   //计数
}


outputNumbers(10);
```

JavaScript没有块级作用域的概念。但是通过IIEF 立即执行函数我们可以实

这个函数中定义了一个for循环，而变量i的初始值被设置为0。在Java、C++等
变量i只会在for循环的语句块中有定义，循环一旦结束，变量i就会被销毁。

可是在JavaScrip中，变量i是定义在ouputNumbers()的活动对象中的，
因此从它有定义开始，就可以在函数内部随处访问它。

即使像下面这样错误地重新声明同一个变量，也不会改变它的值.

```
function outputNumbers(count){
    for (var i=0; i < count; i++){
        console.log(i);
    }

    var i;          //重新声明变量
    console.log(i);       //计数
}
```

JavaScript从来不会告诉你是否多次声明了同一个变量；
遇到这种情况，它只会对后续的声明视而不见（不过，它会执行后续声明中的变
匿名函数可以用来模仿块级作用域并避免这个问题。

用作块级作用域（通常称为私有作用域）的匿名函数的语法如下所示。

```
(function(){
    //这里是块级作用域
})();
```

以上代码定义并立即调用了一个匿名函数。将函数声明包含在一对圆括号中，
表示它实际上是一个函数表达式。而紧随其后的另一对圆括号会立即调用这个函

现在对上面的函数进行改造

```
function outputNumbers(count){
    (function () {
        for (var i=0; i < count; i++){
            console.log(i);
        }
    })();
```

```
    console.log(i);    //导致一个错误!
}
```

在这个重写后的outputNumbers()函数中，我们在for循环外部插入了一个私有
在匿名函数中定义的任何变量，都会在执行结束时被销毁。因此，变量i只能在循

two more example

```
var Employee = (function () {
    function Employee(name, salary) {
        this.name = name;
        this.salary = salary;
    }
    Employee.prototype.display =
        function () {
            console.log(this.name);
        };
    return Employee;
})();
var emp = new Employee("Jon", 87321);
console.log(emp.salary);
```

```
var Employee = new function () {
    function Employee(name, salary) {
        this.name = name;
        this.salary = salary;
    }
    Employee.prototype.display =
        function () {
            console.log(this.name);
        };
    return Employee;
}
var emp = new Employee("Jon", 87321);
console.log(emp.salary);
```

Chapter8：Template String

1. delete operator

```
The JavaScript delete operator removes a property from an o
if no more references to the same property are held,
it is eventually released automatically.
```

```
const Employee = {
  firstname: 'John',
  lastname: 'Doe'
}

console.log(Employee.firstname);
// expected output: "John"

delete Employee.firstname;

console.log(Employee.firstname);
// expected output: undefined
```

- The `delete` operator removes a given property from an object. On successful deletion, it will return `true`, else `false` will be returned.

Ex:

```
var Employee = {
  age: 28,
  name: 'abc',
  designation: 'developer'
}

console.log(delete Employee.name);   // returns true
console.log(delete Employee.age);    // returns true

// When trying to delete a property that does
// not exist, true is returned
console.log(delete Employee.salary); // returns true
```

1. Template String

- ES5

```
let firstName = 'John';
let lastName = 'Smith';
const yearOfBirth = 1990;

function calcAge(year){
    return 2016-year;
}

//ES5
console.log('This is '+firstName+ ' '+lastName+' . He was b
yearOfBirth + '. Today, he is '+calcAge(yearOfBirth)+' yea
/* output:
 This is John Smith . He was born in 1990. Today, he is 26
 */

//ES5
console.log('This is '+firstName+ ' '+lastName+' . He was b
yearOfBirth + '. Today, he is '+calcAge(yearOfBirth)+' yea
/* This is John Smith . He was born in1990. Today, he is 2

//ES6
console.log(`This is ${firstName} ${lastName}. He was born
Today, he is ${calcAge(yearOfBirth)} years old.`);
/* This is John Smith. He was born in 1990.
Today, he is 26 years old. */


const n = `${firstName} ${lastName}`;
console.log(n.startsWith('J'));  //true
console.log(n.endsWith('th'));  //true
console.log(n.includes(' '));  //true
console.log(firstName.repeat(5)); //JohnJohnJohnJohnJohn
console.log(`${firstName} `.repeat(5)); //John John John J
```

Chapter9：异步编程 Asynchronous

Synchronous

```
const second = () => {
        console.log('Async Hey there')
}

const first = () => {
    console.log('Hey there');
    second();
    console.log("The end");
}
first();

/*
Hey there
Async Hey there
The end
 */
```

Asynchonous

```
const second = () => {
    setTimeout(() => {
        console.log('Async Hey there')
    }, 2000);
}

const first = () => {
    console.log('Hey there');
    second();
    console.log("The end");
}
first();


/*
Hey there
The end
Async Hey there
*/



//One more example:
cosnt image = document.getElementById('img').scr;

processLargeImage(image, () => {
    console.log('Image processed!')
})
```

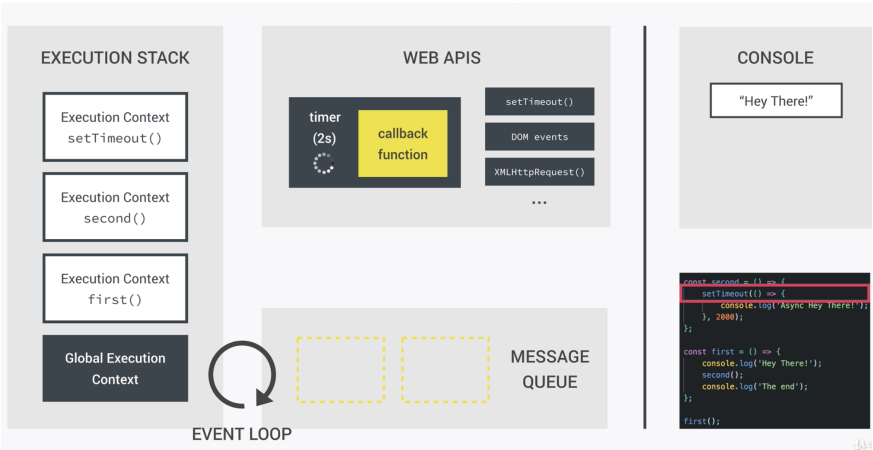
Understanding Asynchronous: The Event Loop

- Allow asynchronous functions to run in the "background";
- We pass in callbacks that run once the function has finished its work;
- Move on immediately: Non-blocking!(非阻塞)

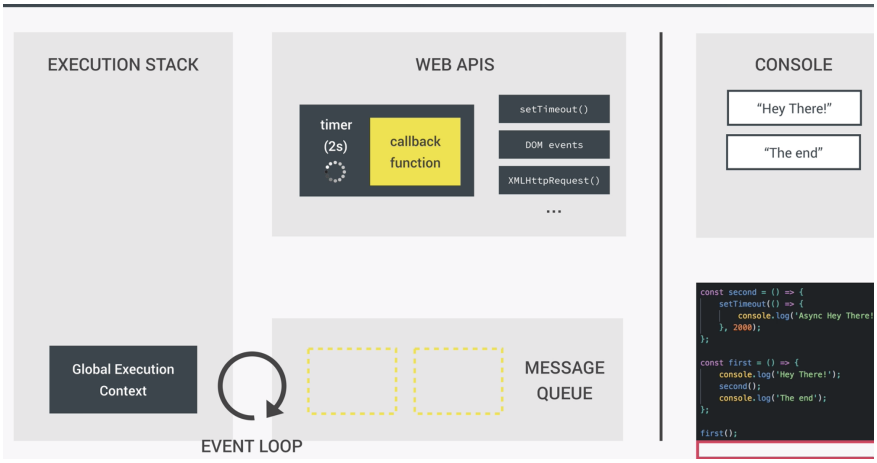
where does this setTimeout() function actually come from?

- It's part of something called the Web API
- which actually live outside the javaScript engine itself.

- Stuff like DOM manipulation methods, setTimeout, HTTP requests for AJAX, geolocation, local storage ..., actually live outside of the javaScript engine.
- we just access to them because they also in a javaScript runtime
- This is exactly where the timer will keep running for two seconds, asynchronously of course, so that our code can keep running without being blocked.
- so when we call the setTimeout(), the timer is created, together of course, with our callback function right inside the Web API environment. And there it keeps sitting until it finishes its work all in an Asychronous way.

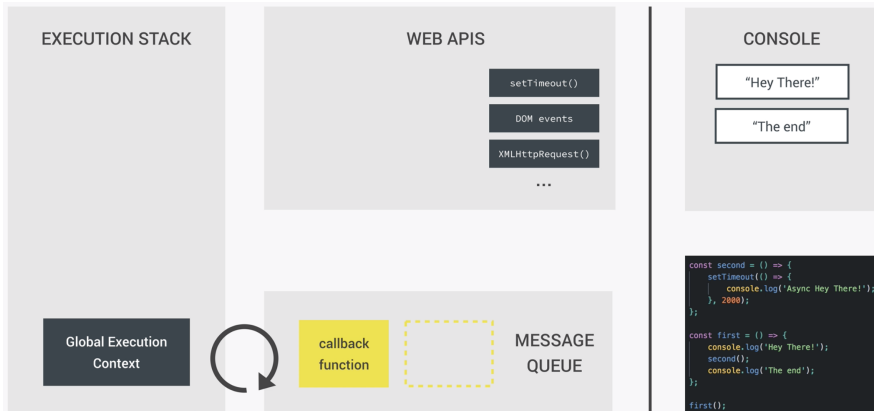


- Again, the callback function is not called right now, but instead it says attached to the timer until it finishes.
- And since the timer keeps working, basically in the background, we don't have to wait, and keep executing our code.
- Next up, the setTimeout() returns, pops off the stack and so does the Execution Context of the second function which now returns as well
- and we are back to the initial first function.
- now we just log `the end` to the console , and we give ourselves a new Execution Context, print the text to the console and pop the text off agian.
- Right now we have executed all our code in a Synchronous way, and have the timer run Asynchronously in the background.
- Now suppose that our two seconds have passed and the timer disappears.



But what happens to our callback function now?

- it simply moves to the Message Queue, where it waits to be executed as soon as the Execution stack is empty.



- so how are these callback functions in the Message Queue executed? finally, the Event loop comes in.
- The job of the Event Loop is to constantly **monitor** the Message Queue and the Execution Stack, and to push the first callback function in line onto the Execution Stack, as soon as the stack is empty.
- In our example here, right now the stack is empty, and we have one callback waiting to be executed. And the event loop takes the callback and pushes it onto the stack,

- now look one more example

```
const second = () => {
    setTimeout(() => {
        console.log('Async Hey there')
    }, 3000);
}

const another = () => {
    setTimeout(() => {
        console.log('Async another there')
    }, 2000);
}

const third = () => {
    console.log('third');
}
const forth = () => {
    console.log('forth');
}
const fifth = () => {
    console.log('fifth');
}
const sixth = () => {
    console.log('sixth');
}

const first = () => {
    console.log('Hey there');
    second();
    another();
    third();
    forth();
    fifth();
    sixth();
    console.log("The end");
}
first();

//output:
/*
Hey there
third
forth
fifth
sixth
The end
Async another there
Async Hey there
*/
```

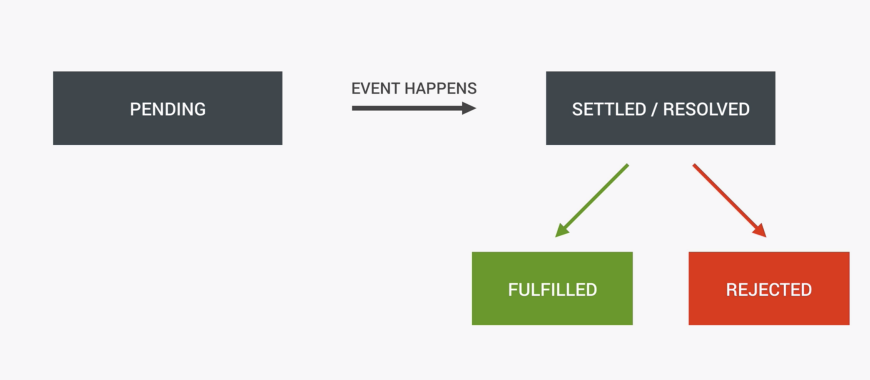
The Old Way: Asynchronous javaScript with Callbacks

```
function getRecipe() {
    setTimeout(() => {
        const recipeID = [523, 883, 432, 974];
        console.log(recipeID);

        setTimeout((id) => {
            const recipe = { title: 'Fresh tomato pasta', p
            console.log(`${id}: ${recipe.title}`);

            setTimeout(publisher => {
                const recipe2 = { title: 'Italian Piazza',
                console.log(recipe);
            }, 1500, recipe.publisher);

        }, 1500, recipeID[2]);

    }, 1500);
}
getRecipe();


        /*
            [523, 883, 432, 974]
            432: Fresh tomato pasta
            {title: "Fresh tomato pasta", publisher: "Jonas
        */
```

From Callback Hell to Promises

`Promise`

- Object that keeps track about whether a certain event has happened already or not;
- Determines what happens after the event has happened;
- Implements the concept of a future value that we're expecting



- testing resolve function

```
        /* Passing two callback functions: resolve , reject
        that's because this executor function here is used
        whether the event it is handing was successful or n
        And if it was successful we're going to call the re
        if not we call the reject function
        */
const getIDS = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve([523, 883, 432, 974]);
        //since setTimeout function is alawys finished, we

    }, 1500);
});

//Test handle then function
getIDS.then(IDs => {
    console.log(IDs);
}) //[523, 883, 432, 974]
    .catch(error => {
        console.log(error)
    });
    // It's impossible to fail, since setTimeout() always s
```

- testing reject function

```
        const getIDS = new Promise((resolve, reject) => {
            setTimeout(() => {
                reject([523, 883, 432, 974]);
                //since setTimeout function is alawys finis

            }, 1500);
        });

        getIDS
        .then(IDs => {
            console.log(IDs);
        })
        /*
Uncaught (in promise)
[523, 883, 432, 974]
Promise.then (async)
(anonymous)
        */

       //if we add a catch function here, wen can get the e
        .catch(error => {
            console.log('Error!!');
        });
        //Error!
```

example:

```
const getIDS = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve([523, 883, 432, 974]);
        //since setTimeout function is alawys finis

    }, 1500);
});

const getRecipe = (recID) => {
    return new Promise((resolve, reject) => {
        setTimeout(ID => {
            const recipe = {title: 'Fresh tomato pa
            console.log(`${ID}: ${recipe.title}`);
        }, 1500, recID);
    });
};

getIDS
.then(IDs => {
    console.log(IDs);
    return getRecipe(IDs[2]);
})
.then(recipe => {
    console.log(recipe);
})
.catch(error => {
    console.log('Error!!');
});

/*
[523, 883, 432, 974]
432: Fresh tomato pasta
*/
```

example:

```
const getIDS = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve([523, 883, 432, 974]);
        //since setTimeout function is alawys finished, we

    }, 1500);
});

const getRecipe = (recID) => {
    return new Promise((resolve, reject) => {
        setTimeout(ID => {
            const recipe = { title: 'Fresh tomato pasta', p
            resolve(`${ID}: ${recipe.title}`);
        }, 1500, recID);
    });
};

const getRelated = (publisher) => {
    return new Promise((resolve, reject) => {
        setTimeout((pub) => {
            const recipe = { title: 'Italian Piazza', publi
            resolve(`${pub}: ${recipe.title}`);
        }, 1500, publisher);
    });
}


getIDS
    .then(IDs => {
        console.log(IDs);
        return getRecipe(IDs[2]);
    })
    .then(recipe => {
        console.log(recipe);
        return getRelated('Jonas');
    })
    .then(recipe => {
        console.log(recipe);
    })
    .catch(error => {
        console.log('Error!!');
    });

