

ES6 in Depth

Table of Contents

Introduction	0
Installation	1
Block Scope	2
Constant	3
Template String	4
New String Methods	5
New Number Methods	6
Arrow Function	7
Default Parameters	8
Lexical This Scope	9
Object Enhancement	10
New Array Methods	11
Spread Operator	12
Destructuring	13
Class	14
Set	15
Map	16
WeakSet	17
WeakMap	18
Iterators	19
Generators	20
For Of	21
Promise	22
Import Export	23

JS History

1995 - JS created
1997- ECMAScript 1
2009- ECMAScript 5 (ES5)
2015- ECMAScript 6 (ES6)

ECMAScript

Filename extensions	.es
Initial release	June 1997; 18 years ago (1997-06)
Latest release	Edition 6 (June 17, 2015; 9 months ago)
Type of format	Scripting language
Website	ECMA-262, ECMA-290, ECMA-327, ECMA-357, ECMA-402

3 more rows

[ECMAScript - Wikipedia, the free encyclopedia](#)
<https://en.wikipedia.org/wiki/ECMAScript>

ES6

ECMAScript 6, also known as ECMAScript 2015, is the latest version of the ECMAScript standard <https://github.com/lukehoban/es6features>

- Angular 2 based on ES6

Browser Compatibility Table

<https://kangax.github.io/compat-table/es6/>

Playground

<http://www.es6fiddle.net/>

Get Started Boilerplate

Webpack Babel ES6 starter

<https://github.com/hesing/project-starters>

Browserify Babel ES6 starter

<https://github.com/hesing/es6-in-depth/tree/es6-starter>

Gulp Babel ES6 starter

<https://github.com/hesing/gulp-babel-es6-starter>

Using ES6 today (non supporting browsers)

<https://babeljs.io/>

Typecript vs Babel

Typecript	Babel
Language- Typed superset of js	Not a language
Also can be used as transpiler	Transpiler
Less spec complaint	Spec complaint

Prerequisites

- Node JS

Quick Tip

To see all global installed node packages...

```
npm ls -global --depth 0
```

Install Babel

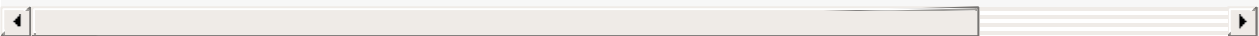
```
// global install
npm i babel-cli -g
babel --version

// local install
npm i babel-cli -D
node_modules/.bin/babel --version
npm i babel-preset-es2015 -D
```

Trnspilation form terminal

```
babel src --preset es2015 --out-dir build
babel src --preset es2015 -d build // same as above

// bundling
babel src --preset es2015 --out-file build/bundle.js
babel src --preset es2015 --out-file build/bundle.js -s // '-s' enable source maps
babel src --preset es2015 --out-file build/bundle.js -s -w // '-w' enable watch mode
```



Stage 0 (all in one babel package)

```
npm install babel-preset-stage-0
// .babelrc
"presets": ["stage-0"]
```

Move plugin configuration to .babelrc

```
// .babelrc
{
  "presets": ["es2015", ""stage-0""],
  "sourceMaps": true
}
```

Quick Tip

```
// npm shortcut commands
npm test
npm start
npm stop
npm restart

// run custom commands
npm run demo
```

npm run demo

```
// package.json
{
  "scripts": {
    "demo": "babel --version"
  }
}

npm run demo
```

Note

Fully qualified path not required, npm knows `node_modules/.bin/babel`

Babel with Npm Scripts

```
npm i babel-cli -D
npm i babel-preset-es2015 babel-preset-stage-0 -D
```

```
{
  "scripts": {
    "build:js": "babel public/src --out-dir public/build"
  }
}

// npm run build:js
```

Babel with gulp

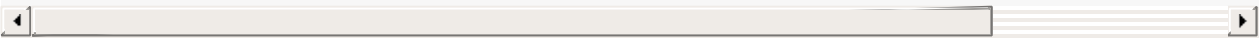
```
npm i gulp gulp-babel -D
```

```
//gulpfile.js
var gulp =require("gulp");
var babel =require("gulp-babel");

gulp.task("default", function(){
  return gulp.src("public/src/**/*.js")
    .pipe(babel())
    .dest("public/build");
});
```

