

Apresentando ES6

Instrutor: Igor Cavalcanti
igor.wcavalcanti@dataprev.gov.br



Agenda

- Let, Var, Const (hoisting)
- for-in e for-of
- Parâmetros de Função extendidos
- Operador Spread
- Parâmetro Rest
- Destructuring assignment
- Funções seta (arrow functions)
- Classes
- Módulos

LET, VAR, CONST (hoisting)



LET, VAR, CONST (hoisting)

- **var**: Declara variáveis globais, podem ser usadas antes da declaração (hoisting)
- **let**: Declara variáveis com escopo definido. Só podem ser utilizadas depois de declaradas. Não podem ser redeclaradas no mesmo escopo.
- **const**: Declara variáveis com escopo definido. Só podem ser utilizadas depois de declaradas. Não podem ser redeclaradas no mesmo escopo. Não podem ter valores (atribuídos) modificados

LET, VAR, CONST (hoisting)

```
var exibeMensagem = function() {  
  var mensagemForaDoIf = 'externo';  
  
  if (true) {  
    var mensagemDentroDoIf = 'interno';  
    console.log(mensagemDentroDoIf); // interno  
  }  
  
  console.log(mensagemForaDoIf); // externo  
  console.log(mensagemDentroDoIf); // interno  
}  
exibeMensagem();  
  
# hoisting.js
```

LET, VAR, CONST (hoisting)

```
var exibeMensagem2 = function() {  
  var mensagemForaDoIf = 'externo';  
  
  if(true) {  
    let mensagemDentroDoIf = 'interno';  
    console.log(mensagemDentroDoIf); // interno  
  }  
  
  console.log(mensagemForaDoIf); // externo  
  console.log(mensagemDentroDoIf); // interno  
}  
exibeMensagem2();
```

hoisting2.js

LET, VAR, CONST (hoisting)

```
var a = 12; //acessivel globalmente
function myFunction() {
    console.log(a);
    var b = 13; //acessivel dentro da função
    if (true) {
        var c = 14; /////acessivel dentro da função
        console.log(b);
    }
    console.log(c);
}
myFunction();

// hoisting3.js
```

LET, VAR, CONST (hoisting)

```
let a = 12; //acessivel globalmente
function myFunction() {
    console.log(a);
    let b = 13; //acessivel dentro da funcao
    if (true) {
        let c = 14; //acessivel dentro do "if"
        console.log(b);
    }
    console.log(c);
}
MyFunction();

// hoisting4.js
```


LET, VAR, CONST (hoisting)

```
let a = 0;  
let a = 1; //TypeError  
function myFunction() {  
    let b = 2;  
    let b = 3; //TypeError  
    if (true) {  
        let c = 4;  
        let c = 5; //TypeError  
    }  
}  
myFunction();  
  
//hoisting5.js
```

LET, VAR, CONST (hoisting)

```
var a = 1;
let b = 2;

function myFunction() {
    var a = 3; //variavel diferente
    let b = 4; //variavel diferente
    if (true) {
        var a = 5; //sobreescrita
        let b = 6; //variavel diferente
        console.log(a);
        console.log(b);
    }
    console.log(a);
    console.log(b);
}
myFunction();
console.log(a);
console.log(b);
```

LET, VAR, CONST (hoisting)

```
const pi = 3.141;  
var r = 2;  
console.log(pi * r * r); //saida "12.564"  
pi = 12; //lança exceção  
  
//hoisting7.js
```

LET, VAR, CONST (hoisting)

```
const a = 12; //acessivel globalmente
function myFunction(){
  console.log(a);
  const b = 13; //acessivel dentro da funcao
  if(true){
    const c = 14; //acessivel no "if"
    console.log(b);
  }
  console.log(c);
}
myFunction();

//hoisting8.js
```

LET, VAR, CONST (hoisting)

```
const a = {  
  "name" : "John"  
};  
console.log(a.name);  
a.name = "Eden";  
console.log(a.name);  
a = {}; //throws read-only exception
```

```
//hoisting9.js
```