/*
[ 523, 883, 432, 974 ]
432: Fresh tomato pasta
Jonas: Italian Piazza
*/
```

Async/Await

```
The async function declaration defines an asynchronous func
```

```
//Async/Await
const getIDS = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve([523, 883, 432, 974]);
        //since setTimeout function is alawys finished, we

    }, 1500);
});
const getRecipe = (recID) => {
    return new Promise((resolve, reject) => {
        setTimeout(ID => {
            const recipe = { title: 'Fresh tomato pasta', p
            resolve(`${ID}: ${recipe.title}`);
        }, 1500, recID);
    });
};
const getRelated = (publisher) => {
    return new Promise((resolve, reject) => {
        setTimeout((pub) => {
            const recipe = { title: 'Italian Piazza', publi
            resolve(`${pub}: ${recipe.title}`);
        }, 1500, publisher);
    });
}

async function getRecipeAW() {
    const IDs = await getIDS;
    console.log(IDs);
    const recipe = await getRecipe(IDs[2]);
    console.log(recipe);
    const related = await getRelated('Jonas Schmedtamnn');
    console.log(related);

}

getRecipeAW();

/*
[ 523, 883, 432, 974 ]
432: Fresh tomato pasta
Jonas Schmedtamnn: Italian Piazza
*/
```

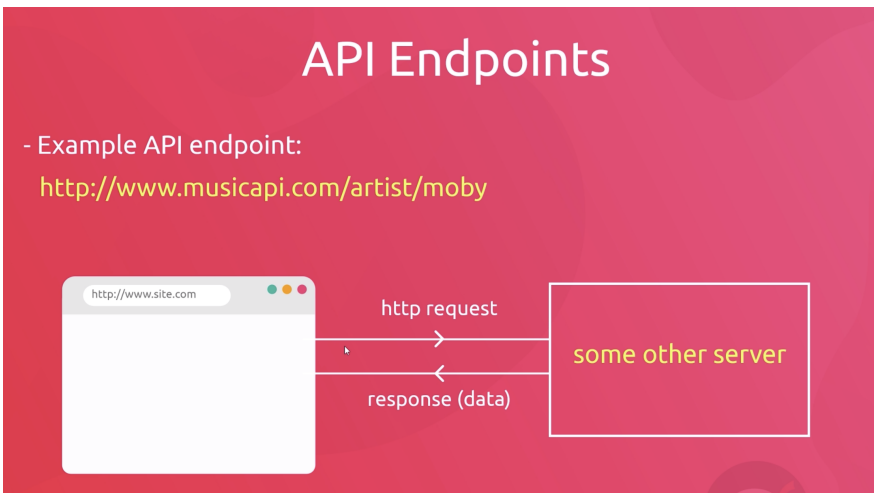
the last step is equal to:

```
async function getRecipeAW() {
    const IDs = await getIDS;
    console.log(IDs);
    const recipe = await getRecipe(IDs[2]);
    console.log(recipe);
    const related = await getRelated('Jonas Schmedtamnn');
    console.log(related);

    return recipe;
}

getRecipeAW().then((result) => console.log(`${result} it th

/*
[ 523, 883, 432, 974 ]
432: Fresh tomato pasta
Jonas Schmedtamnn: Italian Piazza
432: Fresh tomato pasta it the best ever!
*/
```

Ninja version

`What are HTTP Requests ?`

- Make HTTP requests to get data from anohter server
- We make these request to API endpoints



what is API ?

- `Application Programming Interface`
- APIs Make Life Easier for Developers

- APIs Control Access to Resources

Now there's many different APIs that we can use to get data

- Twitter Youtube Instagram Spotify and loads more and each API is going to have its own set of endpoints that we make request to for data.
- You could even make your own API using the service side language such as `Python` or even `JavaScript` if you're using `Node.js`
- Once we make a request to an endpoint from the browser we typically `get back a selection of data in a format called JSON and Json is a format which looks very much like javascript objects`

so the API that we're going to be using to practice with is:

JSONPlaceholder API

- this allows us to play around with API endpoints and to get back some JSON data

`HTTPRequst && Response Status`

```
const getTodos = () => {
    const request = new XMLHttpRequest();

    request.addEventListener('readystatechange', () => {
        // console.log(request, request.readyState);
        if (request.readyState === 4 && request.status ===
            console.log(request, request.responseText);
        } else if (request.readyState === 4) {
            console.log('could not fetch the data');
        }
    });

    request.open('GET', 'https://jsonplaceholder.typicode.c
    request.send();
}

getTodos();
```

callback function

```
const getTodos = (callback) => {
    const request = new XMLHttpRequest();

    request.addEventListener('readystatechange', () => {
        // console.log(request, request.readyState);
        if (request.readyState === 4 && request.status ===
            callback(undefined, request.responseText);
        } else if (request.readyState === 4) {
            callback('could not fetch data', undefined);
        }
    });

    request.open('GET', 'https://jsonplaceholder.typicode.c
    request.send();
}


console.log(1)
console.log(2)
getTodos((error, data) => {
    console.log('callback is fired.');
    if (error) {
        console.log(error);
    } else {
        console.log(data);
    }
});
console.log(3)
console.log(4)




/*
1
2
3
4
callback is fired.
[
  {
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
  },
    ...
```

```
        ...
            ...
*/
```

JSON data

- what is JSON data?

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
  },
  {
    "userId": 1,
    "id": 2,
    "title": "quis ut nam facilis et officia qui",
    "completed": false
  },


// Remember my vscode setting.json data?
{
    "workbench.colorTheme": "Material Theme",
    "editor.minimap.enabled": false,
    "window.zoomLevel": 1,
    "terminal.integrated.fontSize": 13,
    // "terminal.integrated.fontFamily": "Meslo LG S DZ for
    "terminal.integrated.fontFamily": "Roboto Mono for Powe
    "breadcrumbs.enabled": false,
    "[javascript]": {
        "editor.defaultFormatter": "vscode.typescript-langu
        "editor.formatOnType": true,
        "editor.formatOnSave": true
    },
    "workbench.iconTheme": "material-icon-theme"
}
```

- The `JSON.parse()` method parses a `JSON string`, constructing the JavaScript value or object described by the string.

```
const getTodos = (callback) => {
    const request = new XMLHttpRequest();

    request.addEventListener('readystatechange', () => {
        // console.log(request, request.readyState);
        if (request.readyState === 4 && request.status ===
            const data = JSON.parse(request.responseText);
            callback(undefined, data);
        } else if (request.readyState === 4) {
            callback('could not fetch data', undefined);
        }
    });

    request.open('GET', 'https://jsonplaceholder.typicode.c
    request.send();
}


console.log(1)
console.log(2)
getTodos((error, data) => {
    console.log('callback is fired.');
    if (error) {
        console.log(error);
    } else {
        console.log(data);
    }
});
console.log(3)
console.log(4)
```

we can also create our JSON data as well

todos.json

```
[
    {"text": "play mariokart", "author": "shaun"},
    {"text": "buy some bread", "author": "Mario"},
    {"text": "do the plumming", "author": "Luigi"}
]
```

```
//create our JSON data
const getTodos = (callback) => {
    const request = new XMLHttpRequest();

    request.addEventListener('readystatechange', () => {
        if (request.readyState === 4 && request.status ===
            const data = JSON.parse(request.responseText);
            callback(undefined, data);
        } else if (request.readyState === 4) {
            callback('could not fetch data', undefined);
        }
    });

    request.open('GET', 'todos.json');
    request.send();
}

getTodos((error, data) => {
    console.log('callback is fired.');
    if (error) {
        console.log(error);
    } else {
        console.log(data);
    }
});
```

2. 异步编程 Asynchronous

```
1. call back

2. generator

3. promise

4. async + await
```

`Promise`

- 主要用于异步计算
- 可以将异步操作队列化，按照期望的顺序执行，返回符合预期的结果
- 可以在对象之间传递和操作promise， 帮助我们处理队列

javascript 包含大量异步操作

- javascript 为检查表单而生
- 创造它的首要目标是操作dom
- 所以，javascript 的操作大多都是异步的

为什么异步操作可以避免界面冻结?

- 假设你去一家饭店，自己找座位坐下了，然后招呼服务员拿菜单
- 服务员说：“对不起，我是 `同步` 服务员，我要服务完这张桌子才能招呼你。”
- 你是不是很生气很想抽ta?
- 那一桌人明明已经吃上了，你只是想要菜单，这么小的一个动作，服务员却要你等到别人的一个大动作完成。
- 这就是 `同步` 的问题：
- `顺序交付的工作1234， 必须按照1234的顺序完成。`

异步

- 异步，则是将耗时很长的 a 交付的工作交给系统之后， 就去继续做 b 交付的工作。 等到系统完成前面的工作之后， 再通过回调或者事件，继续做 a 剩下的工作。

- 从观察者的角度来看起来，a b 工作的完成顺序，和交付他们的事件顺序无关，所以叫 `异步` 。

稍有不慎，就会踏入 `callback hell`

`除此之外，还有更深层次的问题`

- 假设需求：
- 遍历目录，找出最大的一个文件

以下才是Promise诞生的原因：

```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

回调有四个问题

1. 嵌套层次很深，难以维护
2. 无法正常使用return 和 throw
3. 无法正常检索堆栈信息
4. 多个回调之间难以建立联系

浏览器中的javascript

- 异步操作以事件为主
- callback主要出现在 ajax 和 file api
- 这个时候问题尚不严重

有了node.js之后

`对异步的依赖进一步加剧了。。。`

- 无阻塞高并发，是node.js的招牌

- 异步操作是其保障
- 大量操作依赖callback function

Promise的概念和优点

【优点】

- Promise是一个代理对象，它和原先要进行的操作并无关系
- Promise通过引入一个回调，避免了更多的回调

【状态】

- `pending`：待定，称为初始状态
- `fulfilled`：实现，称为操作成功状态
- `rejected`：被否决，称为操作失败状态
- 当Promise状态发生改变的时候，就会触发.then()里的响应函数来处理后续步骤
- Promise状态已经改变，就不会再变

Promise的基本语法

```
new Promise(
    /* 执行器 executor */
    function (resolve, reject) {
      // 一段耗时很长的异步操作
      resolve(); // 数据处理完成
      reject(); // 数据处理出错
    }
 ).then(function A() {
    // 成功，下一步
  }, function B() {
    // 失败，做相应处理
  });
```

Promise一个简单的例子

```
console.log('here we go');
new Promise(resolve => {
    setTimeout(() => {
      resolve('hello');
      console.log(123);
    }, 2000);
  })
  .then(name => {
    console.log(name + ' world');
  });

/*
here we go
123
hello world
*/
```

- 以上代码和课程稍微有些不同，目的是和定时器做一些对比，以此发现一点什么。

```
console.log('here we go');
setTimeout(() => {
  greeting("hello");
  console.log(123);
}, 2000)

function greeting(name) {
  console.log(name + ' world');
}

/*
here we go
hello world
123
*/
```

- 通过以上两段代码的运行结果比较，可以浅显的得出：**resolve()状态引发的then()是异步的**

Promise两步执行的范例

```
console.log('here we go');
new Promise(resolve => {
    setTimeout(() => {
      resolve('hello');
    }, 2000);
  })
  .then(value => {
    console.log(value);
    return new Promise(resolve => {
      setTimeout(() => {
        resolve('world');
      }, 2000);
    });
  })
  .then(value => {
    console.log(value + ' world');
  });
```

- 这个范例主要是简单的演示了Promise如何解决回调地狱这个让人头大的问题。

对已经完成的Promise执行then()

```
console.log('start');
let promise = new Promise(resolve => {
  setTimeout(() => {
    console.log('the promise fulfilled');
    resolve('hello, world');
  }, 1000);
});

setTimeout(() => {
  promise.then(value => {
    console.log(value);
  });
}, 3000);

/*
start
the promise fulfilled
hello, world

*/
```

- 讲师的原话：这段代码展示了Promise作为队列这个重要的特性，就是说我们在任何一个地方生成了一个Promise对象， 都可以把它当做成一个变量传递到其他地方执行。不管Promise前面的状态到底有没有完成，队列都会按照固定的顺序去执行。

then()不返回Promise

```
console.log('here we go');
new Promise(resolve => {
    setTimeout(() => {
      resolve('hello');
    }, 2000);
  })
  .then(value => {
    console.log(value);
    console.log('everyone');
    (function () {
      return new Promise(resolve => {
        setTimeout(() => {
          console.log('Mr.Laurence');
          resolve('Merry Xmas');
        }, 2000);
      });
    }());
    return false;
  })
  .then(value => {
    console.log(value + ' world');
  });

/*
here we go
hello
everyone
false world
Mr.Laurence
*/
```

- 我对以上代码的理解是这样的：最后一个then()方法里的value值代表的是上一个then()里的返回值， 当没有return的时候，默认返回值为undefined。而resolve()里的数据为什么没被调用呢？因为上一个then()方法里return的是false而不是Promise实例。

then()解析

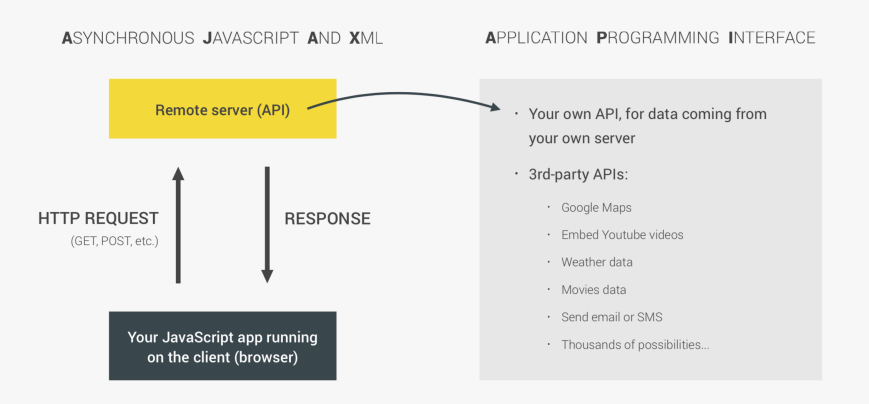
- then()接受两个函数作为参数，分别代表fulfilled和rejected

- then()返回一个新的Promise实例，所以它可以链式调用
- 当前面的Promise状态改变时，then()根据其最终状态，选择特定的状态响应函数执行
- 状态响应函数可以返回新的Promise或其他值
- 如果返回新的Promise，那么下一级then()会在新的Promise状态改变之后执行
- 如果返回其他任何值，则会立刻执行下一级then()

3. AJAX

AJAX and API

- ASYNCHRONOUS JAVASCRIPT AND XML



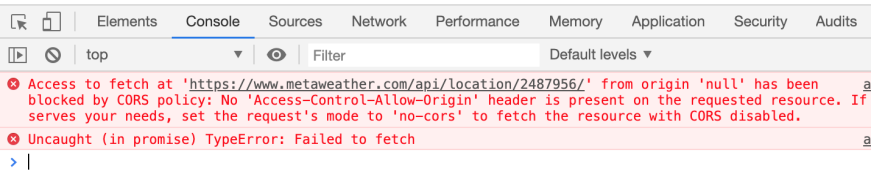
- what is API ?
- `Application Programming Interface`
- APIs Make Life Easier for Developers
- APIs Control Access to Resources
- we first open a website: metaweather

metaweather

Making AJAX Calls with Fetch and Promises