Babel with webpack

```
npm i webpack -D
npm install babel-loader babel-core babel-preset-es2015 babel-pres
```




```
module.exports = {
  entry: './public/src/main.js',
  output: {
    path: 'public/webpackbuild',
    filename: 'bundle.js'
  },
  module: {
    loaders: [{
      test: /\.jsx?$/,
      exclude: /(node_modules|bower_components)/,
      loader: 'babel',
      query: {
        presets: ['es2015', 'stage-0']
      }
    }]
  }
}
```

Babel with browserify

```
npm i browserify -g
npm i browserify -D
npm i babelify -D
```

```
browserify --entry public/src/main.js --outfile public/build/browserify.js
```

```
// using with babel '-t' (transformer)
```

```
browserify --entry public/src/main.js --outfile public/build/browserify.js
```

Block Scope in ES6

```
// ES5
var x = 12;

if( x< 15){
  var msg = 'Quantity is small';
}

console.log(msg); // Quantity is small

// ES6
let x = 12;

if(x< 15){
  let msg = 'Quantity is small';
}

console.log(msg); // ReferenceError: msg is not defined
```

Example

ES5

```
var arr = [];  
  
for( var i =0; i < 10; i++){  
  arr.push(function(){  
    console.log(i);  
  });  
}  
  
arr.forEach(function(fn){  
  fn(); // (10) 10 // printed 10 times  
});
```

ES6 (using let)

```
var arr = [];  
  
for( let i =0; i < 10; i++){  
  arr.push(function(){  
    console.log(i);  
  });  
}  
  
arr.forEach(function(fn){  
  fn();  
});
```

// output

0
1
2
3
4
5
6
7
8
9

Constant Declaration

have block scope like let keyword

```
// ES5
var value = "hey";
value = "Cool";
console.log(value); // Cool

// ES6
const value = "hey";
value = "Cool";
console.log(value); // Error: "value" is read-only
```

Another Example

```
const obj = {};
obj.name = "Hemant";
console.log(obj); // {name: "Hemant"}

// but
const obj = {};
obj = {location: "HYD"};
console.log(obj); // Error: "obj" is read-only
```

Use Cases

```
const API_KEY = "XXX";
const API_SECRET = "DDGD_DDDGG-SS";
const port = 4000;
const PI = 3.14;
```

String Templates in ES6

```
var name="Hemant",  
    loc = "Hyderabad";  
  
var str = `  
  my name is ${name},  
  I am stying in ${loc}`;  
  
console.log(str);  
  
// output  
  my name is Hemant,  
  I am stying in Hyderabad
```

Use Case 1 (variable manipulation)

```
var x =1, y=2;  
  
var str = `sum of  
  ${x} + ${y} = ${x +y}  
  `;  
  
console.log(str);  
  
// output  
sum of  
  1 + 2 = 3
```

Use Case 2

```
var x =1, y=2;

var str = ` Today's date is
    ${new Date()}
`;

// output
Today's date is
    Sun Dec 06 2015 17:36:32 GMT+0530 (India Standard Time)
```

New Methods of the String Object

```
let str = "I am string";  
console.log(str.startsWith("I")); // true  
console.log(str.endsWith("ng")); // true  
console.log(str.repeat(2)); // I am stringI am string  
console.log(str.includes('rin')); // true
```


New static properties added to Number object

```
Number.EPSILON // 2.220446049250313e-16  
Number.MIN_SAFE_INTEGER // 9007199254740991  
Number.MAX_SAFE_INTEGER // 9007199254740991
```

New Methods on Number Object

```
Number.isFinite()  
Number.isInteger(3) // true  
Number.isSafeInteger(2.4) // false
```

Number updated methods

```
Number.isNaN(1); // false  
Number.parseInt()  
Number.parseFloat()
```

Arrow Function in ES6

```
// ES5
function greet(name){
  console.log("Hello! "+ name);
}
greet("Hemant");

// ES6
var greet = (name) => console.log("Hello! "+ name);
greet("Hemant");

// for single method parameter we can ommit () see below
// OR ( as returned value)
var greet = name => "Hello! "+ name;
console.log(greet("Hemant"));
```

