7/12/2017

Variables and datatypes

```
Var name = 'John';

Console.log (name);

Var lastName = 'smith';

Var age = 26;

Console.log (age);

Vare fullAge = true;

Console.log (fullAge);
```

- Number : Floating Point numbers for decimals and integers
- String : Sequence of characters used for text
- Boolean : Logical data type that can only be true or false.
- undefined : Data type of a variable which does not have a value yet.
- NULL : Also means 'non-existent'.

Variable mutation and Type Corcting

```
// single line comment
/* multiline
   comment
*/
```

} it is useful to describe the content or stop the code.

Concatination

Type coversion means JavaScript Convert datatypes on the go.

```
Var name = 'John'
Var age = 26;

Console.log (name + age); // John26
Console.log (age + age); // 52

Var job, isMarried;

Console.log (job); // undefined

job = 'teacher';
isMarried = false;

Console.log (name + ' is a ' + age + ' years old ' + job + '. Is married? ' + isMarried +
'.');

age = 'thirty six';
job = driver;
Var lastName = prompt ('What is your lastname?');
console.log (lastname);
alert (name + ' is a ' + age + ' years old ' + job + '. Is married ' + isMarried
+ '.');
```

Operators

```
Var birthYear = 2016-26;

birthYear = 2016-26*2;
console.log(birthYear);

var ageJohn = 30;
var ageMark = 30;

ageJohn = (3+5) * 4-6;
ageJohn = ageMarkk = (3+5)*4-6;
Console.log(ageJohn); //26
console.log(ageMark); //26

ageJohn++;

Console.log(ageJohn);

ageMark *=2 //52 (ageMark = ageMark*2);
```

if/else statement

which allow us to code based on the descitions

```
Var name = 'John';
Var age = 26;
Var isMarried = 'No';

if (isMarried === 'yes') {
    Console.log(name + ' is married!');
} else {
    Console.log(name + ' will hopefully marry soon :)');
}

isMarried = false

if (isMarried) {
    console.log ('yes');
} else {
    Console.log('No');
}
```

```
if (23 == "23") {
    console.log('something to Print....'); // Print
}

if (23 === "23")
    console.log('something to print....'); // won't Print
}
```

Boolean logic & switch statement

```
var age = 16;

if (age < 20) {
    console.log('John is a teenager'); // prited
} else {
    console.log('John isa man:)
}
```

- AND (&&) => true && true
- OR (||) => true if one is true
- NOT (!) => inverts true / false value

```
Var age 16;

age >= 20;        // => false
age < 30;         // => true
!(age < 30);      // => false

age >= 20 && age < 30;  // => false

age >= 20 || age < 30;  // => true


if (age < 20) {
    console.log('John is a teenager');
} else if (age > 20 && age < 30) {
    console.log('John is a youngman');
} else {
    console.log('John is a man');
}
```

switch statement //

```
Var job = 'teacher';
    job = Prompt("what does john do?");
switch(job) {
    case 'teacher':
        console.log('John teaches kids.');
        break;
    case 'driver':
        console.log('John drives a cab in Lisbon.');
        break;
    case 'cop':
        console.log('John helps fight crime.');
        break;
    default:
        Console.log('John does something else.');
}
```

Coding Challenge

John and a friend invented a simple game where the player with the highest value of his height (in centimeters) plus five times his age wins (what a silly game);

1. Create variable for the height and ages of two friends and assign them some values

2. Calculate their scores

3. Decribe who wins and print the winner to the console. Include the score in the string that you output to the console. Don't forget that there can be draw (both players with the same score).

4. Extra: Add a 3rd player and now describe who wins.
Hint: you will need the && operator to take the decision.
if you can't solve this one, 😉

solution for code challenge

```
Var heightJohn = 176;
Var heighMark = 168;
Var heighJane = 156;
Var ageJohn = 27;
Var ageMark = 29;
Var ageJane = 30;
Var scoreJohn = heightJohn + ageJohn*5;

Var scoreMark = heightMark + ageMark*5;
Var scoreJane = heightJane + ageJane*5;

if (scoreJohn > scoreMark && scoreJohn > scoreJane) {
        console.log ('John won the game with ' + scoreJohn + 'points');
} else if (scoreMark > scoreJohn && scoreMark > scoreJane)
        console.log ('Mark won the game with ' + scoreMark + 'points.');
} else if (scoreMark == scoreJohn || scoreJohn == scoreJane || scoreJane == scoreMark) {
        console.log ('It's a draw Try it again');
}

else if (scoreJane > scoreJohn && scoreJane > scoremark ) {
        console.log ("Jane won the game with " + scoreJane + 'points');
}
```

DRY - Don't repeat yourself

TATA CONSULTANCY SERVICES
Experience certainty.