FOR-IN e FOR-OF



FOR-IN e FOR-OF

```
//For-in itera nas chaves
let iterable = [10, 20, 30];

for (let value in iterable) {
    console.log(value);
    value += 1;
    console.log(value);
}

//forinforon.js

// algo estranho na saída?
```

FOR-IN e FOR-OF

```
//For-of itera nos valores
let iterable = [10, 20, 30];

for (let value of iterable) {
    console.log(value);
    value += 1;
    console.log(value);
}

// algo estranho na saída?

//forinforon02.js
```


Parâmetros de Função Estendidos



Parâmetros de Função Estendidos

```
// ECMA 5

function myFunction(x, y,z)
{
    x = x === undefined ? 1 : x;
    y = y === undefined ? 2 : y;
    z = z === undefined ? 3 : z;
    console.log(x, y, z); //Output "6 7 3"
}
myFunction(6, 7);

//funcoes_01.js
```

Parâmetros de Função Estendidos

```
function myFunction(x = 1, y = 2, z = 3){  
    console.log(x, y, z); // Output "6 7 3"  
}  
myFunction(6,7);  
  
//funcoes_02.js
```

Parâmetros de Função Estendidos

```
function myFunction(x = 1, y = 2, z = 3){  
    console.log(x, y, z); // Output "1 7 9"  
}  
myFunction(undefined,7,9);  
  
//funcoes_03.js
```

Parâmetros de Função Estendidos

```
function myFunction(x = 1, y = 2, z = 3 + 5){  
    console.log(x, y, z); // Output "6 7 8"  
}  
myFunction(6,7);  
  
//funcoes_04.js
```

{...OperatorSpread}



Operador Spread

- O operador spread é representado pelo token '...'
- O operador spread quebra um objeto iterável em valores individuais
- O operador pode ser colocado em argumentos de funções ou onde se espera múltiplos elementos

Operador Spread

```
//Antes do ES6
function myFunction(a, b){
    return a + b;
}
var data = [1, 4];
var result = myFunction.apply(null, data);
console.log(result); //Output "5"
```

```
// spread_01.js
/*
https://www.w3schools.com/js/js\_function\_apply.asp
  O método "apply" pega o array, extrai seus valores e os
  passa individualmente para a função
*/
```


Operador Spread

```
//Depois do ES6
function myFunction(a, b){
    return a + b;
}
let data = [1, 4];
let result = myFunction(...data);
console.log(result); //Output "5"
```

```
// spread_02.js
```

```
/*
  O interpretador antes de chamar o método, substitui o
  "...data" por "1,4". O método "apply" não é chamado
  implicitamente
*/
```

Operator Spread

```
let array1 = [2,3,4];  
let array2 = [1];  
array2.push(...array1);  
console.log(array2); //Output "1, 2, 3, 4"
```

```
//spread_03.js
```

Operator Spread

```
let array1 = [1];  
let array2 = [2];  
let array3 = [...array1, ...array2, ...[3, 4]];  
let array4 = [5];
```

```
function myFunction(a, b, c, d, e)  
{  
  return a+b+c+d+e;  
}
```

```
let result = myFunction(...array3, ...array4);  
console.log(result); //Output "15"
```

```
//spread_04.js
```

Operador Spread

```
const obj = {a: 4, b : 5, c : 6};
```

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

```
const newObj = {...obj, a: 10};
```

```
console.log(newObj);
```

```
// Para rodar esse script, é necessário: node --harmony  
spread_05.js
```

```
// spread_05.js
```

Operador Spread

```
var obj1 = { foo: 'bar', x: 42 };  
var obj2 = { foo: 'baz', y: 13 };
```

```
var clonedObj = { ...obj1 };  
// Object { foo: "bar", x: 42 }
```

```
console.log(clonedObj);
```

```
var mergedObj = { ...obj1, ...obj2 };  
// Object { foo: "baz", x: 42, y: 13 }  
console.log(mergedObj);
```

```
// Para rodar esse script, é necessário: node --harmony  
spread_06.js
```

```
// spread_06.js
```

Parâmetros Rest



O parâmetro rest

- O operador rest é o token '...' no último argumento de uma função
- Ele é do tipo array