Default Parameters in ES6

```
function greet(greeting, name="Anonymous"){  
  console.log(greeting + ", " + name);  
}
```

```
greet(); // undefined, Anonymous  
greet('Hello'); // Hello, Anonymous  
greet('Hello', 'Dady'); // Hello, Dady
```

```
// ES5  
function greet(cb){  
  cb();  
}  
  
greet(function(){  
  console.log("completed"); // completed  
})  
  
// ES6  
function greet(() => console.log("completed")){  
  cb();  
}  
greet();
```

Arrow function with this scope

```
// ES5
var obj = {
  name: "Hemant",
  actions: ["Dance", "Sing", "Swim", "Run"],
  displayAction: function(){
    var self = this;
    this.actions.forEach(function(action){
      console.log(self.name + " can " + action);
    });
  }
}

obj.displayAction();

// OR

var obj = {
  name: "Hemant",
  actions: ["Dance", "Sing", "Swim", "Run", "Eat"],
  displayAction: function(){
    this.actions.forEach(function(action){
      console.log(this.name + " can " + action);
    }).bind(this);
  }
}

obj.displayAction();
```

this with ES6 arrow function

```
var obj = {  
  name: "Hemant",  
  actions: ["Dance", "Sing", "Swim", "Run"],  
  displayAction: function(){  
    this.actions.forEach(action => console.log(this.name + " can " + action));  
  }  
}  
  
obj.displayAction();
```

Object Enhancement in ES6

```
var color = "red";
var speed = 10;
var drive= 'driveBMW';
function go(){}

var car = {color, speed, go, [drive]: function(){} }; // ES5 - {color: color, speed: speed, go: go, [drive]: function(){} }

console.log(car.color);
console.log(car.speed);
car.go();
car.driveBMW();
```

Shorthand properties

```
var firstname = "Hemant";
var lastname = "Singh";

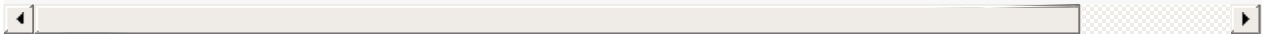
var fullname = {firstname, lastname};

var prefix = 'Mr.';

var person = {prefix, fullname};
console.log(person);
```

Copy properties to other object - Object.assign()

```
var obj1 = {  
  name: "hemant",  
  age: 30  
};  
  
var obj2 = {  
  loc: "hyd",  
  pin: 223  
};  
  
console.log(JSON.stringify(Object.assign({status: 'cool'}, obj1, obj2), null, 2));  
// {"status":"cool","name":"hemant","age":30,"loc":"hyd","pin":223}
```



...

New Array static methods

Static methods

```
Array.of  
Array.from
```

Array.of

```
var arr = Array(4); // arr.length = 4  
var arr1 = Array.of(4); // arr1.length = 1
```

Array.from

want to use Array methods on `arguments`

```
// ES5  
function greetMe(){  
    console.log(arguments.join('--')); // TypeError: arguments.join  
}  
  
console.log(greetMe('hemant', 'hello'));
```

```
// ES6  
function greetMe(){  
    console.log(Array.from(arguments).join('--')); // hemant--hello  
}  
  
console.log(greetMe('hemant', 'hello'));
```


Array instance methods

```
[].find()  
[].findIndex()  
[].copyWithin()
```

`[].find`

```
var arr = [  
  { name: "hemant", age: 30},  
  { name: "Vinay", age: 20},  
  { name: "Varun", age: 88}  
];  
  
var myName = arr.find(function(item){  
  return item.name === 'Varun';  
});  
  
console.log(myName); // { name: "Varun", age: 88}
```

`[].findIndex()`

```
var arr = [  
  { name: "hemant", age: 30},  
  { name: "Vinay", age: 20},  
  { name: "Varun", age: 88}  
];  
  
var indexOfName = arr.findIndex(function(item){  
  return item.name === 'Varun';  
});  
  
console.log(indexOfName); // 2
```

`Array.prototype.copyWithIn()`

```
var arr = [  
  { name: "hemant", age: 30},  
  { name: "Vinay", age: 20},  
  { name: "Varun", age: 88}  
];  
  
console.log(arr.copyWithIn(2, 0));  
// output  
[  
  { name: "hemant", age: 30},  
  { name: "Vinay", age: 20},  
  { name: "hemant", age: 30}  
]
```

Spread Operator

```
console.log([1,2,3]); // [1, 2, 3]
console.log(...[1,2,3]); // 1 2 3
```