TATA

Functions

```
function calculateAge (name, yearOfBirth) {
    var age = 2017 - yearOfBirth;
    return age;
}

var ageJohn = CalculateAge ('John', 1990);

function calculateAge (name, yearOfBirth) {
    var age = 2017 - yearOfBirth;
    Console.log (name + 'is' + age + 'yearsold');
}

CalculateAge ('John', 1990);
Calculate Age ('Mark', 1986);
Calculate Age ('Jane', 1948);
```

```
function calculateAge(yearOfBirth) {
    var age = 2016-yearofBirth;
    return age;
}
var age_john = calculateAge(1990);
Var age_mike = calculateAge(1969);
Var age_mary = calculateAge(1948);

function yearsUntilRetirement(name, year) {
    var age = calculateAge(year);
    Var retirement = 65-age;
 if (retirement >=0) {
    Console.log(name+' retiresin '+retirement+'years.');
} else { console.log(name+' is ~~already retired.~~');
}
}
YearsuntilRetirement('john', 1990);
YearsuntilRetirement('mike', 1969);
YearsuntilRetirement('mary', 1948);
```

080-25533995
41141005
8970024365
205784289
326646-

Statement & Expressions

```
// function statement
function someFun (par) {
  //code
}
// function expressions
var somfun = function (par) {
  //code
}
```

```
if (x == 5) {
  //do something
}
```

```
3+4
var x = 3;
```

Arrays

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

console.log (names[0]);

names[1] = 'Ben';

console.log (names);
```

```
var John = ['John', 'Smith', 1999, 'Teacher', false];

John.push('blue');

console.log(John);

John.unshift('Mr.'); John.pop(); John.shift();
console.log(John);

var a = John.indexOf('Smith');
console.log(a);

if(John.indexOf('Teacher') === -1) {

    console.log('John is not a Teacher')
}
```

Objects and Properties

```
Var arr = [1, 2, 3]

arr[0];
```

this is for array

For object we don't have perticular order

object leteral

```
Var John = {
    name : 'John',
    lastName : 'Smith',
    YearOfBirth : 1990,
    job : 'teacher'
    isMaried : false
}

console.log(John);
```

object leteral

① `console.log(John.lastName);`

② `console.log(John['lastName']);`

```
Var xyz = 'job',

Console.log(John[xyz]);
```

Manuplating mutation

```
John.lastName = 'miller';
John['job'] = 'Programmer';

Console.log(John);
```

```
var jane = new Object();
jane.name = 'Jane';
jane.lastName = 'Smith';
jane['yearOfBirth'] = 1969;
jane['job'] = 'retired';
jane['isMarried'] = 'yes';

console.log(jane);
```

Object & Methods

```
var john = {
    name: 'John',
    lastName: 'Smith',
    yearOfBirth: 1990,
    job: 'teacher',
    isMarried: false,
    family: ['Jane', 'Mark', 'Bob'],

    calculateAge: function(yearOfBirth) {
        return 2016 - yearOfBirth;
    }
};

console.log(john.family[2]);
console.log(john.calculateAge(1990));
```

v2.

```
var john = {
    name : 'John',
    lastName : 'Smith',
    yearOfBirth : 1990,
    job : 'teacher',
    isMarried : false,
    family : ['Jane', 'Mark', 'Bob'],
    calculateAge : function() {
        this.age = 2016 - this.yearOfBirth;
    }
};

john.calculateAge();
console.log(john);
```

Var name = ['John', 'Jane', 'marry', 'mark', 'Bob'];

```
for (var i=0; i<name.length; i++) {
    console.log(name[i]);
}
for (var i=name.length-1; i>=0; i--)
{
    console.log(name[i]);
}
```

while loop

```
var i=0;
while (i<name.length) {
    console.log(name[i])
    i++;
}
```

Loops & Iterations

```
for (var i=0; i<5; i++) {
    console.log(i);
}
```

```
for (var i = 0; i <= 5; i++) {
    console.log(i);
    if (i === 3) {
        break;
    }
}
```