Parâmetro Rest

```
function myFunction(a, b, ...args){  
    console.log(args); //Output "3, 4, 5"  
}  
myFunction(1, 2, 3, 4, 5);  
  
// Para rodar: node --harmony parametro_rest.js  
  
// parametro_rest.js
```


Destructuring assignment



Destructuring assignment

- O 'destructuring assignment' é uma expressão que permite a atribuição de valores ou propriedades de um iterável ou objeto, para variáveis com uma sintaxe parecida com a construção de array e/ou objetos literais
- A expressão permite a extração de valores com uma sintaxe mais simples
- Já utilizado em linguagens como Perl ou Python
- Temos dois tipos:
 - Arrays
 - Objetos

array destructuring assignment

```
//Antes do ES6
var myArray = [1, 2, 3];
var a = myArray[0];
var b = myArray[1];
var c = myArray[2];

// Depois do ES6
let myArray = [1, 2, 3];
let a, b, c;
[a, b, c] = myArray; //array destructuring assignment
syntax

//Alternativa mais simples:
let [a, b, c] = [1, 2, 3];

//destructuring_01.js
```

array destructuring assignment

```
// ignorando valores
let [a, , b] = [1, 2, 3];
console.log(a);
console.log(b);

// usando operador rest (spread)
let [c, ...d] = [1, 2, 3, 4, 5, 6];
console.log(c);
console.log(Array.isArray(d));
console.log(d);

// valores também podem ser ignorados
let [e, , ,...f] = [1, 2, 3, 4, 5, 6];
console.log(e);
console.log(f);

//destructuring_02.js
```

array destructuring assignment

```
// valores default para variáveis  
let [a, b, c = 3] = [1, 2];  
console.log(c); //Output "3"  
  
// usando operador rest (spread)  
let [a, ...b] = [1, 2, 3, 4, 5, 6];  
console.log(a);  
console.log(Array.isArray(b));  
console.log(b);  
  
// valores também podem ser ignorados  
let [a, , ...b] = [1, 2, 3, 4, 5, 6];  
console.log(a);  
console.log(b);  
  
//destructuring_03.js
```

array destructuring assignment

```
// arrays aninhados
let [a, b, [c, d]] = [1, 2, [3, 4]];

// Como parâmetro de função
function myFunction([a, b, c = 3])
{
  console.log(a, b, c); //Output "1 2 3"
}
myFunction([1, 2]);

function myFunction([a, b, c = 3] = [1, 2, 3])
{
  console.log(a, b, c); //Output "1 2 3"
}
myFunction(undefined);
// destructuring_04.js
```

object destructuring assignment

```
// antes do ES6
var object = {"name" : "John", "age" : 23};
var name = object.name;
var age = object.age;

//Depois do ES6
let object = {"name" : "John", "age" : 23};
let name, age;
({name, age} = object); //object destructuring
assignment syntax
// destructuring_05.js
/*
```

Observação:

As variáveis com o mesmo nome dos membros do objeto
*/

object destructuring assignment

```
// variáveis com nomes diferentes das propriedades
let object = {"name" : "John", "age" : 23};
let x, y;
({name: x, age: y} = object);

// Forma mais simples:
let {name: x, age: y} = {"name" : "John", "age" : 23};

// destructuring_06.js
```


object destructuring assignment

```
// valores default para variáveis

let {a, b, c = 3} = {a: "1", b: "2"};
console.log(c); //Output "3"

// nomes de variáveis calculados:
let {"first"+"Name": x} = { firstName: "Eden" };
console.log(x); //Output "Eden"

//objetos aninhados
var {name, otherInfo: {age}} = {name: "Eden", otherInfo:
{age:
23}};
console.log(name, age); //Eden 23

// destructuring_07.js
```

object destructuring assignment

```
// Usando como parâmetro de função

function myFunction({name = 'Eden', age = 23, profession
="Designer"} = {})
{
  console.log(name, age, profession); //Output "John 23
Designer"
}
myFunction({name: "John", age: 23});

// destructuring_08.js
```

Funções seta (arrow functions)



Arrow functions

- Uma nova maneira de declarar funções usando o operador `'=>'`
- São funções anônimas
- O `'this'` em arrow functions
 - O `this` se refere ao escopo logo acima da definição da função
- Não pode ser usada como construtor de objetos