```
fetch('https://www.metaweather.com/api/location/2487956/');
```

- we will see a problem:

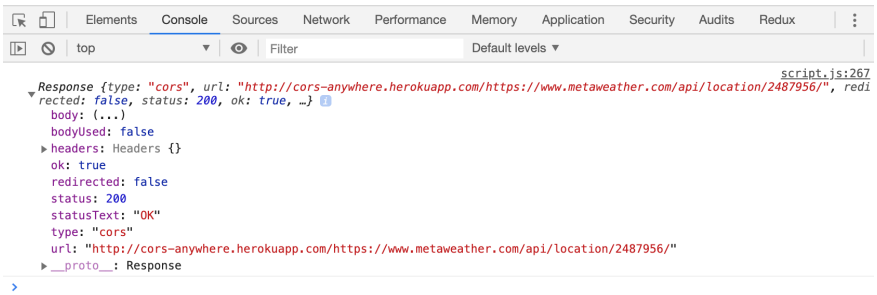


- but we can solve this issue:

`fetch('http://cors-anywhere.herokuapp.com/https://www.metaweather.com/api/location/2487956/');`

- Now there is no issue on the develop console

```
fetch
    ('http://cors-anywhere.herokuapp.com/https://www.metawe
    .then((result) => {
        console.log(result);
    })
    .catch(error => {
        console.log(error);
    });
```




```
fetch
    ('http://cors-anywhere.herokuapp.com/https://www.metawe
    .then((result) => {
        console.log(result);
        return result.json();
        //here will return a promise because this actually
        //this conversion to JSON,
    })
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.log(error);
    });
```

```
Response {type: "cors", url: "http://cors-anywhere.herokuapp.com/https://www.metaweather.com/api/location/2487956/", redirected: false, status: 200, ok: true, …}
  body: (...)
  bodyUsed: true
  headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: "OK"
  type: "cors"
  url: "http://cors-anywhere.herokuapp.com/https://www.metaweather.com/api/location/2487956/"
  __proto__: Response
```

script.js:273

```
{consolidated_weather: Array(6), time: "2019-07-25T22:01:06.441705-07:00", sun_rise: "2019-07-25T06:07:19.786412-07:00", sun_set: "2019-07-25T20:25:26.337851-07:00", timezone_name: "LMT", …}
  consolidated_weather: (6) [{…}, {…}, {…}, {…}, {…}, {…}]
  latt_long: "37.777119, -122.41964"
  location_type: "City"
  parent: {title: "California", location_type: "Region / State / Province", woeid: 2347563, latt_long: "37.271881,-119.270233…
  sources: (7) [{…}, {…}, {…}, {…}, {…}, {…}, {…}]
  sun_rise: "2019-07-25T06:07:19.786412-07:00"
  sun_set: "2019-07-25T20:25:26.337851-07:00"
  time: "2019-07-25T22:01:06.441705-07:00"
  timezone: "US/Pacific"
  timezone_name: "LMT"
  title: "San Francisco"
  woeid: 2487956
  __proto__: Object
```

```
>
```

```
fetch
    ('http://cors-anywhere.herokuapp.com/https://www.metawe
    .then((result) => {
        console.log(result);
        return result.json();
        //here will return a promise because this actually
        //this conversion to JSON,
    })
    .then(data => {
        // console.log(data);
        const today = data.consolidated_weather[0];
        console.log(today);
    })
    .catch(error => {
        console.log(error);
    });
```

```
//Making AJAX Calls with Fetch and Promises
fetch
    ('http://cors-anywhere.herokuapp.com/https://www.metaweather.com/api
    .then((result) => {
        console.log(result);
        return result.json();
        //here will return a promise because this actually happens asyn
        //this conversion to JSON,
    })
    .then(data => {
        // console.log(data);
        const today = data.consolidated_weather[0];
        console.log(today);
    })
    .catch(error => {
        console.log(error);
    });
```

Elements | Console | Sources | Network | Performance | Memory | Application

top | Filter | Default levels

```
Response {type: "cors", url: "http://cors-anywhere.herokuapp.com/https://www.metawea
d: false, status: 200, ok: true, …}

{id: 6571207929364480, weather_state_name: "Showers", weather_state_abbr: "s", wind_
7-26T03:05:16.536387Z", …}
  air_pressure: 1014.9100000000001
  applicable_date: "2019-07-25"
  created: "2019-07-26T03:05:16.536387Z"
  humidity: 79
  id: 6571207929364480
  max_temp: 20.45
  min_temp: 13.870000000000001
  predictability: 73
  the_temp: 20.89
  visibility: 12.655467072297782
  weather_state_abbr: "s"
  weather_state_name: "Showers"
  wind_direction: 207.95935179030775
  wind_direction_compass: "SSW"
  wind_speed: 5.182408042021263
  __proto__: Object
>
```

```
fetch
    ('http://cors-anywhere.herokuapp.com/https://www.metawe
    .then((result) => {
        console.log(result);
        return result.json();
        //here will return a promise because this actually
        //this conversion to JSON,
    })
    .then(data => {
        // console.log(data);
        const today = data.consolidated_weather[0];
        console.log(`Temperatures in ${data.title} stay bet
    })
    .catch(error => {
        console.log(error);
    });

/*
Temperatures in San Francisco stay between 13.870000000000
*/
```

`updating our function`

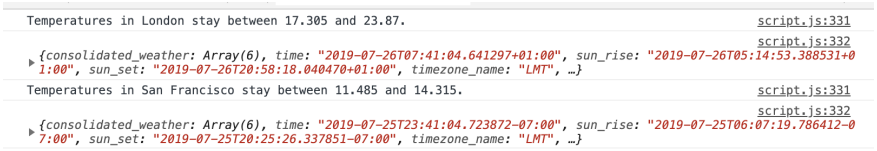
```
function getWeather(woeid) {
    fetch
        (`http://cors-anywhere.herokuapp.com/https://www.me
        .then((result) => {
            console.log(result);
            return result.json();
            //here will return a promise because this actua
            //this conversion to JSON,
        })
        .then(data => {
            // console.log(data);
            const today = data.consolidated_weather[0];
            console.log(`Temperatures in ${data.title} stay
        })
        .catch(error => {
            console.log(error);
        });
}
getWeather(2487956);
getWeather(44418); //London ID

/*
Temperatures in San Francisco stay between 13.870000000000
script.js:299 Temperatures in London stay between 21.08 and
*/
```

The above is a very simple AJAX call using `fetch`, and promises using `then` and the `catch` methods

`Making AJAX Calls with Fetch and Async/Await`

- `async` function always return a promise

```
async function getWeatherAW(woeid) {

    try {
        const result =
            await fetch(`http://cors-anywhere.herokuapp.cor
        const data = await result.json();
        const tomorrow = data.consolidated_weather[1];
        console.log(`Temperatures in ${data.title} stay bet
        console.log(data);
    } catch (error) {
        console.log(error);
    }
}
getWeatherAW(2487956);
getWeatherAW(44418); //London ID
```



Chapter10：let const

1. let const

```
// let and const

// ES5
var name5 = 'Jane Smith';
var age5 = 23;
name5 = 'Jane Miller';  //mutate
console.log(name5);

// ES6
const name6 = 'Jane Smith';
let age6 = 23;
name6 = 'Jane Miller';
console.log(name6);  //error, since name6 is const
```

scope between ES5 and ES6

```
//ES5
function driversLicence5(passedTest){
    if(passedTest){
        var firstName = 'John';
        var yearOfBirth = 1990;

    }
    console.log(firstName + ', born in '+ yearOfBirth +
        ' is now officially allowed to drive a car.');
}
driversLicence5(true);
/* output:
John, born in 1990 is now officially allowed to drive a ca

// ES6
function driversLicence6(passedTest){
    if(passedTest){
        let firstName = 'John';
        const yearOfBirth = 1990;

    }
    console.log(firstName + ', born in '+ yearOfBirth +
        ' is now officially allowed to drive a car.');
}
driversLicence6(true);
/* output:
script.js:34 Uncaught ReferenceError: firstName is not def
    at driversLicence6 (script.js:34)
    at script.js:37
 */
```

- look at the above code:
    1. ES5: since var defined in function scope, firstName and yearOfBirth still working.
    2. ES6: let and const defined in block scope so outside if statement the fistName and yearofBirth don't work.
- so we change the ES6 codes:

```
// ES6
function driversLicence6(passedTest){
    let firstName ;
    const yearOfBirth = 1990;
    if(passedTest){
        firstName = 'John';
    }
    console.log(firstName + ', born in '+ yearOfBirth +
        ' is now officially allowed to drive a car.');
}
driversLicence6(true);
/*
John, born in 1990 is now officially allowed to drive a ca
 */
```

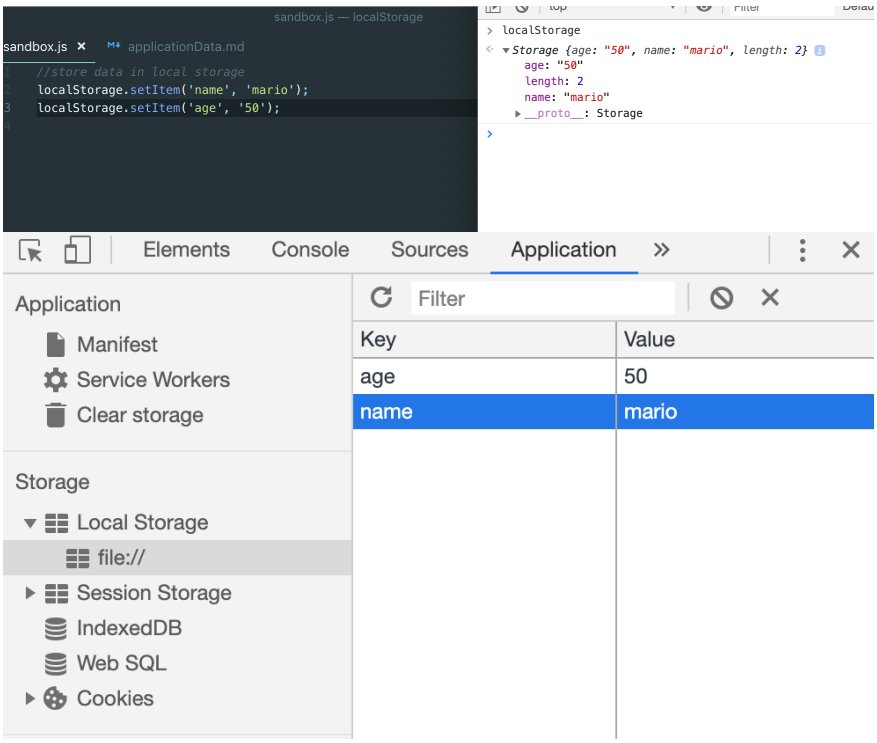
- again, since ES6 is block-Scoped, outter for loop i still be 23

```
let i = 23;
for(let i=0; i<5; i++){
    console.log(i)
}
console.log(i)
/*  output:
 0
 1
 2
 3
 4
 23
*/
```

2. local storage

What is Local Storage?

- Set up a database to store & retrieve data
- Use local storage to store and retrieve data
- Remember: `local Storage` is on the `Window` object so we could print out window the local storage



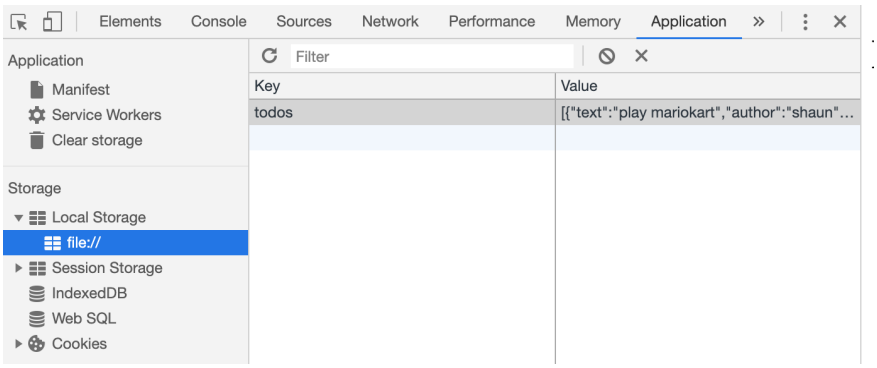
```
//store data in local storage
localStorage.setItem('name', 'mario');
localStorage.setItem('age', '50');


//get data from local storage
let name = localStorage.getItem('name')
let age = localStorage.getItem('age')
console.log(name, age);  //mario 50


//updating data
localStorage.setItem('name', 'luigi');
localStorage.age = '40'

name = localStorage.getItem('name');
age = localStorage.getItem('age')
console.log(name, age);//luigi 40
```

delete data from local storage

```
//store data in local storage
localStorage.setItem('name', 'mario');
localStorage.setItem('age', '50');


//get data from local storage
let name = localStorage.getItem('name')
let age = localStorage.getItem('age')
console.log(name, age);


//deleting data from local storage
// localStorage.removeItem('name');
localStorage.clear();
name = localStorage.getItem('name');
age = localStorage.getItem('age');
console.log(name, age)
//output: null, null
```

Stringifying & Parsing Data

```
const todos = [
    { text: 'play mariokart', author: 'shaun' },
    { text: 'buy some milk', author: 'mario' },
    { text: 'buy some bread', author: 'luigi' }
];


// console.log(JSON.stringify(todos))
localStorage.setItem('todos', JSON.stringify(todos))
```



- print to `JSON` data format

```
const stored = localStorage.getItem('todos')

console.log(stored);

//output:
[
    {"text":"play mariokart","author":"shaun"},
    {"text":"buy some milk","author":"mario"},
    {"text":"buy some bread","author":"luigi"}
]
```

```
console.log(JSON.parse(stored));

//output

    { text: 'play mariokart', author: 'shaun' }
    { text: 'buy some milk', author: 'mario' }
    { text: 'buy some bread', author: 'luigi' }
```

Chapter11：Map

1. maps

```
//Data Structure: Maps

const question = new Map();
question.set('question', 'What is the official name of the
question.set(1, 'ES5')
question.set(2, 'ES6')
question.set(3, 'ES2015')
question.set(4, 'ES7')
question.set('correct', 3);
question.set(true, 'Correct answer:D');
question.set(false, 'Wrong, please try again!');
/* input: question
0: {"question" => "What is the official name of the lateste
1: {1 => "ES5"}
2: {2 => "ES6"}
3: {3 => "ES2015"}
4: {4 => "ES7"}
5: {"correct" => 3}
6: {true => "Correct answer:D"}
7: {false => "Wrong, please try again!"}

*/

console.log(question.get('question'));//What is the officia
console.log(question.size) //8

if(question.has(4)){
    //question.delete(4)
    console.log('Answer 4 is here.')
}
//question.clear();


question.forEach((value, key) =>
    console.log(`This is ${key}, and it's set to ${value}`)
);
/*
This is question, and it's set to What is the official name
This is 1, and it's set to ES5
This is 2, and it's set to ES6
This is 3, and it's set to ES2015
This is 4, and it's set to ES7
This is correct, and it's set to 3
This is true, and it's set to Correct answer:D
This is false, and it's set to Wrong, please try again!
*/

for(let [key, value] of question.entries()){
```

```
    console.log(`This is ${key}, and it's set to ${value}`)
}/* same result like the previous results */
```

- map.js

```
const numbers = [0, 1, 2, 3, 4];

let newNumbers = [];

for (let i = 0; i < numbers.length; i++) {
    newNumbers.push(numbers[i] * 10);
}

console.log(numbers);
console.log(newNumbers);
```

```
[Done] exited with code=0 in 0.123 seconds

[Running] node "/Users/Git/gitJsSyntax/maps_ES6/maps.js"
[ 0, 1, 2, 3, 4 ]
[ 0, 10, 20, 30, 40 ]
```

2. Destructuring 解构赋值

```
// ES5
var john = ['John', 26];
//var name = john[0];
//var age = john[1];

// ES6
const [name, age] = ['John', 26];
console.log(name)
console.log(age)

const obj = {
    firstName: 'John',
    lastName: 'Smith'
};
const {firstName, lastName} = obj;
console.log(firstName); //John
console.log(lastName);  //Smith

const {firstName: a, lastName: b} = obj;
console.log(a);//John
console.log(b);//Smith

function calcAgeRetirement(year){
    const age = new Date().getFullYear() - year;
    return [age, 65-age];
}

const [age2, retirement] = calcAgeRetirement(1990);
console.log(age2);//29
console.log(retirement);//36
```

3. Rest parameters