```
for (var i = 0; i <= 5; i++) {
    if (i === 3) {
        continue;
    }
    console.log(i);
}
```

Coding Callange -2

1. Create an array with some years where person were born
2. Create an empty array (just [])
3. Use a loop to fill the array with the ages of the persons
4. use another loop to log into the console whether each person is of full age (is an older), as well as their age
5. Finally, Create a function called PrintFullAge which receives the vector of years as an argument, excutes the step 2., 3. and 4. and returns a vector of true/false boolean values: true if the person is of full age (>=18 years) and false if not (<18 years)
6. Call the function with two different vectors and store the results in two variables: full_1 and full_2

Solution for code challenge

```
var years = [1990, 2001, 1995, 1994, 2014];
var ages = [ ];
var fullAges = [ ];
for (i=0; i < years.length ; i++) {

    ages[i] = 2016 - years[i];

}

for (i=0; i< ages.length; i++) {
    if (ages[i] >=18) {
        console.log('person is' + (i+1) + ages[i] + 'years old, and is of full age.');
        fullages.push(true)
    } else {
        console.log('person is' + ages[i] + 'years old, and is Not of full age.'); fullAges.push(false);
    }
}

function printFullAge(years) {

    return fullAges;

}
var full_1 = printFullAges(years);
var full_2 = printFullAge([2012, 1915, 1999]);
```

Object and function

Everything is an Object: Inheritance and the Prototype Chain

Primitives

- numbers
- strings
- Booleans
- undefined
- NULL

Everything else

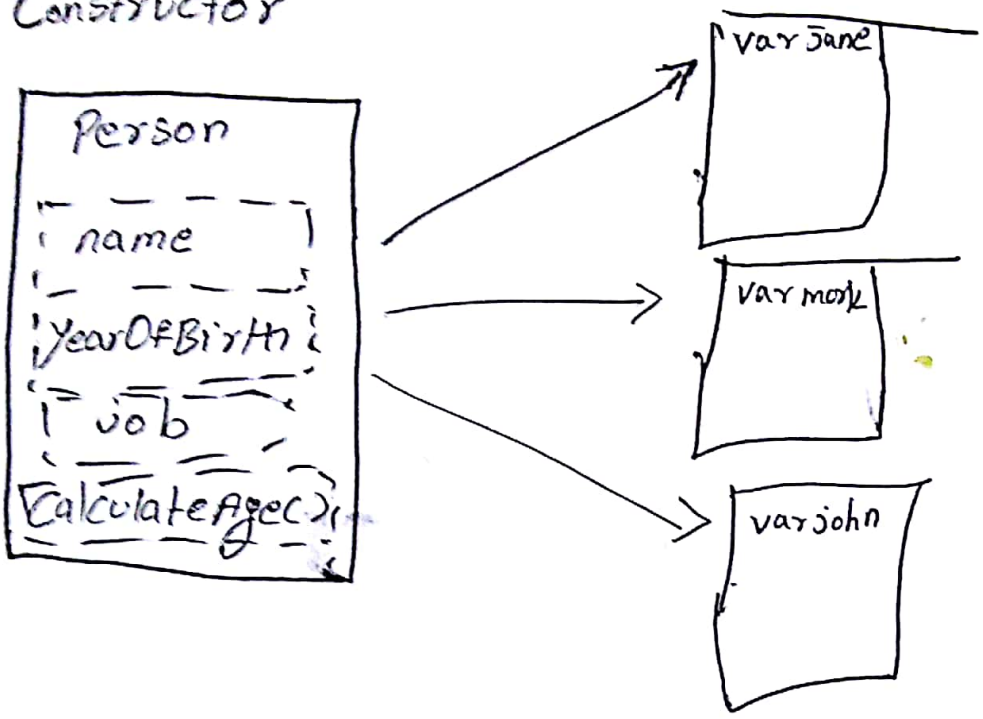
- Array
- functions
- Objects
- Dates
- Wrappers for numbers, Strings, Booleans

.. is an Objects

object oriented Programming

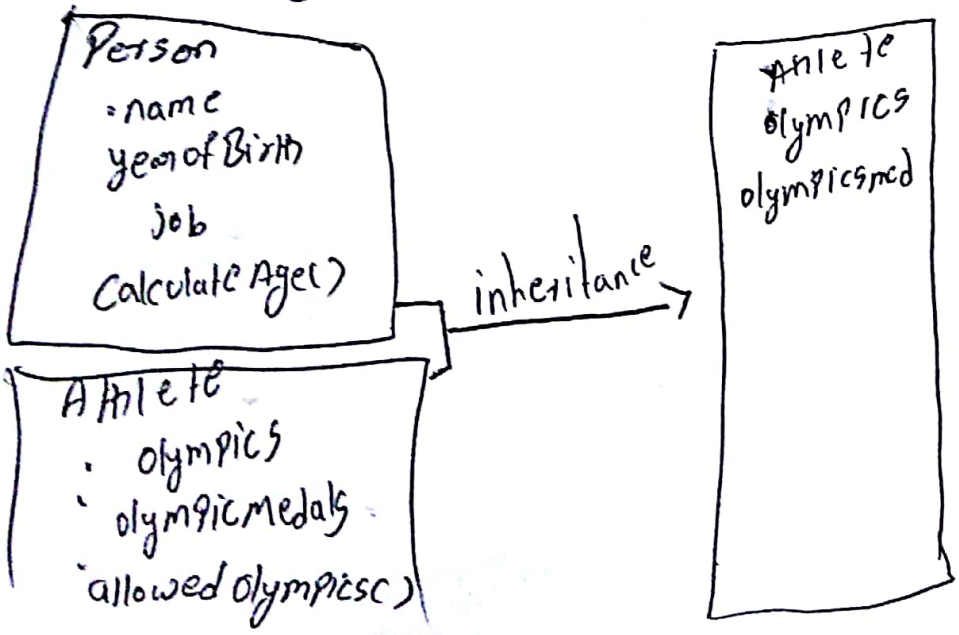
- Objects interacting with one another through metha and Properties;
- Used to store data, structure applications into modu and keeping code clean.

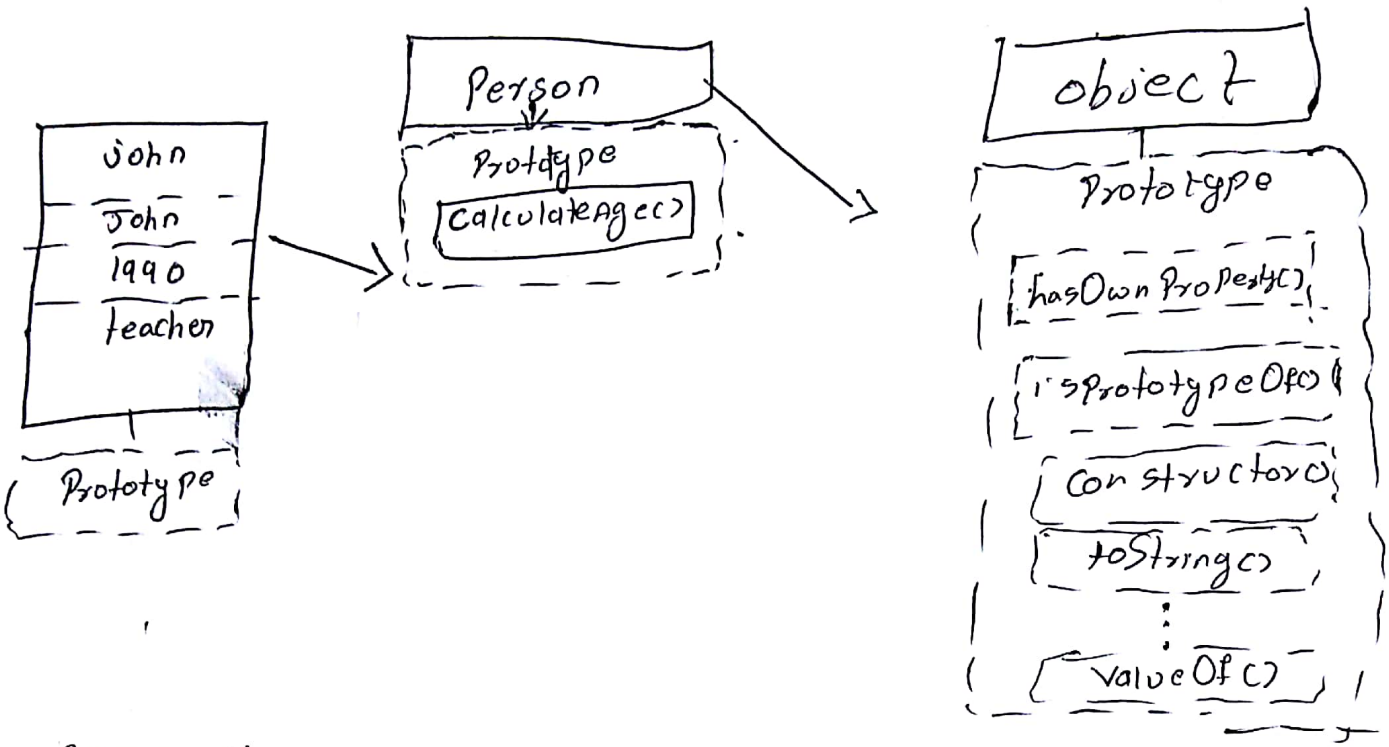
Constructor



inheritance.

It is nothing but one object based on the second one.





Summary

- Every JavaScript object has a prototype property, which inheritance possible in JavaScript;
- The prototype property of an object is where we put methods and properties that we want other object to inherit;
- The Constructor's prototype is not the prototype of the Constructor itself it's the prototype of All instance 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

Creating object using function Constructor