Use Case 1

```
var arr1 = [1,2,3];
var arr2 = [4,5,6];

// arr1.push(arr2) or [1,2,3, arr2]
// console.log(arr1); // [1, 2, 3, [4, 5, 6]]

arr1.push(...arr2)
console.log(arr1); // [1, 2, 3, 4, 5, 6]
```

Use Case 2

```
function display(a,b,c){
  console.log(a+b+c); // 60
}

display(...[10,20,30]);
```

Destructuring

Extract values from array or object using literal syntax

```
var {color, position} = {  
  color: "blue",  
  name: "hemant",  
  state: "USA",  
  position: "UI Dev"  
};  
  
console.log(color, position); // blue UI Dev
```

Use Case 1

```
function generateObj(){  
  return {  
    color: "blue",  
    name: "hemant",  
    state: "USA",  
    position: "UI Dev"  
  };  
}  
  
var {name, state} = generateObj();  
console.log(name, state); // hemant USA
```

rename variable name

```
function generateObj(){
  return {
    color: "blue",
    name: "Paa",
    state: "India",
    position: "UI Dev"
  };
}

var {name:firstName, state: location} = generateObj();
console.log(firstName, location); // Paa India
```

Use Case 2 - ignoring some values

```
var [first,,,last] = [1,2,3,4,5];

console.log(first, last); // 1 5
```

Use Case 3 - get few item only

```
var friends = [
  {
    name: "Hemant",
    location: "Hyd",
    status: "Cool",
    email: "hemant@gmail.com"
  },
  {
    name: "Vinay",
    location: "USA",
    status: "N00",
    email: "vinay@gmail.com"
  }
];

friends.forEach(function({name, email}){
  console.log(name +", "+ email);
});

// output
Hemant, hemant@gmail.com
Vinay, vinay@gmail.com
```

Class

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

  showDetail(){
    console.log(`${this.name} have ${this.wheels} wheels`);
  }
}

var c = new Car("BMW", 4);
c.showDetail(); // BMW have 4 wheels
```

Class Inheritance

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

  showDetail(){
    console.log(`${this.name} have ${this.wheels} wheels`);
  }
}

var c = new Car("BMW", 4);
c.showDetail(); // BMW have 4 wheels

class Maruti extends Car{

  constructor(){
    super("Maruti", 4);
    this.cost = "20k";
    super.showDetail();
  }

  showDetail(){
    console.log(`${this.name} have ${this.wheels} wheels and it
  }
}

var m = new Maruti();
m.showDetail(); // Maruti have 4 wheels and it's cost is 20k
```


Set

A data structure of storing unique values, it's not an array

```
var s = new Set([1,2,2,2,2,3]);

s // [1,2,3]
s.add(4); // [1,2,3,4] , it's chainable - s.add(4).add(3).add(5)
s.delete(2); // [1,3,4]
s.clear(); // return undefined ( clears set )
s.has(3); // true
s.forEach(function(item){
  console.log(item); // 1,3,4
})

-----

var s = new Set([1,2,2,2,2,3]);

console.log(s.keys());
console.log(s.values());
console.log(s.entries());

// output
main.js:3 SetIterator {1, 2, 3}
main.js:4 SetIterator {1, 2, 3}
main.js:5 SetIterator {[1, 1], [2, 2], [3, 3]}
```

Map

- Similar to sets but store key/value pairs
- Set have add, Map have set/get methods

```
var m = new Map([['one', 'two'], [1,2]]);

console.log(m); // Map {"one" => "two", 1 => 2}
m.set('three', [1,2,3]);
console.log(m); // Map {"one" => "two", 1 => 2, "three" => [1, 2, 3]}
m.set('four', 4).set('five', 5);
console.log(m); // Map {"one" => "two", 1 => 2, "three" => [1, 2, 3], "four" => 4, "five" => 5}
console.log(m.get('three')); // [1, 2, 3]
console.log(m.has('one')); true

-----

console.log(m.keys());
console.log(m.values());
console.log(m.entries());

// output
MapIterator {"one", 1, "three", "four", "five"}
main.js:12 MapIterator {"two", 2, [1, 2, 3], 4, 5}
main.js:13 MapIterator [{"one", "two"}, [1, 2], ["three", Array[3]]]
```

Weak Set

- References to keys/values held weakly
- Do not prevent garbage collection
- Small api (add, delete, check for values) compared to set/map
- can't be iterated