```
//ES5
function isFullAge5(){
    console.log(arguments);
}
isFullAge5(1990, 1999, 1965)
/*
0: 1990
1: 1999
2: 1965
callee: ƒ isFullAge5()
length: 3
Symbol(Symbol.iterator): ƒ values()
__proto__: Object
*/

function isFullAge5(){
    var argsArr = Array.prototype.slice.call(arguments);

    argsArr.forEach(element => {
        console.log((2016 - element) >= 18);
    });
}
isFullAge5(1990, 1999, 1965); // true false true
isFullAge5(1990, 1999, 1965, 2016, 1987); // true false tru
```

ES6

```
function isFullAge6(...years){
    years.forEach(element => console.log((2016 - element) >
}

isFullAge6(1990, 1999, 1965); // true false true
```

1. Recursion

2. Search Algorithm

3. Bubble Sort

4. Selection Sort

5. Insertion Sort

6. Merge Sort

7. quick Sort

8. Radix Sort

Singly LinkedList && Big-O

- a stupid way

```
class Node{
    constructor(val){
        this.val = val;
        this.next = null;
    }
}

var first = new Node('Hi');
first.next = new Node('there, ');
first.next.next = new Node('how');
first.next.next.next = new Node('are');
first.next.next.next.next = new Node('you?');

while(first.next){
    console.log(first.val);
    first = first.next;
}
```

implement Singly LinkedList; Pushing

`Adding a new` **node** `to the end of the Lined List`

`Pushing pseudocode`

- This function should accept a value
- Create a new node using the value passed to the function
- if there is no head property on the list, set the head and tail to be the newly created node
- Otherwise set the next property on the tail to be the new node and set the tail property on the list to be the newly created node

```
// piece of data - val
// reference to next node - next
class Node{
    constructor(val){
        this.val = val;
        this.next = null;
    }
}

class SinglyLinkedList{
    constructor(){
        this.head = null;
        this.tail = null;
        this.length = 0;
    }
    //adding a push function to implement the functioning:
    push(val){
        const node = new Node(val);
        if(!this.head){
            this.head = node;
            this.tail = node;
        }else{
            this.tail.next = node;
            this.tail = node;
        }
        this.length++;
        return this;
    }
    traverse(){
        var current = this.head;
        while(current){
            console.log(current.val);
            current = current.next;
        }
    }
}
var list = new SinglyLinkedList();
list.push('Hello');
list.push('World');
list.push('!');
list.traverse();

//output:
Hello
World
!
```

Popping

`Removing a` **node** `from the end of the Linked List`

- adding a pop() function

```
    pop(){
        if(!this.head){ return undefined;}
        var current = this.head;
        var tempTail = current;
        while(current.next){
            tempTail = current;
            current = current.next;
        }
        console.log(current.val);
        console.log(tempTail.val);
    }
```

- if we call list.pop()

```
/*
!
World
*/
// since the when current.next == null, stop wihlie loop, t
// tempTail stop at penult. The current stop at the last no
```

- updating pop() function

```
    pop(){
        if(!this.head){ return undefined;}
        var current = this.head;
        var tempTail = current;
        while(current.next){
            tempTail = current;
            current = current.next;
        }
/*
        hello -> World -> !
                        curr
                temp
*/
        this.tail = tempTail;
        this.tail.next = null;
        this.length--;
        if(this.length == 0){
            this.head = null;
            this.tail = null;
        }
        return current;
    }


var list = new SinglyLinkedList();
list.push('Hello');
list.push('World');
list.push('!');
list.traverse();
// console.log(list.pop().val);
// console.log(list);
list.pop();
list.pop();
list.pop();
console.log(list);

/* output: */
Hello
World
!
SinglyLinkedList { head: null, tail: null, length: 0 }
```

Shifting pseudocode

- If there are no nodes, return undefined

- Store the current head property in a variable
- Set the head property to be the current head's next property
- Decrement the length by 1
- Return the value of the node removed

```
    shift() {
        if (!this.head) return undefined;
        var currentHead = this.head;
        this.head = currentHead.next;
        this.length--;
        return currentHead;
    }

list.shift();
console.log(list)
list.shift();
console.log(list)
list.shift();
console.log(list)


//output:

SinglyLinkedList {
  head: Node { val: 'World', next: Node { val: '!', next: n
  tail: Node { val: '!', next: null },
  length: 2
}
SinglyLinkedList {
  head: Node { val: '!', next: null },
  tail: Node { val: '!', next: null },
  length: 1
}
SinglyLinkedList { head: null, tail: null, length: 0 }
```

Unshifting (insertFirst)

- Adding a new node to the beginning of the Linked List!
- This function should accept a value
- Create a new node using the value passed to the function

- if there is no head property on the list, set the head and tail to be newly created node
- Othewise set the newly created node's next property to be the current head property on the list.
- Set the head property on the list to be that newly created node
- Increment the length of the list by 1
- Return the linked list

```
    unshift(val){
        var newNode = new Node(val);
        if(!head){
            this.head = newNode;
            this.tail = this.head;
        }else{
            newNode.next = this.head;
            this.head = newNode;
        }
            this.length++;
            return this;
            // return this linked list !

    }

list.unshift("1");
list.unshift("2");
list.unshift("3");
console.log(list);

//output:

SinglyLinkedList {
  head: Node { val: '3', next: Node { val: '2', next: [Node
  tail: Node { val: '1', next: [Circular] },
  length: 3
}
```

Getter

- Retrieving a **node** by its position in the Linked List

`Get pseudocode:`

- This function should accept an index

- if the index is less than zero or greater than or queal to the length of the list, return null
- Loop through the list until you reach the index and return the node at the specific index

```
    get(index){
        if(index<0 || index>=this.length){
            return null;
        }
        var counter = 0;
        var current = this.head;
        while(counter !== index){
            current = current.next;
            counter++;
        }
        return current;
    }
list.push("Hello");
list.push("Wolrd");
list.push("!");
list.push(":)");
list.push(":(");
var ptr = list.get(4);
console.log(ptr.val);

//output:

:(
```

Set Value

`Changing the value of a node based on its position in the Linked List`

`Set pseudocode:`

- This function should accept a value and an index
- Use your get function to find the specific node.
- If the node is not found, return false
- if the node is found, set the value of that node to be the value passed to the function and return true

```
    set(index, value){
        var foundNode = this.get(index);
        if(foundNode){
            foundNode.val = value;
            return true;
        }
        return false;
    }

list.set(0, "!!!");
console.log(list.get(0).val);
list.set(1, "***");
console.log(list.get(1).val);
list.set(2, "😄");
console.log(list.get(2).val);
list.set(3, "👽");
console.log(list.get(3).val);

//output:

!!!
***
😄
👽
```

Insert

`Adding a node to the Linked List at a specific position`

`Insert pseudocode:`

- if the index is less than zero or greater than the length, return false
- if the index is the smae as the length, push a new node to the end of the list
- if the index is 0, unshift(insertFirst) a new node to the start of the list
- Otherwise, using the **get** method, access the node at the index -1
- Set the next property on that node to be the new node
- Set the next property on the new node to be the previous next
- Increment the length
- Return true

```
    insert(index, value){
        if(index<0 || index>this.length){
            return false;
        }
        if(index === this.length){
            return this.push(value);
        }
        if(index === 0){
            return this.unshift(value);
        }
        var newNode = new Node(value);
        var prev = this.get(index-1);
        var temp = prev.next;
        prev.next = newNode;
        newNode.next = temp;
        this.length++;
        return true;
    }

list.push(100);
list.push(201);
list.push(250);
list.push(350);
list.insert(1, "😄 ");
console.log(list.get(1).val);
list.insert(1, "火星人");
console.log(list.get(1).val);
console.log(list.length); //6
list.insert(6, "大前端！ ");
console.log(list.get(6).val);
console.log(list.insert(8, "🍉 ")); //false

//output:
😄
火星人
6
大前端！
false
```

Remove

`Removing a node from the linked list at a` **specific** position

`Remove pseudocode`

- if the index is less than zero or greater then the length, return undefined
- if the index is the same as the length-1, pop
- if the index is 0, shift
- Otherwise, using the **get** method, access the node at the index-1
- Set the next property on that node to be the next of the next node
- Decrement the length
- Return the value of the node removed

```
    remove(index){
        if(index<0 || index>= this.length){
            return undefined;
        }
        if(index === 0){
            return this.shift(index);
        }
        if(index === this.length-1){
            return this.pop();
        }
        var previousNode = this.get(index - 1);
        var removed = previousNode.next;
        previousNode.next = removed.next;
        this.length--;
        return removed;
    }

list.push(100);
list.push(201);
list.push("😄 ");
list.push("火星人");
list.push("大前端! ");
list.push("🍉 ");
list.push("💻 ");
console.log(list.length);
console.log(list.remove(0).val);
console.log(list.remove(0).val);
console.log(list.remove(0).val);
console.log(list.remove(0).val);
console.log(list.remove(0).val);
console.log(list.remove(0).val);
console.log(list.remove(0).val);
console.log(list.remove(0)); //undefined


// output:
7
100
201
😄
火星人
大前端!
🍉
💻
undefined
```

Reverse

`Reversing the linked list` **in place!**

`Reverse pseudocode`

- swap the head and tail
- create a variable called next
- create a variable called prev
- create a variable called node and initialize it to the head property
- Loop through the list
- set next to be the next property on whatever prev is
- set prev to be the value of the node variable
- set the node variable to be the value of the next variable

`firsly, adding a print method`

```
    print(){
        var arr = [];
        var current = this.head;
        while(current){
            arr.push(current.val);
            current = current.next;
        }
        console.log(arr);
    }
```

`implementing the reverse function`

```
    reverse(){
         var node = this.head;
         this.head = this.tail;
         this.tail = node;

         var next;
         var prev= null;
         for(var i=0; i<this.length; i++){
             next = node.next;
             node.next = prev;
             prev = node;
             node = next;
         }
         return this;
     }

list.push(100);
list.push(201);
list.push("😄 ");
list.push("火星人");
list.push("大前端! ");
list.push("🍉 ");
list.push("💻 ");
list.reverse();
list.print();

// output:
[
  '💻 ',
  '🍉 ',
  '大前端! ',
  '火星人',
  '😄 ',
  201,
  100
]
```

Big O

- Insertion **O(1)**
- Removal **It depends...O(1) or O(n)**
- Searching **O(n)**
- Access **O(n)**

10. Doubly LinkedList

11. Stack && Queue

12. Binary Search Tree

What is a tree?

`A data structure that consists of nodes in a parent/child relationship`

- Lists - linear
- Trees - nonlinear
- A Singly Linked List sort of a special case of a tree

`Tree Terminology`

- Root - The top node in a tree
- Child - A node directly connected to another node when moving away from the Root
- Parent - The converse notion of a child.
- Siblings - A group of nodes with the same parent.
- Leaf - A node with no child
- Edge - The connection between one node and another

`Trees Lots of different applications`

- Html DOM
- Network routing
- Abstract Syntax Tree
- Artficial Intelligence
- Folders in Operating Systems
- Computer File Systems

`Kinds of Trees`

- Trees

- Binary Tress
- Binary Search Trees

`How BSTS work?`

- Every parent node has at most **two** children
- Every node to the left of a parent node is **always less** than the parent
- Every node to the right of a parent node is **always greater** than the parent

`The BinarySearchTree class`

```
class BinarySearchTree{
    constructor(){
        this.root = null;
    }
}

class Node{
    constructor(value){
        this.value = value;
        this.left = null;
        this.right = null;
    }
}
```

Ch 1：firbase

1. firebase-ta

FIREBASE AND FIRESTORE NOTES

QUERY REFERENCE AND QUERY SNAPSHOT

- Firestore library gives us one of two different types of potential objects
    1. Query Reference
    2. Query Snapshot
- A query is a request we make to firestore to give us something from the database
- Firestore returns us two types of objects: references and snapshots Of these objects, they can either be documents or collection versions.
- Firestore will always return us these objects, even if nothing exists at from that query

QUERY REFERENCE

- A QueryReference is simply an object that represent the current place in the database that we are querying. We get them by calling either
-

`firestore.doc('/users/:userId')`

`firestore.collection('/users')`

- The query reference object does not have the actual data of the collection or document. It instead has properties that tell us details about it, or the method to get the snapshot object which gives us the data we are looking for.

Document Reference VS Collection Reference

- We use documentRef objects to perform our CRUD methods (create, read , update, delete). The documentRef methods are

```
.set()
.get()
.update()
.delete()
```

- We can also add documents to collections using the collectionRef object using the .add() method

```
collectionRef.add({//value:prop})
```

- We get the snapshotObject from the referenceObject using .get() method i.e documentRef.get() or collectionRef.get()
- documentRef returns a documentSnapshot object.
- collectionRef returns a querySnapshot object
- exists property on the snapshot object tells us whether there is any data there or not.

DOCUMENT SNAPSHOT

- We get a documentSnapshot object from our documentReference object The documentReference object allows us to check if a document exists at this query using the .exists property which returns a boolean. We can also get the actual properties on the object by calling .data() method which return us a JSON object of the document.

FIREBASE AUTH

- Firebase Authentication for Firebase backed applications
- Client(Browser) and Server
- From the frontend we connect to the firebase backend via the Firebase SDK and that allows us to communicate with all the Firebase Services that we need from the frontend of the application

Firebase Authentication

- How Authentication Works ! (Theory Point of View)
- For eg we have some kind of form in our application that may be the Login Form or the Registration Form and we capture user credentials using that form then they are sent securely to the server via the login method or the registration method provided by Firebase SDK.Then on the server, Firebase validates those credentials and

send back an authentication token to the browser and then we can access data in our browser from this token for example name or email of the user that got just loggedin or registered. Now when the requests are made to the firebase server after login for example to the firestore or the cloud functions or something like that then this token is sent along for the ride and these are the services they have access to that information on them too.

- When some request is made to change some kind of data in the firestore, Firestore will be able to look at the auth token of that request and then protect data based on that user token.We can protect sensitive database data from any user who is not an admin and we can tell that from the token that comes along for the ride
- Once the user is logged in to our application then the firebase persists the users state so that means refreshing the page does not mean that they are logged out they will be still be loggedin
- All the Server Side Configuration that you want to enable is done using Firebase Console. This will provide us all the features like Cloud Functions or the Firestore database that is used

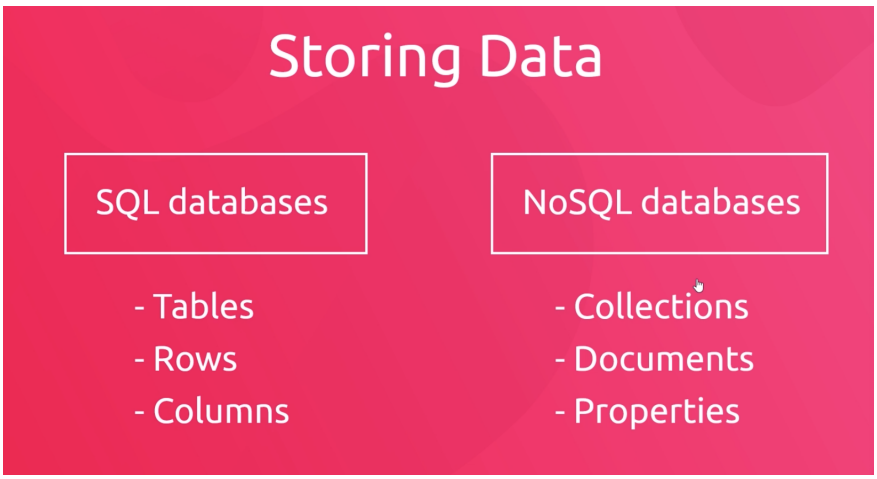
```
STEP 1 : Set up the SignIn method (by default it is not set

STEP 2 :
Provider                              Status
Email/Password                        Enable

Enable the provider Email/Password to the Enable Status
This is for making the users loggedin to our application
This is the email and password authentication that we can
configure using this Step 2

We want to add a new user not from the Firebase console
but from the frontend of our application

FIRESTORE
Firebase Database which is the Cloud Firestore database

We have two kind of databases here
1. Real time Firebase which is the older one
2. The newer version is the Firestore which is awesome as
well

STEP 3 : Go to the database option in the Develop Section

STEP 4 : Click on the Create Database button

STEP 5 : Check the Start in Test Mode radio button and
click enable (This is for initial development phase)
This will let us to easily interact with the database
without having to worry about the security rules
In the locked mode authentication is required for reading
and writing the data.Once the application is done,we will
change the mode to the locked mode.