```
Var John {
    name: 'John',
    yearOfBirth: 1990;
    job = 'teacher'
};
```

object literal

```
Var person = function (name, yearOb, job)
    {
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job
    };

Var john = New person (John, 1990, teacher)

this.CalculateAge = function () {
        console.log (2017 - this.yearOfBirth);
    }

john.CalculateAge();

Person.Prototype.lastName = 'Smith';
```

The Prototype Chain in the console

__proto__ : Object

Creating objects: objects.create

```
Var personProto = {
    calculateAge: function() {
        console.log(2017 - 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}
});
```

Primites vs objects

```
Var a = 23;
var b = a;

a = 46;

Console.log (a); //46
Console.log (b); //23
```

```
var obj1 = {
    name : 'John',
    age = 26
};

Var obj2 = obj2

    obj1.age = 40

Console.log (obj1.age); //40
console.log (obj2.age); //40
```

* When storing an object ina variable, doesn't mean storing value, It means we ponting it to that perticular function.

```
Var age = 27;
Var obj = {
      name: 'Jonas',
      city : 'Lisbon'
      };
function Change (a, b) {
        a = 30
        b.city = 'sanFransicco';
    }
Change (age, obj);
```

```
console.log (age);
console.log (obj.city);
```

Functions are also objects in JavaScript

Passing function as a argument

- A function is an instance of the object type;
- A function behaves like any other object;
- We can store functions in a variable;
- We can pass a function as an argument to another function;
- we can return a function from a function.

↓

First-Class functions

```
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 2017 - el;
}

Var Result = arrayCalc(years, CalculateAge);
Console.log(Result);
```

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

var fullAges = arrayCalc(ages, isFullAge);
console.log(ages);
Console.log(isFullAge);

function maxHeartRate(el) {
    if(el >= 18 && el <= 81) {
        return Math.round(206.9 - (0.67*el));
    } else {
        return -1;
    }
}
Var HeartRate = arrCalc(years, maxHeartRate);
Console.log(HeartRate);
```

Function returning function

```
function interviewQuestion (job) {
    if (job === 'designer') {
        return function (name) {
            console.log (name + ', can yo please explain what is UX
                designing?');
        } 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 yo do?');
            }
        }
    }
var teacherQuestion = interviewQuestion ('teacher');

teacherQuestion ('John'); // what subject do yo teach, John?

interviewQuestion ('designer') ('mark');
```

immediately invoke function expressions (IIFE);

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

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

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

Closures

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 = 2016 - yearOfBirth;
        console.log( (retirementAge - age) + a);
    }
}
var retirementUS = retirement(66);
retirementUS(1990); // 40 years left until retirement.
var retirementGermany = retirement(65);
var retirementIceland = retirement(67);

retirementGermany(1990);
retirementIceland(1990);