Note

- keys must be objects
- the values can be arbitrary values.

```
var ws = new WeakSet();  
ws.add()  
ws.delete()  
ws.has()
```

WeakSet Demo

```
var ws = new WeakSet();  
var age = {age: 20};  
ws.add({name: 'hemant'}).add(age);  
console.log(ws.delete(age)); // true  
console.log(ws.has(age)); // false  
console.log(ws); // WeakSet {Object {name: "hemant"}}
```

Weak Map

- References to keys/values held weakly
- Do not prevent garbage collection
- Small api (add, delete, check for values) compared to set/map
- can't be iterated

Note

- keys must be objects
- the values can be arbitrary values.

```
var wm = new WeakMap();  
wm.add()  
wm.delete()  
wm.has()
```

WeakMap Demo

```
var wm = new WeakMap();  
wm.set({name: 'hemant'}, 30).set({age: 20}, function(){ return 'you'; })  
console.log(wm); // WeakMap {Object {name: "hemant"} => 30, Object {age: 20} => function(){ return 'you'; }}
```

Iterators

- Map/Set have methods (entries(), keys(), values()) that return iterator
- Provides sequential access to items
- Keep track of which item been accessed
- Iterators are objects with a next() method
- next() returns object with property value & done
- value - current value
- done - boolean indicates whether there is more item

Set Iterator Demo

```
new Set([1,2,3]).entries().next();  
// Object {value: Array[2], done: false}  
new Set([1,2,3]).entries().next().value // [1, 1]  
new Set([1,2,3]).entries().next().done // false
```

Map Iterator Demo

```
var mapEntries = new Map([['one', 1], ['two', 2]]).entries();  
  
mapEntries // MapIterator {"one", 1}, {"two", 2}  
mapEntries.next() // Object {value: Array[2], done: false}  
mapEntries.next() // Object {value: Array[2], done: false}  
mapEntries.next() // Object {value: undefined, done: true}
```

Generators

- Generators are the function that can be paused and resumed
- Defined using * after function keyword
- yield to pause a generator optionally pass back some value
- are used with iterators
- return an iterator when invoked
- next() method resume generator
- can pass value to iterator using next('my value')

Note

we have to install "babel-polyfill" to use this feature with webpack

```
// in webpack.config.js add below entry  
// entry: ['babel-polyfill', './main.js'],
```

Generator Demo

```
function* generateMe(){
  yield "One";
  yield "Two";
  yield [1, 2, 3];
}

var generator = generateMe();
console.log(generator.next());
console.log(generator.next());
console.log(generator.next());
console.log(generator.next());

// Output
// { value="One",  done=false}
// { value="Two",  done=false}
// { value=[3],    done=false}
// { done=true,    value=undefined}
```

Demo - Now i know my abc

```
function* abcGenerator(arr){
  for(var i=0; i<arr.length; i++){
    yield arr[i];
  }
}

var gen = abcGenerator(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']);

var abc = setInterval(function(){
  var letter = gen.next();

  if(letter.done){
    clearInterval(abc);
    console.log("Now I know my abc");
  } else {
    console.log(letter.value);
  }
}, 500);
```


for-of loop

- Used to iterate iterable objects
- Loop values instead of keys

Loop array

```
var arr = [1, 2];

for(let val of arr){
  console.log(val);
}
// output
1
2
```

Object literal not iterable

```
var arr = {name: 'hemant', age: 20, loc: 'Hyderabad'};

for(let val of arr){
  console.log(val);
}
// TypeError: arr is not iterable
```

Iterate Set

```
var s = new Set();
s.add("one");
s.add(2);
s.add("three");

for(let val of s){
  console.log(val);
}
// output
one
2
three
```

Iterate Map

```
var m = new Map();
m.set("one", 1);
m.set('two', 2);

for(let val of m){
  console.log(val);
}

// output
["one", 1]
["two", 2]
```

Promise

- Represents an action that hasn't yet completed
- solution to callback hell

```
var myPromise = new Promise(function(resolve, reject){
  var itemRequested = 10;

  if(itemRequested === 10 ){
    reject("Happy to offer you 100 Rs.");
  }

  if(itemRequested === 'star'){
    reject("It's false hope");
  }
});

myPromise.then(function(res){
  console.log(res)
}, function(res){
  console.log(res)
})
```