STEP 6: We can create collection either using the Firestore
or through our application.

UI AND FIRESTORE SECURITY RULES
Firestore security rules where only the authenticated users
allowed access to certain resources like create guides and

Custom Claims in Firebase and Firestore applications

Firebase Authentication
UserId
Email
CUSTOM CLAIMS :
    admin : true
```

```
    premium : true

By using the concept of custom claims, we can ensure that c
that have special privileges will be able to create guides
can only view them if you have the admin privileges then or
the guides.

Client(Browser)

user has admin claim ?

If yes, show the admin UI
If no, show the normal UI

Server Side

If user has admin claim ?
If yes, allow access for data.
If no, block the data access

Cloud Functions
Cloud functions run on the server
Good for code you do not want to expose on the client
Perform tasks not available to client users
Callable from the frontend of the application

Firebase cloud functions are really very cool
they allow to run some code on the serverside only
Even though the cloud functions run on the server we can ca
the frontend of the application and if we have the required
and want to make a user an admin, then we can attach our cu
to our cloud function
```

2. Database Firebase

`Storing Data`

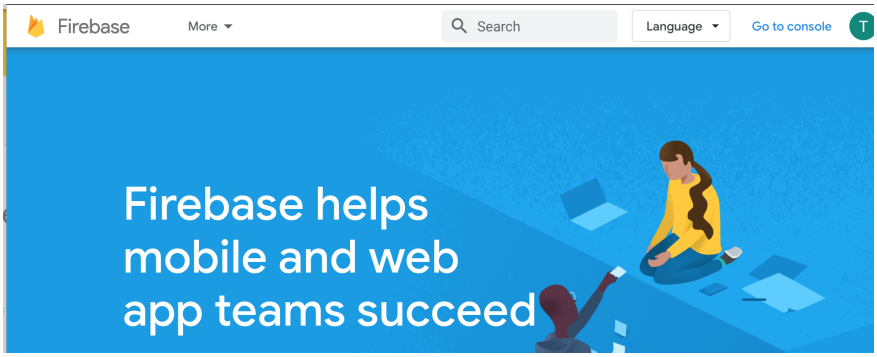
- Websites work with data
  - blog post, todos, comments, user info, scores etc
- We can store this data in a database
  - Firebase(by Google)



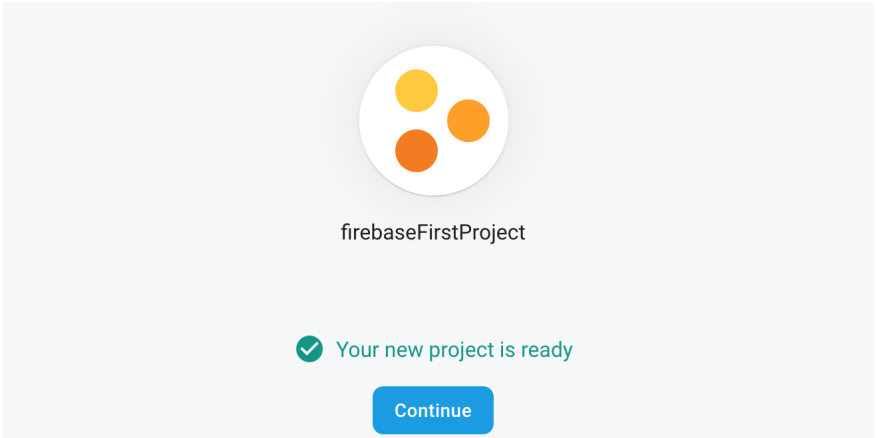
`Firebase && Firestores`

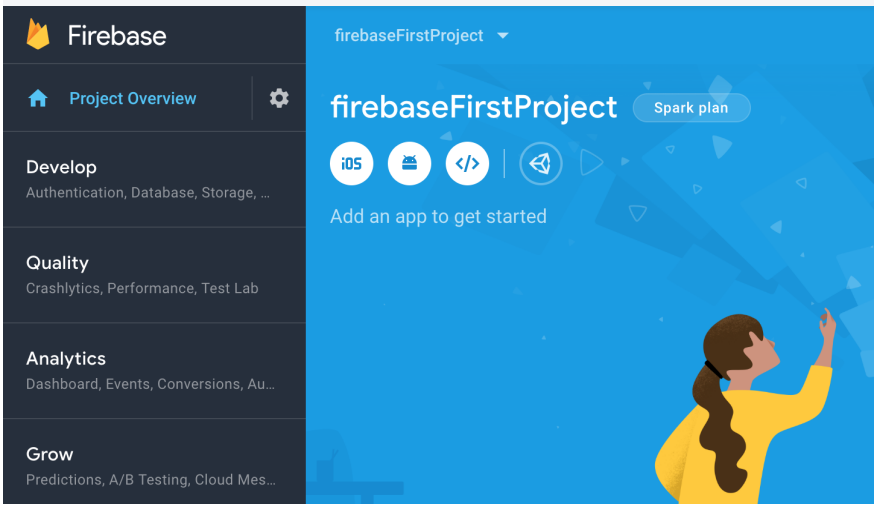
- `Firebase` is known as a backend as a service
- It provides us with a complete backend solution to all of our Websites
- meaning as front end developers we don't need to worry about setting up a server to do things like interact with a database or run server.
- firebase is an out of the box service which can help us do all of this without having to set up our own server.
- And the best thing about this is that it's completely free for somall applications and Websites.

create a new account

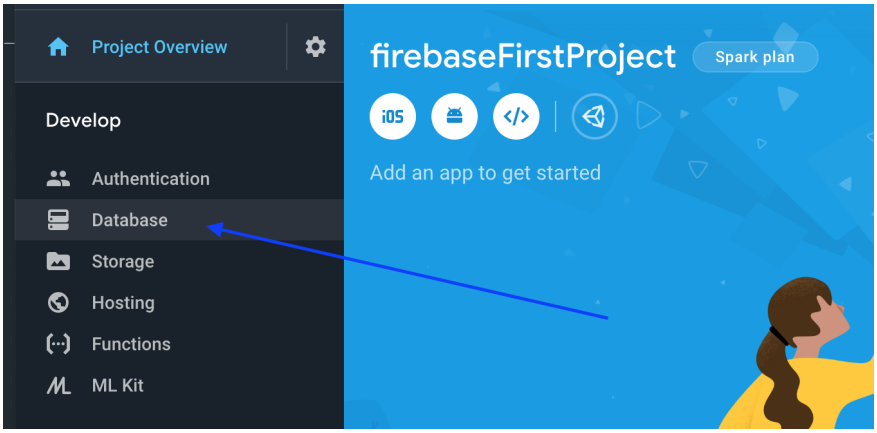


- click `Go to console`
- create a new project





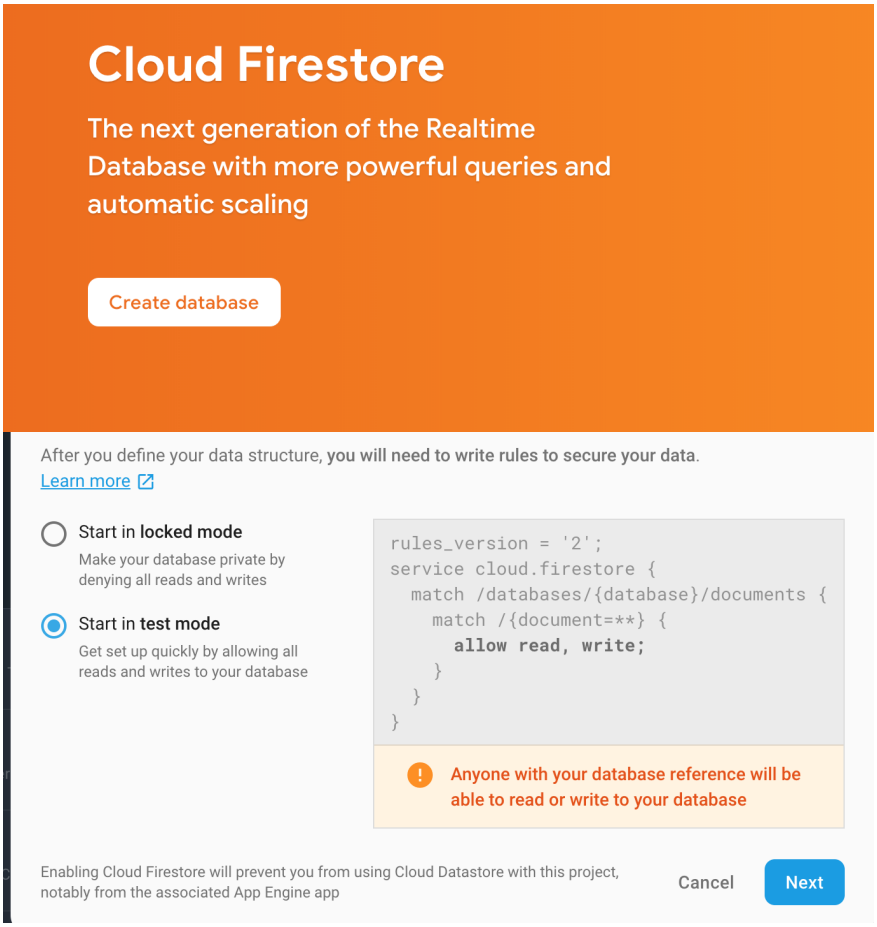
- so this area is the control center for the back end of our Website.
- now we can do a lot of set up
- but we focus on `Database` , click `Database`



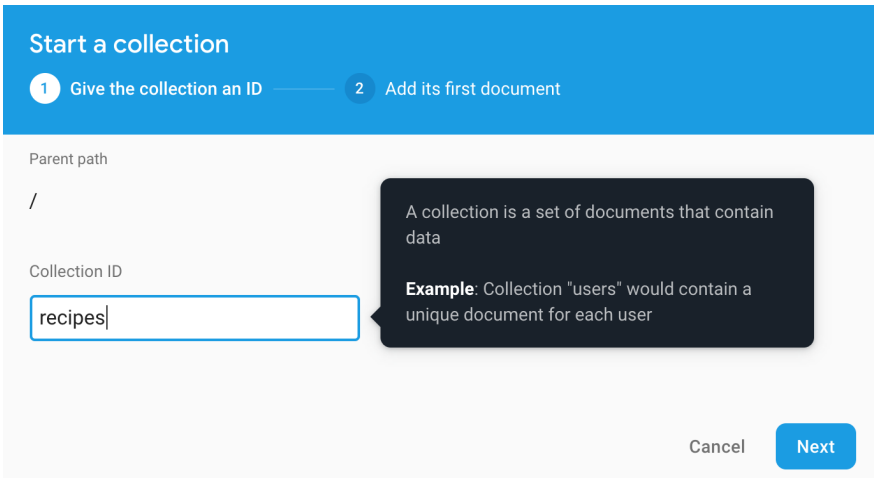
Note:

- if we want to learn more about firebase as a whole and the different features, go to YouTube channel.

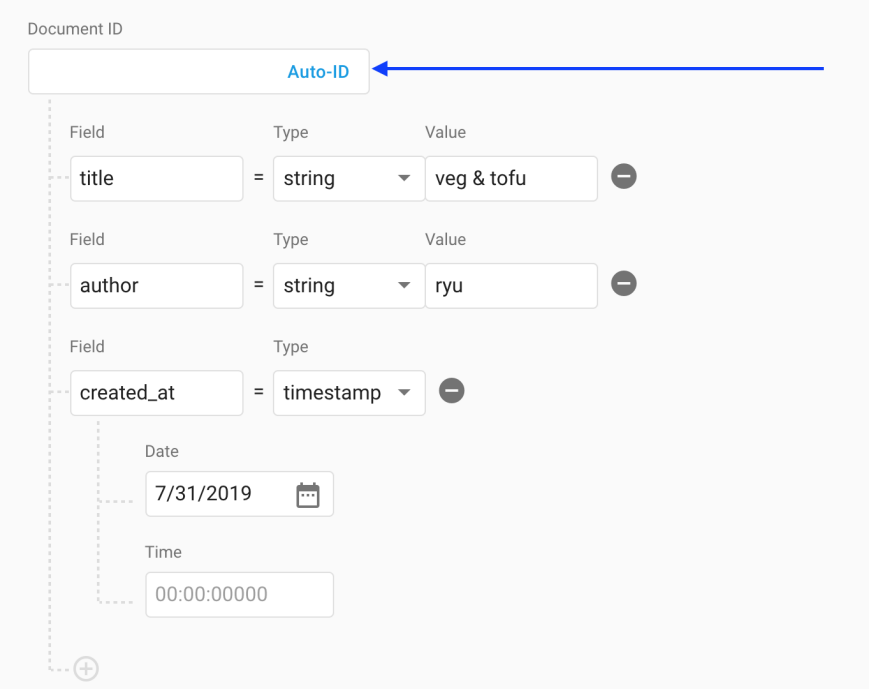
- create a new fire store database
- Now firebase actually comes with two types of database:
  1. `Firestore` , which we're going to be using. And that is the new type of database.
  2. the older `real time database` as well which is still good.
- now, `click` :



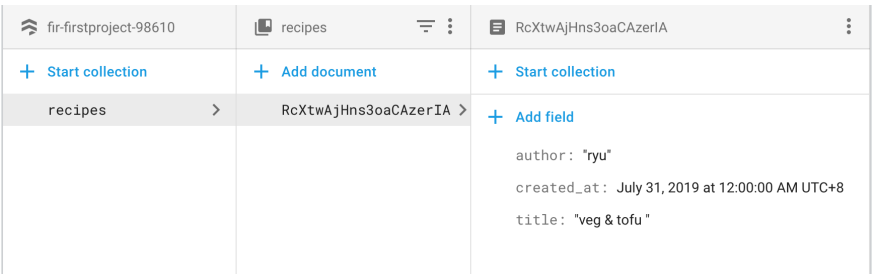
- click `Start collection`



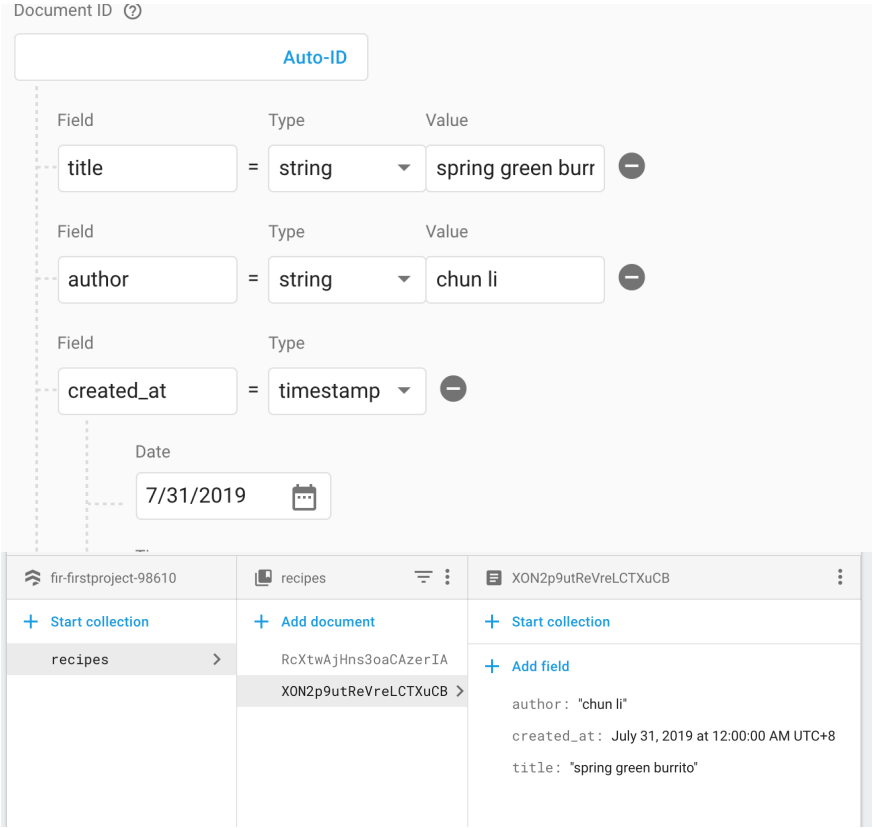
- Now remember documents represent single records and they're very much like `javaScript objects`
- They have `key value pairs`



- see this picture, for `Document ID` , I'm goint to leave because if we leave this blank `firestore` is going to *automatically* generate a unique idea for us.
- Remember: every `Document` in `firestore` has a unique idea and we can use those ideas later on to go out and query the database and get documents.



- now we add another `Document`



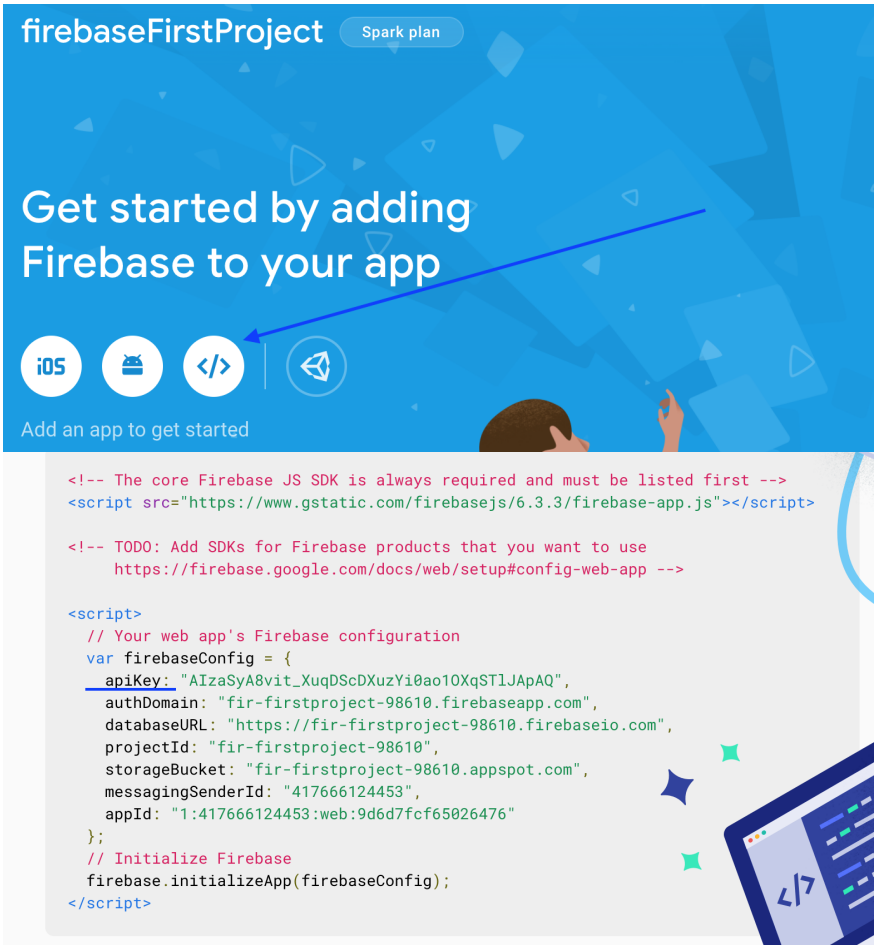
- now we have our first two `documents` inside that `collection`

`connecting to firebase`

- The next logical step that it would be to connect to this database and the backend in general from the front end Our web site in the browser
- to do that, we need to copy a little snippet of code from firebase from the back end into the front end of our Web site.
- And that's goint to allow us to connect to this backend and use services like the firebase firestore database

---

- Now go to `Project Overview`
- click this icon:



- look this piece of codes, this `API key` is a bit like an indentifier, it tells the front end of our website which backend to connect to over.
- we need to `copy` all of this

`index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initia
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Databases</title>
</head>
<body>
  <h1>Databases (Firebase)</h1>
  <!-- The core Firebase JS SDK is always required and must
  <script src="https://www.gstatic.com/firebasejs/6.3.3/fi

  <!-- TODO: Add SDKs for Firebase products that you want t
     https://firebase.google.com/docs/web/setup#config-web-

  <script>
    // Your web app's Firebase configuration
    var firebaseConfig = {
      apiKey: "AIzaSyA8vit_XuqDScDXuzYi0ao1OXqSTlJApAQ",
      authDomain: "fir-firstproject-98610.firebaseapp.com",
      databaseURL: "https://fir-firstproject-98610.firebase
      projectId: "fir-firstproject-98610",
      storageBucket: "fir-firstproject-98610.appspot.com",
      messagingSenderId: "417666124453",
      appId: "1:417666124453:web:9d6d7fcf65026476"
    };
    // Initialize Firebase
    firebase.initializeApp(firebaseConfig);
  </script>

  <script src="sandbox.js"></script>
</body>
</html>
```