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

Bind, Call, apply

```
var john = {
    name: 'John',
    age: 26,
    job: 'teacher'

    Presentation: function (style, timeOfDay) {

        if (style === 'formal') {
        console.log('Good' + timeOfDay + ', Ladies and gentlement! 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 + '.');
```

bind will return a function, so we need to store it in a variable

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

method borrowing

```
john.Presentation('formal', 'morning');
john.Presentation.call(emily, 'friendly', 'afternoon');
john.Presentation.apply(emily, ['friendly', 'afternoon']);
var johnFriendly = john.Presentation.bind(john, 'friendly')
johnFriendly('morning');
```

loops

```
//ES5
for (var i=0; i<boxesArr5.length; i++) {

    if (boxesArr5[i].className === 'box blue') {
        continue; // skips the iteration & continues (break it stops the);
    }
    boxesArr5[i].textContent = 'I Changed to ble!';
```

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

Array methods to find elements

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

map method
    var full = ages.map(function(cur) {
        return cur >= 18;
    });
    console.log(full);
console.log(full.indexOf(true));
console.log(ages[full.indexOf(true)]);
```

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

Console.log(ages.find(cur => cur => 18));
```

The spread operator

To expand an elements of an array.

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

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

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

ES5
- Fully supported in all modern browsers.
- Ready to be used today (2016)

ES6 / ES2015
- Partial support in modern browsers, not support in older browser;
- Can't use it in production today (2016)

ES2016 / ES2017
- Almost no support in modern browsers;
- Can't use it in production today (2016)

- Variable declarations with let and const
- Blocks and IIFEs
- Strings
- Arrow Functions
- Destructuring
- Array
- The spread operator
- Rest and Default Parameters
- Maps
- Classes and subclasses
- How to use ES2015 / ES6 today

Variable declarations with let and const

ES5

```
Var name5 = 'Jane Smith';
Var age5 = 23;
name5 = 'Jane Miller';

Console.log(name5);
```

→ function scope

ES6

```
Const name6 = 'Janesmith';
Let age6 = 23;
name6 = 'Jane miller';

blog,lc scope
Console.log(name6);
//error
```

if you ~~can~~ want to change the variable value in future we use `LET'

if you want to keep the variable value constant we use 'Const'

Driven licence function

```
function DrivoLicences5(PassedTest) {
    if (PassedTest) {
        Var Firstname = 'John';
        Var YearBirth = 1990;
        console.log(Firstname + ' born in ' + year of Birth + ' is now officially allowed to drive a car.');
    }
}
```

```
driversLicence5(true);

// ES6
function driverseLicence6(passTest) {

    if (passTest) {
        let firstName = 'John';
        const yearOfBirth = 1990;

        console.log(firstName + ', born in ' + yearOfBirth + ', is
        now officially allowed to drive a car.');
    }
}
```

if you write the consde out side if statement you will ge an error because let, const are block scope not a function scope

```
driveseLicence6(true);
```

But in some cases we need to use those variables out side the block scope

let & const you need to declare them inside the function not inside the if stements

Also if you ~~defin~~ Console log a variable before declaring, what output you will get?

"undefined"

But if you ~~d~~ do the in the ES6 with let(a) Const it will throwgh an error on your face.

```
let i =23;

for (let i=0; i< 5; i++) {
    Console.log (i);
}

Console.log (i);
```

```
// 0
   1
   2
   3
   4
   23
```

Assing value to i outside the for loop and inside for loop is Completely different variables)ik inside function outside the ~~Both are~~ function.

Blocks and IIFES

- Block is not just restricted to for loop, if statement or functions
- we can just creat a block just by writing curly braces

```
// ES6
{
  // this is a block
  const a = 1; let b = 2; var c = 3;
}
console.log(a+b); // Error we will get, because of block scope
console.log(c); // 3 because it is a var

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

console.log(c); // Error
```

Strings in ES6 / ES2015 (There is very big improvement in strings)

warning

new features

\template literals

we use backtick to write our entire string also the variables inside it.

```
let firstName = 'john';
let lastName = 'smith';

Const yearOfBirth = 1990;

function calcAge (year) {

    return 2016 - year;
}

// ES5
Console.log ('This is' + firstname + ' ' + lastName + '. He was
    born in' + yearOfBirth + '. Today, he is ' +
    + CalcAge (yearOfBirth) + ' years old.');

? This is Quite a work

// ES6                    backticks
Console.log (`This is ${firstName} ${lastName}. He was born in ${yea
    ofBirth}. Today, he is ${calcAge(yearsOfBirth)} yearold`;
```

other methods

```
const n = `${firstName} ${lastName}`;

n.startsWith("J");
console.log(n.startsWith('J')); // true (other value false)

console.log(n.endsWith('a')); // true (other value false)

Console.log(n.includes(' ')); //true

console.log(`${firstName}`.repeat(5));
```