Promise Chaining

```
myPromise.then(function(res){
  return res;
}, function(err){
  return err;
})
.then(function(res){
  console.log(res + "Cool buddy"); // Happy to offer you 100 Rs.
});
```

catch

The catch callback is executed when the promise is rejected

```
new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { reject('Done!'); }, 3000);
})
.then(function(e) { console.log('done', e); })
.catch(function(e) { console.log('catch: ', e); });

// From the console:
// 'catch: Done!'
```

Promise.all

```
Promise.all([promise1, promise2]).then(function(results) {
  // Both promises resolved
})
.catch(function(error) {
  // One or more promises was rejected
});
```

Promise.all demo

```
Promise.all([fetch('/users.json'), navigator.getBattery()]).then(function() {
  // Both promises done!
});
```

If any promise is rejected the catch fires for the first rejection:

```
var promise1 = new Promise(function(resolve, reject) {
  setTimeout(function() { resolve('First!'); }, 4000);
});

var promise2 = new Promise(function(resolve, reject) {
  setTimeout(function() { reject('Second!'); }, 3000);
});

Promise.all([promise1, promise2]).then(function(results) {
  console.log('Then: ', one);
}).catch(function(err) {
  console.log('Catch: ', err);
});

// output
// Catch: Second!
```

Promise.race

Triggers as soon as any promise is resolved

```
var promise1 = new Promise(function(resolve, reject) {
  setTimeout(function() { resolve('First!'); }, 4000);
});

var promise2 = new Promise(function(resolve, reject) {
  setTimeout(function() { resolve('Second!'); }, 3000);
});

Promise.all([promise1, promise2]).then(function(results) {
  console.log('Then: ', one);
}).catch(function(err) {
  console.log('Catch: ', err);
});

// output
// Then: Second!
```

Ex - Native xhr request

```
var ajax = function(url){
  return new Promise(function(resolve, reject){
    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function(){
      if(xhr.readyState === 4 ){
        if(xhr.status === 200){
          resolve(xhr.responseText);
        } else {
          reject("something gone wrong");
        }
      }
    };

    xhr.open('get', url, true);
    xhr.send();
  });
};

ajax('https://api.github.com/users/hesing').then(function(res){
  console.log(res)
}, function(err){
  console.log(err)
});
```

fetch api return promise

```
fetch('/some/url')
  .then(function(response) {
    return //...
  }).then(function(returnedValue) {
    // ...
  }).catch(function(err) {
    // Error
  });
```

- [Battery API](#) return promise
- [fetch API](#) return promise

Import/Export module with ES6

```
// addition.js
function addTwo(a,b){
  return a + b;
}
export { addTwo };

// app.js
import { addTwo } from './addition';
console.log(addTwo(2, 4)); // 6
```

import/export multiple

```
// addition.js
function addTwo(a,b){
  return a + b;
}

function addThree(a,b,c){
  return a + b + c;
}

export { addTwo, addThree };

// app.js
import { addTwo, addThree } from './addition';

console.log(addTwo(2, 4)); // 6
console.log(addThree(2, 4, 5)); // 11
```

export other variation


```
// addition.js
export function addTwo(a,b){
  return a + b;
}

export function addThree(a,b,c){
  return a + b + c;
}
```

Rename exported function - addTwo as addTwoNumber

```
// addition.js
function addTwo(a,b){
  return a + b;
}

function addThree(a,b,c){
  return a + b + c;
}

export { addTwo, addThree };

// app.js
import { addTwo as addTwoNumber, addThree } from './addition';
console.log(addTwoNumber(6, 4)); // 10
console.log(addThree(2, 4, 5)); // 11
```

import all in one shot

```
// app.js
import * as adder from './addition';
console.log(adder.addTwoNumber(6, 4)); // 10
console.log(adder.addThree(2, 4, 5)); // 11
```

Use case (with lodash)

```
// users.js
export var users = [
  {name: "hemant", age: 20, location: "hyderabad"},
  {name: "vinay", age: 80, location: "hyderabad"},
  {name: "paa", age: 90, location: "hyderabad"},
  {name: "varun", age: 44, location: "hyderabad"}
];

// app.js
import * as _ from "lodash"; // npm i lodash -S
import {users } from './users';

console.log(_.where(users, {age: 20}));
```