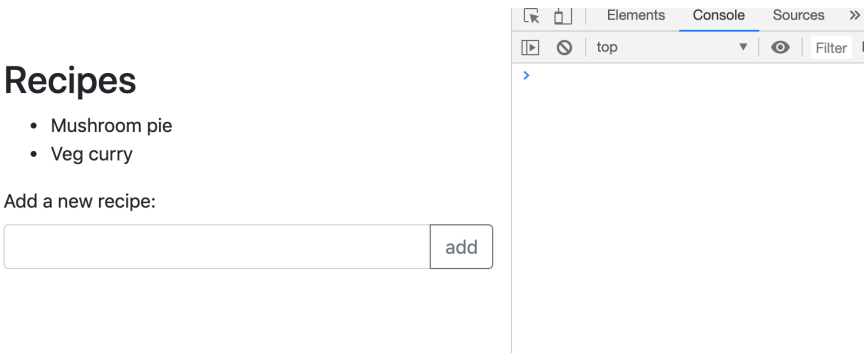
- there's only one more thing to do here and that is to get a reference to our database or to initialize the database.

`import a bootstramp library`

`updating the .html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initia
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="https://stackpath.bootstra
  <title>Databases</title>
</head>
<body>
  <div class="container my-5">
    <h2>Recipes</h2>
    <ul>
      <li>Mushroom pie</li>
      <li>Veg curry</li>
    </ul>
    <form>
      <label for="recipe">Add a new recipe:</label>
      <div class="input-group">
        <input type="text" class="form-control" id="recipe'
        <div class="input-group-append">
          <input type="submit" value="add" class="btn btn-c
        </div>
      </div>
    </form>
  </div>

  <!-- The core Firebase JS SDK is always required and must
  <script src="https://www.gstatic.com/firebasejs/5.8.4/fir
  <script src="https://www.gstatic.com/firebasejs/5.8.4/fir

  <!-- TODO: Add SDKs for Firebase products that you want t
     https://firebase.google.com/docs/web/setup#config-web-

  <script>
    // Your web app's Firebase configuration
    var firebaseConfig = {
      apiKey: "AIzaSyA8vit_XuqDScDXuzYi0ao1OXqSTlJApAQ",
      authDomain: "fir-firstproject-98610.firebaseapp.com",
      databaseURL: "https://fir-firstproject-98610.firebase
      projectId: "fir-firstproject-98610",
      storageBucket: "fir-firstproject-98610.appspot.com",
      messagingSenderId: "417666124453",
      appId: "1:417666124453:web:9d6d7fcf65026476"
    };
    // Initialize Firebase
    firebase.initializeApp(firebaseConfig);
    const db = firebase.firestore();
```

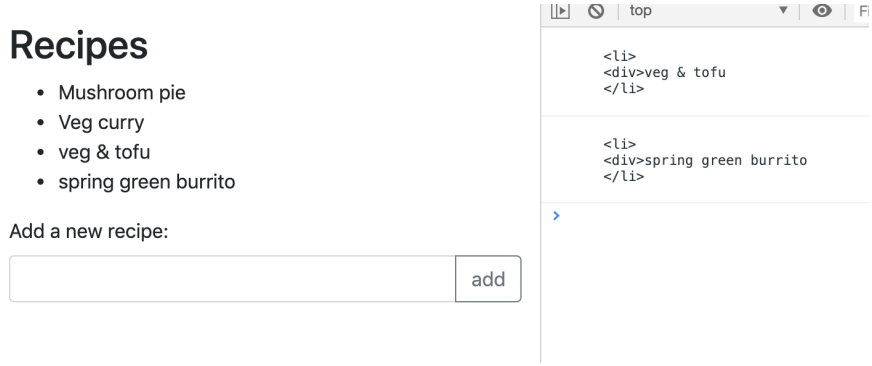
```
  </script>

  <script src="sandbox.js"></script>
</body>

</html>
```



Getting Collections

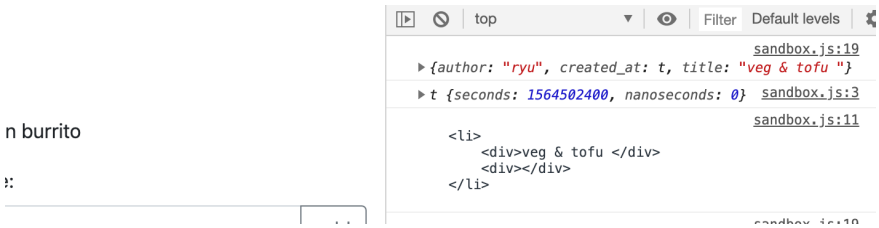
- adding html li elements into our ul innerHtml

```
const list = document.querySelector('ul');
const addRecipe = (recipe) => {
    let html = `
    <li>
    <div>${recipe.title}
    </li>
    `;
    console.log(html);
    list.innerHTML += html;
}

db.collection('recipes').get().then((snapshot) => {
    //when we have the data
    // console.log(snapshot.docs[0].data());
    snapshot.docs.forEach((doc) => {
        addRecipe(doc.data());
    })
}).catch((error) => {
    console.log(error);
})
```

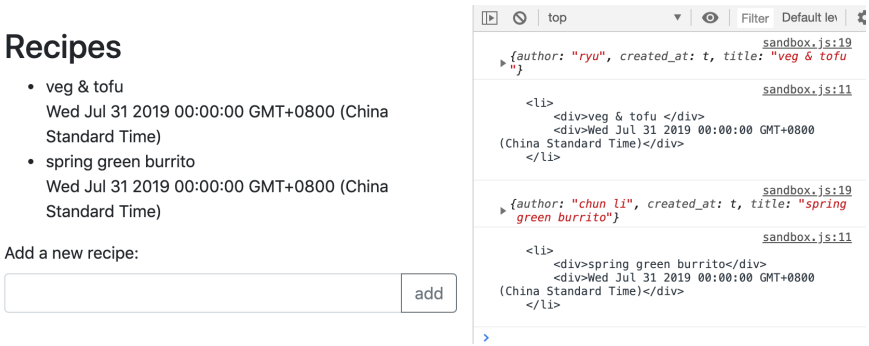


- now I also want to output the time that we created that property

```
const list = document.querySelector('ul');
const addRecipe = (recipe) => {
    console.log(recipe.created_at);   //print the created t
    let html = `
    <li>
    <div>${recipe.title}
    </li>
    `;
    console.log(html);
    list.innerHTML += html;
}
```



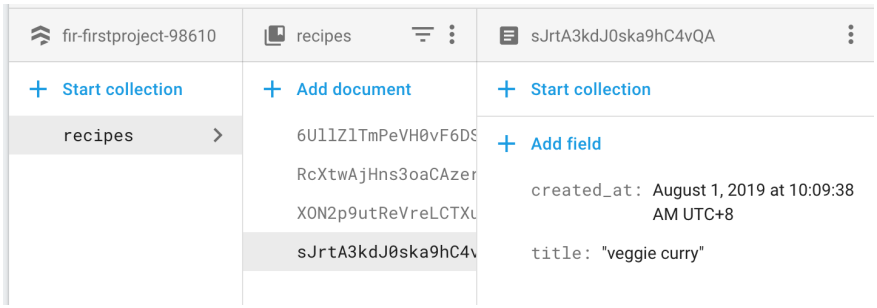
```
const list = document.querySelector('ul');
const addRecipe = (recipe) => {
    let time = recipe.created_at.toDate();
    let html = `
    <li>
        <div>${recipe.title}</div>
        <div>${time}</div>
    </li>
    `;
    console.log(html);
    list.innerHTML += html;
}

db.collection('recipes').get().then((snapshot) => {
    snapshot.docs.forEach((doc) => {
        console.log(doc.data());
        addRecipe(doc.data());
    })
}).catch((error) => {
    console.log(error);
})
```



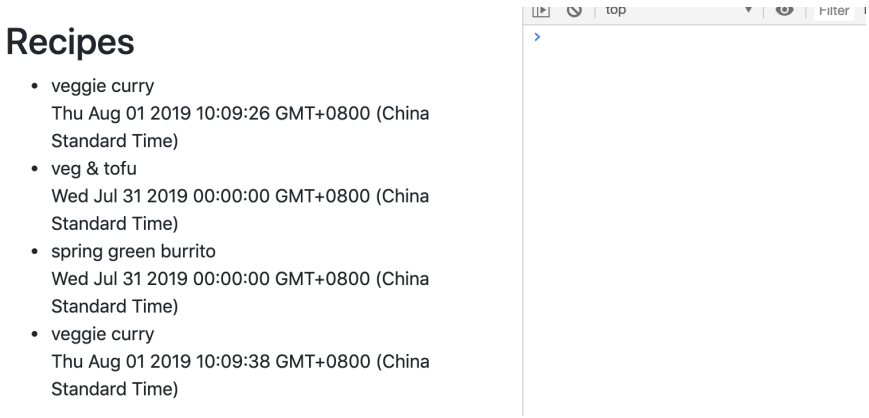
Saving Documents

- when we actually save a document to our firestore database, what we do is we actuall pass it a javascript object and it saves it as a document
- create an object that looks like that and call it recipe.
- adding a document

```
//saving documents
const list = document.querySelector('ul');
const form = document.querySelector('form');

const addRecipe = (recipe) => {
    // console.log(recipe.created_at);
    let time = recipe.created_at.toDate();
    let html = `
    <li>
        <div>${recipe.title}</div>
        <div>${time}</div>
    </li>
    `;
    list.innerHTML += html;
}

db.collection('recipes').get().then((snapshot) => {
    //when we have the data
    // console.log(snapshot.docs[0].data());
    snapshot.docs.forEach((doc) => {
        addRecipe(doc.data());
    })
}).catch((error) => {
    console.log(error);
})