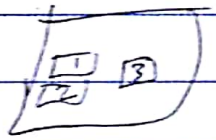
ARROW functions

```
Const years = [1990, 1965, 1982, 1937];

// ES5
var ages5 = years.map(function (el) {

    return 2016 - el;

});

console.log(ages5);

// ES6
let ages6 = years.map(el => 2016 - el);

console.log(ages6);

ages6 = years.map((el, index) => `Age element ${index + 1}: ${2016 - el}.`);

console.log(ages6);
```

operator

```
ages6 = years.map((el, index) => {

    Const now = new Date().getFullYear();

    Const age = now - el;

    return `age element ${index+1} : ${age}.`

});

Console.log(ages6);
```

Arrow Functions: lexical `this` keyword




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

Destructing

It is easy to extract data from a array (or) object and store it ina single variable

// ES5

```
Var john = ['John', 26];

Var name = john[0];
Var age = john[1];
```

// Es 6

```
Const[name, age] = ['John', 26];

Const obj = {
    firstName: 'John',
    lastName: 'Smith'
};

Const {firstName, lastName} = obj;

Console.log (firstName);
Console.log (lastName);
```

```
Const {firstName: a, lastName: b} = obj;

Console.log(a);
Console.log(b);

function calcAgeRetirement(year) {

    Const age = new Date().getFullYear() - year;
    return [age, 65 - age];

}

Const [age2, retirement] = calcAgeRetirement(1990);

Console.log(age2);
Console.log(retirement);
```

Arrays

```
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 =
        `dodgerblue`);
```

```
const familySmith = ['john', 'jane', 'mark'];
const familyMiller = ['mary', 'bob', 'ann'];
const bigFamily = [...familySmith, 'billy', ...familyMiller];
console.log(bigFamily);

const h = document.querySelector('h1');
const boxes = document.querySelectorAll('.box');

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

Rest Parameters

```
// ES5
function isFullAge5() {
    console.log(arguments);
}

isFullAge5(1990, 1999, 1965);
```

array like structure
not an array

```
var args = Array.prototype.slice.call(arguments);
array Arr.forEach(function(cur) {
    console.log(2016 - cur >= 18);
});
```

```
// ES6
function isFullAge6(...years) {
    console.log(years);
}

isFullAge(1990, 1999, 1965);
```

```
years.forEach(cur => console.log((2017 - cur) >= 18));
```

Default Parameters

```
//ES5
function SmithPerson(firstName, yearOfBirth, lastName, nationality) {
    this.firstName = firstName;
    this.yearOfBirth = yearOfBirth;
    this.lastName = lastName;
    this.nationality = nationality;
}
```

→ `lastName === undefined ? lastName = 'smith';`
`nationality === undefined ? nationality = 'american';`
`: nationality = nationality;`

```
var john = new SmithPerson('John', 1990);
var emily = new SmithPerson('Emily', 1993, 'Diaz', 'spanish');
```

```
//ES6
function SmithPerson(firstname, yearOfBirth, lastName='smith', nationality=
        'american') {
    this.firstname = firstname;
    this.lastName = lastName;
    this.yearOfBirth = yearOfBirth;
    this.nationality = nationality;
}
```

Maps [data structure]

```
Const question= new MaP();

    question.set('question', 'what is the official nume of the
                  lateste major Java Script verson?');

    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, pleasetry again');
```

In console

question →[Enter]

```
console.log
    (question.get('question'));

console.log (question.size);
// if(question.has(4)) {
   // question.delete(4);
// }
   // question.clear();
```

we can iterate the map not as like objects

```
question.forEach((value, key) => console.log(`this is ${key}, and it's set to ${value}`))
```

```
for(let [key, value] of question.entries()) {
    if (typeof(key) === 'number') {
        console.log(`Answer ${key}: ${value}`);
    }
}
```

```
const ans = parseInt(prompt('write the correct answer'))
console.log(question.get(ans === question.get('correct')),
```

Classes

```
//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');

//ES6 → they are hosted
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('Hey there!');
    }
}

const john6 = new person6 ('John', 1990, 'teacher');

person6.greeting();
```