Arrow functions

```
// Sintaxe básica:
```

```
(param1, param2, ..., paramN) => { statements }
```

```
(param1, param2, ..., paramN) => expression
```

```
// equivalente à: => { return expression; }
```

```
// Parenteses são opcionais quando função só tem um  
parâmetro
```

```
(singleParam) => { statements }
```

```
singleParam => { statements }
```

```
// Quando a função não tem parâmetros
```

```
() => { statements }
```

Arrow functions

```
// normal
var circleArea = function(pi, r) {
    var area = pi * r * r;
    return area;
}
var result = circleArea(3.14, 3);
console.log(result); //Output "28.26"

//arrow function
let circleArea = (pi, r) => pi * r * r;
let result = circleArea(3.14, 3);
console.log(result); //Output "28.26"

//funcaoSeta_01.js
```

Arrow functions

```
// não é legal usar como método de objeto
```

```
var obj = {  
  i: 10,  
  b: () => console.log(this.i, this),  
  c: function() {  
    console.log(this.i, this);  
  }  
}
```

```
obj.b(); // prints undefined, Window {...} (or the  
global object)
```

```
obj.c(); // prints 10, Object {...}
```

```
//funcaoSeta_02.js
```

Classes



Classes

- O uso de classes no ES6 não introduz um novo modelo de orientação a objetos no javascript;
- As classes continuam como funções internamente
- Classes no ES6 é somente uma nova sintaxe para criação de uma função que é usada como construtor
- Classes podem ser definidas como:
 - declaração
 - expressão

Classes

```
// declaração de classe
class Student {
    constructor(name) {
        this.name = name;
    }
}
var s1 = new Student("Eden");
console.log(s1.name); //Output "Eden"

//classe_01.js
```

Classes

```
// Modo antigo
function Student(name){
    this.name = name;
}
var s1 = new Student("Eden");
console.log(s1.name); //Output "Eden"

//classe_02.js
```

Classes

```
// Classe == função

class Student{
    constructor(name){
        this.name = name;
    }
}

function School(name){
    this.name = name;
}

console.log(typeof Student);
console.log(typeof School == typeof Student);

//classe_03.js
```

Classes

```
// Expressão de classe

var Student = class {
    constructor(name){
        this.name = name;
    }
}

var s1 = new Student("Eden");
console.log(s1.name); //Output "Eden"

//classe_04.js
```

Classes

```
// Definição de Métodos
```

```
class Person
```

```
{  
    constructor(name, age){  
        this.name = name;  
        this.age = age;  
    }  
    printProfile(){  
        console.log("Name is: " + this.name + " and  
Age is: " +this.age);  
    }  
}
```

```
var p = new Person("Eden", 12)
```

```
p.printProfile();
```

```
console.log("printProfile" in p.__proto__);
```

```
console.log("printProfile" in Person.prototype);
```

Classes

```
// Definição de Métodos, método antigo
function Person(name, age){
    this.name = name;
    this.age = age;
}
Person.prototype.printProfile = function(){
    console.log("Name is: " + this.name + " and Age
is: " +this.age);
}
var p = new Person("Eden", 12)
p.printProfile();
console.log("printProfile" in p.__proto__);
console.log("printProfile" in Person.prototype);

//classe_06.js
```

Classes

```
// Definição de getter e setter
class Person
{
    constructor(name){
        this._name_ = name;
    }
    get name(){
        return this._name_;
    }
    set name(name){
        this._name_ = name;
    }
}
var p = new Person("Eden");
console.log(p.name);
p.name = "John";
console.log(p.name);
```


Classes

```
// Definição de getter e setter
// continuação
console.log(p.name);
console.log("name" in p.__proto__);
console.log("name" in Person.prototype);
console.log(Object.getOwnPropertyDescriptor(p.__proto__,
"name").set);
console.log(Object.getOwnPropertyDescriptor(Person.prototype,
"name").get);
console.log(Object.getOwnPropertyDescriptor(p,
"_name_").value);

//classe_07.js

/*
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global\_Objects/Object/getOwnPropertyDescriptor
*/
```