// add documents
form.addEventListener('submit', e => {
    e.preventDefault();
    const now = new Date();
    const recipe = {
        title: form.recipe.value,
        created_at: firebase.firestore.Timestamp.fromDate(
    };
    db.collection('recipes').add(recipe).then(() => {
        console.log('recipe added')
    }).catch(error => {
        console.log(error);
    });
})
```



- so when we refersh the page, the new document has been added



Deleting Documents

- first, what I'd like to do is output some kind of little button in each one of these are like tags
- and when we click that button, it's going to delete that document from the database

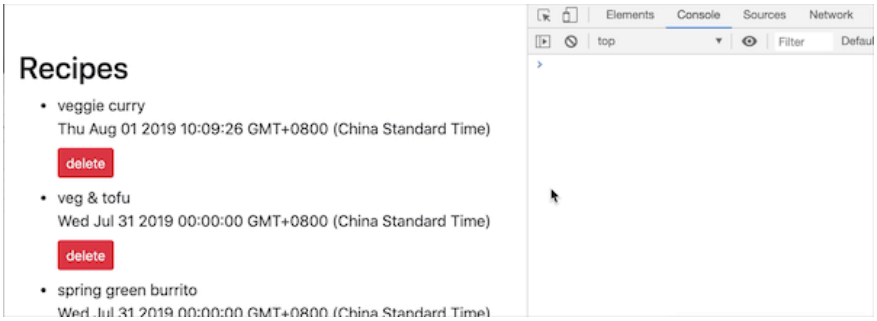
```
const addRecipe = (recipe) => {
    // console.log(recipe.created_at);
    let time = recipe.created_at.toDate();
    let html = `
    <li>
        <div>${recipe.title}</div>
        <div>${time}</div>
        <button class="btn btn-danger btn-sm my-2">delete</
    </li>
    `;
    list.innerHTML += html;
}
```

- we bind an `id` to every li element

```
const addRecipe = (recipe, id) => {
    // console.log(recipe.created_at);
    let time = recipe.created_at.toDate();
    let html = `
    <li data-id="${id}">
        <div>${recipe.title}</div>
        <div>${time}</div>
        <button class="btn btn-danger btn-sm my-2">delete</
    </li>
    `;
    list.innerHTML += html;
}
```

`get the id`

```
list.addEventListener('click', e => {
    if (e.target.tagName === 'BUTTON') {
        const id = e.target.parentElement.getAttribute('dat
        console.log(id);
    }
})
```



- now we can delete a li element

```
//deleting data
list.addEventListener('click', e => {
    if (e.target.tagName === 'BUTTON') {
        const id = e.target.parentElement.getAttribute('dat
        db.collection('recipes').doc(id).delete().then(() =
            console.log('recipe delete')
        });
    }
})
```

- but the problem is that you can find no element is deleted because we have to refersh

Real-time Listeners

- since when we delete the li element our UI is not automactically updated
- It would be nice if we could keep our UI or template in sync with our firestore database
- Now a firestore database has `real time` capabilities to do this
- we can set up a `real-time` listener in our javascript code and that listener will actively listen at all times to our database collection
- Then when something changes in the collection like a new document added or a document removed it will get the update back and we can update the UI with it.

```
//Real-time Listeners
//delete document
const list = document.querySelector('ul');
const form = document.querySelector('form');

const addRecipe = (recipe, id) => {
    // console.log(recipe.created_at);
    let time = recipe.created_at.toDate();
    let html = `
    <li data-id="${id}">
        <div>${recipe.title}</div>
        <div>${time}</div>
        <button class="btn btn-danger btn-sm my-2">delete</
    </li>
    `;
    list.innerHTML += html;
}

const deleteRecipe = (id) => {
    const recipes = document.querySelectorAll('li');
    recipes.forEach(recipe => {
        if (recipe.getAttribute('data-id') === id) {
            recipe.remove();
        }
    })
}

db.collection('recipes').get().then((snapshot) => {
    snapshot.docs.forEach((doc) => {
        addRecipe(doc.data(), doc.id);
    })
}).catch((error) => {
    console.log(error);
})

//get documents
db.collection('recipes').onSnapshot(snapshot => {
    // console.log(snapshot.docChanges());
    snapshot.docChanges().forEach(change => {
        const doc = change.doc;
        if (change.type === 'added') {
            addRecipe(doc.data(), doc.id);
        } else if (change.type === 'removed') {
            deleteRecipe(doc.id);
        }
    })
});//every time the collection changes in any way, firesto
//an the snapshot represents the state of that conllection
```

```
// add documents
form.addEventListener('submit', e => {
    e.preventDefault();
    const now = new Date();
    const recipe = {
        title: form.recipe.value,
        created_at: firebase.firestore.Timestamp.fromDate(n
    };
    db.collection('recipes').add(recipe).then(() => {
        console.log('recipe added')
    }).catch(error => {
        console.log(error);
    });
})

//deleting data
list.addEventListener('click', e => {
    if (e.target.tagName === 'BUTTON') {
        const id = e.target.parentElement.getAttribute('dat
        db.collection('recipes').doc(id).delete().then(() =
            console.log('recipe delete')
        });
    }
})
```

Unsubscribing

```
/Real-time Listeners
//Unsubscribing
const list = document.querySelector('ul');
const form = document.querySelector('form');
const button = document.querySelector('button');

const addRecipe = (recipe, id) => {
    // console.log(recipe.created_at);
    let time = recipe.created_at.toDate();
    let html = `
    <li data-id="${id}">
        <div>${recipe.title}</div>
        <div>${time}</div>
        <button class="btn btn-danger btn-sm my-2">delete</
    </li>
    `;
    list.innerHTML += html;
}

const deleteRecipe = (id) => {
    const recipes = document.querySelectorAll('li');
    recipes.forEach(recipe => {
        if (recipe.getAttribute('data-id') === id) {
            recipe.remove();
        }
    })
}

db.collection('recipes').get().then((snapshot) => {
    snapshot.docs.forEach((doc) => {
        addRecipe(doc.data(), doc.id);
    })
}).catch((error) => {
    console.log(error);
})

//get documents
const unsub = db.collection('recipes').onSnapshot(snapshot
    // console.log(snapshot.docChanges());
    snapshot.docChanges().forEach(change => {
        const doc = change.doc;
        if (change.type === 'added') {
            addRecipe(doc.data(), doc.id);
        } else if (change.type === 'removed') {
            deleteRecipe(doc.id);
        }
    })
});//every time the collection changes in any way, firesto
```

```
//an the snapshot represents the state of that conllection




// add documents
form.addEventListener('submit', e => {
    e.preventDefault();
    const now = new Date();
    const recipe = {
        title: form.recipe.value,
        created_at: firebase.firestore.Timestamp.fromDate(n
    };
    db.collection('recipes').add(recipe).then(() => {
        console.log('recipe added')
    }).catch(error => {
        console.log(error);
    });
})

//deleting data
list.addEventListener('click', e => {
    if (e.target.tagName === 'BUTTON') {
        const id = e.target.parentElement.getAttribute('dat
        db.collection('recipes').doc(id).delete().then(() =
            console.log('recipe delete')
        });
    }
})


//unsub from database changes
button.addEventListener('click', () => {
    unsub();
    console.log('unsubscribed from collection changes')
})
```

Ch 2：Babel Webpack

1. babel webpack

Installing Node.js & babel

```
→ node -v
//v12.5.0


npm init
```

```
Press ^C at any time to quit.
package name: (babel_webpack)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/Git/gitJsSyntax/babel_webpack/pacl

{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",

  "license": "ISC"
}

Is this OK? (yes) y
```

- install `babel/core balel/cli`

→ `npm install @babel/core @babel/cli --sav-dev`

- in order to convert `ES6` to `ES5`, we need to install one more thing.

→ `npm install @babel/preset-env --save-dev`

- so once we have htat installed we have to define the `preset` in a new file inside our project directory called `.babelrc`

`.babelrc`

```
{
    "presets":["@babel/preset-env"]
}
```

Using Babel

1. Install Node.js (and npm) on our computer
2. Use `npm init` to create a package.json file in the project directory
3. Use npm to install `babel/core & babel/cli` packages
4. Install the babel preset (env) & register it in `.babelrc`
5. now inside `package.json` and `.babelrc`

```
{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@babel/cli": "^7.5.5",
    "@babel/core": "^7.5.5"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.5.5"
  }
}
```

```
{
    "presets":["@babel/preset-env"]
}
```

- now we create a before.js file

```
const greet = (name) => {
  console.log(`hello ${name}`);
}
greet();
```

- so we're goint to use the `babelrc` and `cli` just stands for command line interface and it means we can run a babel down here in the terminal
- now the `babel cli` is inside the node modules folder inside the `bin` directory right here.

→ `node_modules/.bin/babel before.js -o after.js`

- inside after.js

```
"use strict";

var greet = function greet(name) {
  console.log("hello ".concat(name));
};
greet();
```

node_modules

- This is where all of the third party package code lives, so if we install a package it lives inside this `node modules` folder
- If I was to ever send my project to another developer or upload it to github then I wouldn't send or upload this node modules folder every other file or folder I would send or upload but not the node modules because it's so huge it contains a lot of code in it.
- So if I don't send or upload this folder(node_modules) with my project how will another developer run this project correctly without that folder.

- well, they can't They need to get that folder back and reinstall all of those things. But that's simple thanks to our `package.json` file down here
- Remember the `package.json` file keeps track of all the packages that we install for this project.
- So when a developer gets this project without the `node_modules` folder they can still see inside the `package.json` file what packages need to be installed right here.
- Now they don't have to install each one individually they can do them altogether by just using one command. And that is `npm install` and this reinstall is all the packages inside `package.json` and create a known modules folder for them to get.

- Remember: If you were to download the files from github repository then you won't see the `node_modules` folder, and you would have to run `npm install` to install all of these packages right there and that will create the `node_modules` folder for you.

NPM Script & Watching Files

- we create all required folders and files

- now we convert the older code to the new .js file

`→ node_modules/.bin/babel src/index.js -o dist/assets/bundle.js`

```
//index.js
const greet = name => {
    console.log(`hello ${name}`);
}

greet('mario');
greet('luigi');
greet('link');
```

```
//bundle.js
"use strict";

var greet = function greet(name) {
  console.log("hello ".concat(name));
};

greet('mario');
greet('luigi');
greet('link');
```

- but the problem is that there is a lot to write every time we want to do this conversion because we might make changes to this code several times and we have to write this out every time.
- so I'm goint to copy the command line instruction, and we're going to go over to `package.json`, and you'll notice we have this script section in package.

`package.json`

```
{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "babel": "node_modules/.bin/babel src/index.js -o dist/
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@babel/cli": "^7.5.5",
    "@babel/core": "^7.5.5"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.5.5"
  }
}
```

- so we altering the babel

`"babel": "node_modules/.bin/babel src/index.js -o dist/assets/bundle.js"`

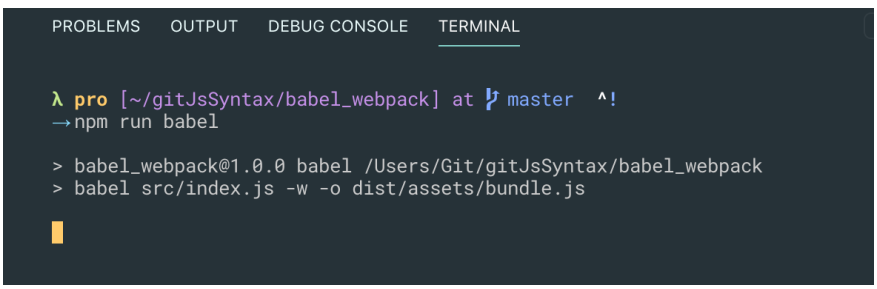
- now we can run `npm run babel`

```
→ npm run babel

> babel_webpack@1.0.0 babel /Users/Git/gitJsSyntax/babel_we
> babel src/index.js -o dist/assets/bundle.js
```

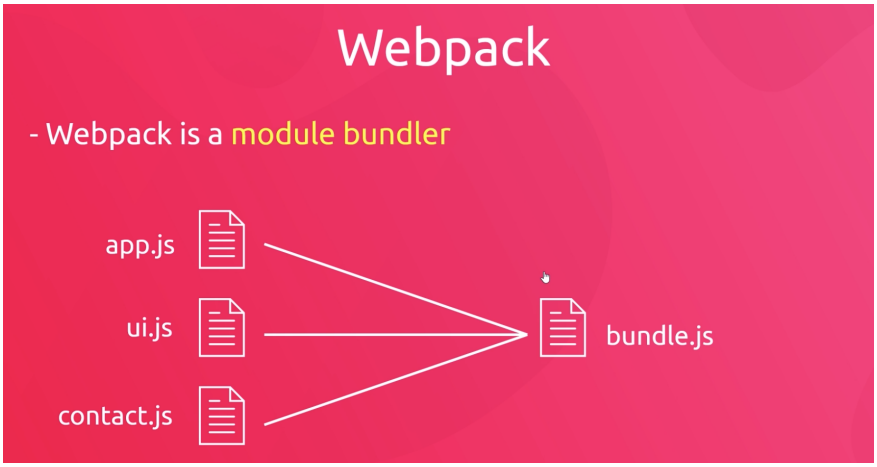
- now we have altered the `bundle.js` 's content
- `watch flag` , inside package.json:

`"babel": "node_modules/.bin/babel src/index.js -w -o dist/assets/bundle.js"`

- now after we alter .js file, we just need to save the .js file. It will automactially change the codes



Webpack



- Webpack is a `module bundler`
- Works well with `babel`
- Local development server

setting up a Webpack file

- create a `webpack.config.js`
- now this is not javascript that can run normally in a browser
- it's javascript that can only run directly on a computer with the help of node.js
- it's going to be running in the computer to compile all our code together bundle it all together and spit out this bundle and this bundle is ultimately the only thing that is going to be running inside a browser

`webpack.config.js`

```
module.exports = {};
```

- so module exports basically means that we're going to export a Webpack configuration object right here
- inside this object we need to specify two main properties:
    1. we need to set an `entry` property
    2. `output` property

`webpack.config.js`

```
const path = require('path');

module.exports = {
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist/assets'),
        filename: 'bundle.js'
    }
};
```

Webpack CLI

→ `npm install webpack webpack-cli --save-dev`

- we have installed webpack

run:

→ `node_modules/.bin/webpack`

- updating package.json:

```
{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",

  "main": "index.js",
  "scripts": {
    "babel": "node_modules/.bin/babel src/index.js -w -o di
    "webpack": "node_modules/.bin/webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@babel/cli": "^7.5.5",
    "@babel/core": "^7.5.5"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.5.5",
    "webpack": "^4.39.1",
    "webpack-cli": "^3.3.6"
  }
}
```

→ `npm run webpack`

```
→ npm run webpack

> babel_webpack@1.0.0 webpack /Users/Git/gitJsSyntax/babel_
> webpack
```

`bundle.js`

```
!function(e){var t={};function n(r){if(t[r])return t[r].exp
```

Introduction to Modules`

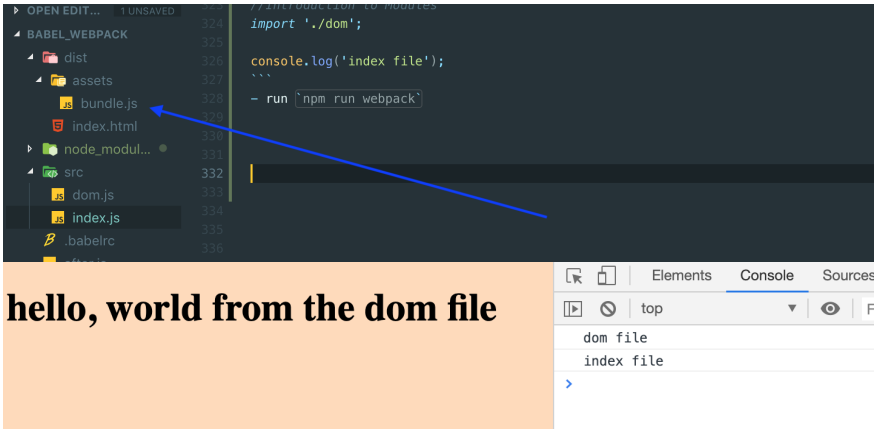
- we temporarily clear `index.js` 's content
- create a `dom.js`

```
console.log('dom file');

const body = document.querySelector('body');

const styleBody = () => {
    body.style.background = 'peachpuff';
};

const addTitle = (text) => {
    const title = document.createElement('h1');
    title.textContent = text;
    body.appendChild(title)

};
styleBody();
addTitle('hello, world from the dom file');
```

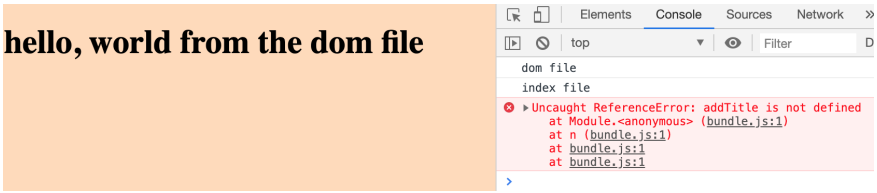
`index.js`

```
//Introduction to Modules
import './dom';

console.log('index file');
```

- run `npm run webpack`



- what if we call the `addTitle()` from `dom.js`
- run → `npm run webpack`



- we can see that there is an error: `addTitle` is not defined
- so when we use modules to `import` a file like this it runs the code inside that file but it doesn't automatically share any variables or functions with the file that import it.
- we couldn't automactically access any of the variables or function that we define inside the Dom file
- so in order for this to work we have to manually export functions that we want to use inside index from this file.
- well we can do that by putting an export keyword in front of the things we want to export.

`dom.js`

```
//Modules
console.log('dom file');

const body = document.querySelector('body');

export const styleBody = () => {
    body.style.background = 'peachpuff';
};

export const addTitle = (text) => {
    const title = document.createElement('h1');
    title.textContent = text;
    body.appendChild(title)

};
styleBody();
addTitle('hello, world from the dom file');
```

- now if I save it if I come over here we have to explicitly import those things from this file as well.
- so inside index.js

```
import { styleBody, addTitle } from './dom';

console.log('index file');
addTitle('test')
styleBody();
```

`updating dom.js`

```
//Modules
console.log('dom file');

const body = document.querySelector('body');

export const styleBody = () => {
    body.style.background = 'peachpuff';
};

export const addTitle = (text) => {
    const title = document.createElement('h1');
    title.textContent = text;
    body.appendChild(title)

};
```

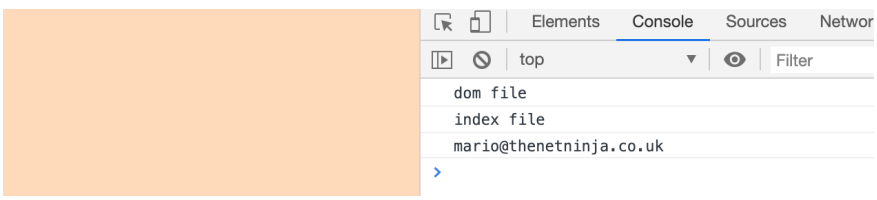
- so now if I `npm run webpack` again

- now we have added a titile "test"
- now we can export more than just functions from a file we can export any kind of data like arrays object strings classes
- let's now try exporting a string:

```
//Modules
console.log('dom file');

const body = document.querySelector('body');

export const styleBody = () => {
    body.style.background = 'peachpuff';
};

export const addTitle = (text) => {
    const title = document.createElement('h1');
    title.textContent = text;
    body.appendChild(title)
};
export const contact = 'mario@thenetninja.co.uk';
```

```
import { styleBody, addTitle, contact } from './dom';
console.log('index file');
addTitle('test')
styleBody();
console.log(contact);
```

→ `npm run webpack`



```
//Deafult exports
console.log('dom file');

const body = document.querySelector('body');

const styleBody = () => {
    body.style.background = 'peachpuff';
};

const addTitle = (text) => {
    const title = document.createElement('h1');
    title.textContent = text;
    body.appendChild(title)
};
const contact = 'mario@thenetninja.co.uk';

export { styleBody, addTitle, contact }
```

→ `npm run webpack`

- then, the result is the same

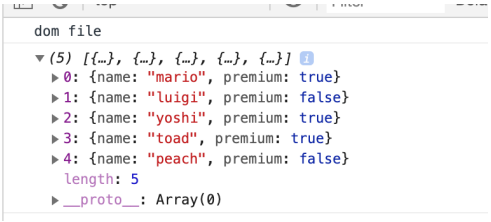
default exports

- create a `data.js` inside src folder

```
const users = [
    { name: 'mario', premium: true },
    { name: 'luigi', premium: false },
    { name: 'yoshi', premium: true },
    { name: 'toad', premium: true },
    { name: 'peach', premium: false }
];
```

- imagine now I want to export this thing from this file.
- Now previously when we've been exporting things we've been exporting multiple different things.
- we'd make something a default export

```
//Default Exports
import { styleBody, addTitle, contact } from './dom';
import users from './data';

console.log(users);
```



- updating data.js

```
const users = [
    { name: 'mario', premium: true },
    { name: 'luigi', premium: false },
    { name: 'yoshi', premium: true },
    { name: 'toad', premium: true },
    { name: 'peach', premium: false }
];

export const getPremUser = (users) => {
    return users.filter(users => users.premium);
}

export default users;
```

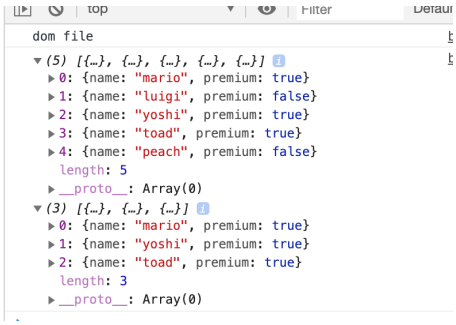
dom.js

```
//Deafult exports
console.log('dom file');

const body = document.querySelector('body');

const styleBody = () => {
    body.style.background = 'peachpuff';
};

const addTitle = (text) => {
    const title = document.createElement('h1');
    title.textContent = text;
    body.appendChild(title)
};
const contact = 'mario@thenetninja.co.uk';

export { styleBody, addTitle, contact }
```

index.js

```
//Default Exports
import { styleBody, addTitle, contact } from './dom';
import users, { getPremUser } from './data';

const premUsers = getPremUser(users);
console.log(users, premUsers);
```

→ `npm run webpack`



- Now let's have a look at the alternative way of exporting and things where we define it all at the bottom.

`altering data.js`

```
const users = [
    { name: 'mario', premium: true },
    { name: 'luigi', premium: false },
    { name: 'yoshi', premium: true },
    { name: 'toad', premium: true },
    { name: 'peach', premium: false }
];

const getPremUser = (users) => {
    return users.filter(users => users.premium);
}

export { getPremUser, users as default }
```

- the same output

Watching for Changes

- now we can use modules inside our project which is going to really help us to structure our code a bit better in the future.
- There's a small problem every time we make a change to a file inside our source folder at the minute we have to rerun the webpack command to rebound all the javascript.

- It would be nice if we could instead have webpack automatically rerun every time we make a change to a file and save it much like we did with babel before and we can do this.
- All we need to do is come to package.json and where we have our web script come to the end.

`alter package.json`

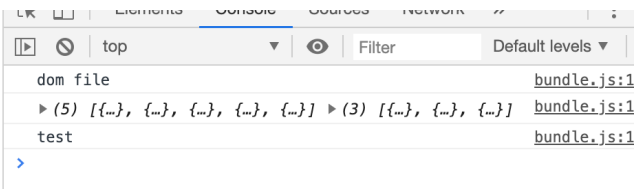
```
{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "babel": "node_modules/.bin/babel src/index.js -w -o di
    "webpack": "node_modules/.bin/webpack -w"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@babel/cli": "^7.5.5",
    "@babel/core": "^7.5.5"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.5.5",
    "webpack": "^4.39.1",
    "webpack-cli": "^3.3.6"
  }
}
```

- now we run

```
→ npm run webpack

> babel_webpack@1.0.0 webpack /Users/Git/gitJsSyntax/babel_
> webpack -w
```

- now we optionally adding a statement into index.js

```
console.log('test');
```



- if you want to cancel this

`control + c` on terminal

Webpack Dev Server

→ `npm install webpack-dev-server@3.2.1`

- next step, we need to do is configure that webpack dev server inside the webpack.config.js
- to do that, we need another property, which is called `dev server` like so and this is an object.

`webpack.config.js`

```
const path = require('path');

module.exports = {
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist/assets'),
        filename: 'bundle.js'
    },
    devServer: {
        contentBase: path.resolve(__dirname, 'dist'),
        publicPath: '/assets/'
    }
};
```

`package.json`

```
{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "babel": "node_modules/.bin/babel src/index.js -w -o di
    "webpack": "node_modules/.bin/webpack -w",
    "serve": "webpack-dev-server"
  },
```

- sometimes we need to update `webpack-dev-server`
- run this instruction:

→ `npm install webpack-dev-server`

- and it will update itself to the newest version and it should work just fine

now, we clear `bundle.js` 's content

- go to index.js, say test 3
- if we go back to `bundle.js` , we still see nothing. All it's done is created a virtual file which is erving up from this path.
- we can still see the same results and we can still see all of this logged to the console because it's serving that virtual file that virtual javascript file but we don't see it inside `bundle.js` , there is no physical manifestation of it.
- Now this is great for development because it speeds things up for us every time we make a save.

`Production & Development Modes`

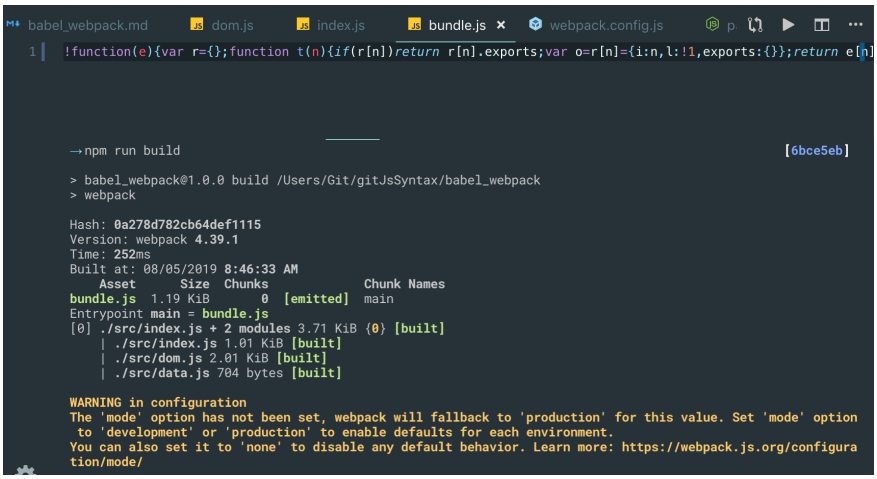
- take off the `-w` file because we're only typically going to build this once when we want to build
- we don't need to watch files anymore and rerun this when we're making changes because tthat's what the development script is for.

`altering package.json`

```
{
  "name": "babel_webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "node_modules/.bin/webpack",
    "serve": "webpack-dev-server"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@babel/cli": "^7.5.5",
    "@babel/core": "^7.5.5",
    "webpack-dev-server": "^3.7.2"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.5.5",
    "webpack": "^4.39.1",
    "webpack-cli": "^3.3.6"
  }
}
```

→ `npm run build`

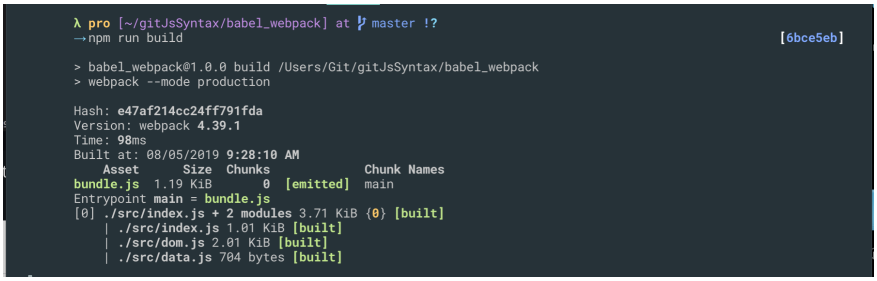
- now we can see the `bundle.js` 's content:



- the problem is that we see the yellow warning

`altering package.json`

```
  "scripts": {
    "build": "node_modules/.bin/webpack --mode production",
    "serve": "webpack-dev-server --mode development"
  },
```



- we don't get that big yellow warning anymore.

Babel & Webpack Together

→ `npm install babel-loader --save -dev`

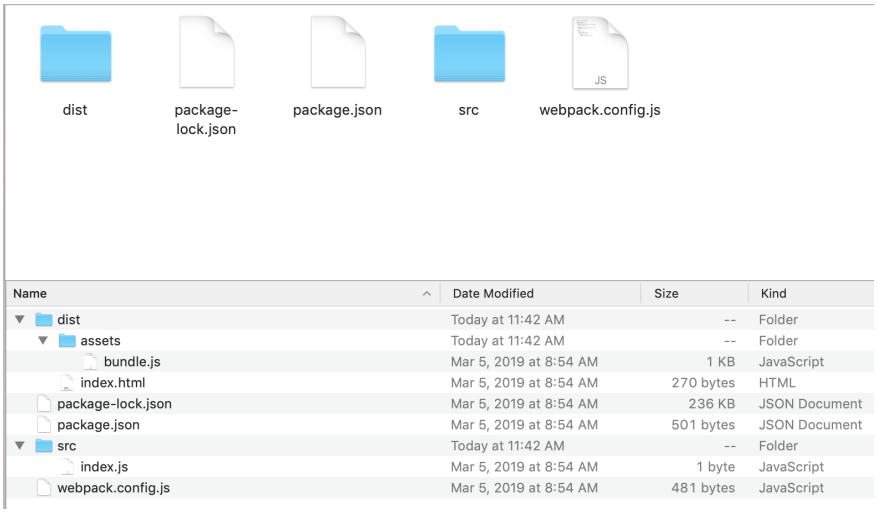
- we're going to create a module property
- regular expression:

```
rules: [{
    test: /\.js$/
}]
//this regualr expression is testing for any file that ends
// $ means the end
// dot(.) means any character by default
// " \.js$ " this whole expression means look for files wh



    module: {
        rules: [{
            test: /\.js$/,
            exclude: /node_modules/
        }]
    }
//imagine we import some kind of package from node modules
//so we don't want to do any additional processing on thei
//we don't want to run anything from there through the babe

//updating
    module: {
        rules: [{
            test: /\.js$/,
            exclude: /node_modules/,
            use: {
                loader: 'babel-loader',
                options: {
                    presets: ['@babel/preset-env']
                }
            }
        }]
    }
// now we're looking for any .js file that's not coming fr
```

Webpack Boilerplate

- Now you might be thinking that this is an awful lot of work to do every time you create a new project which uses javascript.
- First of all we don't always have to use web for every single one of our projects.
- secondly when we do use this workflow and we do use Webpack we're not going to need to rewrite our whole configuration and our whole project structure from scratch.

- Typically what I do is create a boilerplate(样板) web project
- Then I upload that to Github once then whenever I want to create a new project
- What I do is go to download that boilerplate and get up or running right away with that boilerplate.



- so now everything is setting up ready for us, however, we don't see a `node_modules` folder over here and we do need those package listed package.json
- Remember: we don't have to install each one of these manually
- we just need to type

```
npm install
```

- then the only we need to run

```
npm run serve
```

- finally, we can rewrite `index.js`
- we can test our project by overwrite index.js to check!

End