Módulos



Módulos

```
// lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

// someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

// otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

Módulos

```
// lib/mathplusplus.js
export * from "lib/math"
export var e = 2.71828182846
export default (x) => Math.exp(x)

// someApp.js
import exp, { pi, e } from "lib/mathplusplus"
console.log("e^{π} = " + exp(pi))
```

Módulos

```
export {variableName};  
export {variableName1, variableName2, variableName3};  
export {variableName as myVariableName};  
export {variableName1 as myVariableName1, variableName2  
as  
myVariableName2};  
export {variableName as default};  
export {variableName as default, variableName1 as  
myVariableName1,  
variableName2};  
export default function(){};  
export {variableName1, variableName2} from  
"myAnotherModule";  
export * from "myAnotherModule";
```

Módulos

```
import x from "module-relative-path";  
import {x} from "module-relative-path";  
import {x1 as x2} from "module-relative-path";  
import {x1, x2} from "module-relative-path";  
import {x1, x2 as x3} from "module-relative-path";  
import x, {x1, x2} from "module-relative-path";  
import "module-relative-path";  
import * as x from "module-relative-path";  
import x1, * as x2 from "module-relative-path";
```

Projeto Prático



Projeto ES6

- O que faremos:
 - Setar projeto Babel, numa aplicação ES5
 - Usando let
 - Usando Destructuring
 - Usando Arrow Functions
 - Setando o webpack
 - Usando Módulos
 - Usando Classes

Projeto ES6 - Setar projeto Babel, numa aplicação ES5

- Descompactar o arquivo es6-tutorial.zip
- Abrir o arquivo index.html no browser e clicar no botão Calculate .

Projeto ES6 - Setar projeto Babel, numa aplicação ES5

- Num terminal do diretório es6-tutorial
 - **digitar o comando:** `npm init`
- O comando criará o arquivo `package.json`
- Instalar os módulos `babel-cli` e `babel-core`
 - `npm install babel-cli babel-core --save-dev`
- Para ficar compatível com ES6, instalar o preset `babel-preset-es2015`
 - `npm install babel-preset-es2015 --save-dev`
- Instalar um servidor local para rodar a aplicação
 - `npm install http-server --save-dev`

Projeto ES6 - Setar projeto Babel, numa aplicação ES5

- Criar o diretório build no projeto
- Abrir o arquivo package.json, remover o que tiver na seção scripts e substituir pelo seguinte:

```
"scripts": {  
  "babel": "babel --presets es2015 js/main.js -o  
build/main.bundle.js",  
  "start": "http-server"  
},
```

Projeto ES6 - Setar projeto Babel, numa aplicação ES5

- Para rodar o babel e compilar o main.js, executar o comando:
 - `npm run babel`
- Alterar no arquivo index.html para referenciar a versão compilada do arquivo js/main.js
 - `<script src="build/main.bundle.js"></script>`
- Rodar o servidor
 - `npm start`

Projeto ES6 - Setar projeto Babel, numa aplicação ES5

- Caso a porta 8080 já esteja em uso alterar o package.json especificando uma porta disponível
- Por exemplo:

```
"scripts": {  
  "babel": "babel --presets es2015 js/main.js -o  
build/main.bundle.js",  
  "start": "http-server -p 9000"  
},
```

Projeto ES6 - Setar projeto Babel, numa aplicação ES5

- Se abrir o arquivo build/main.bundle.js, perceberá que o este arquivo é igual ao arquivo js/main.js

Projeto ES6 - Usando let para variaveis

- Na função `calculateMonthlyPayment`, temos a variável `monthlyRate` está disponível, mesmo antes de declará-la
- Substitua todas as ocorrências de `var` por `let` na função.
- Rodar:
 - `npm run babel`

Projeto ES6 - Usando let para variaveis

- Ao rodar a aplicação novamente, perceberá que o botão não funciona mais
- Ao inspecionar terá a mensagem:

`main.bundle.js:7` `Uncaught` `ReferenceError:`
`monthlyRate is not defined(...)`

Projeto ES6 - Usando let para variaveis

- No arquivo modificar a função existente para :

```
let calculateMonthlyPayment = function(principal, years, rate) {  
  let monthlyRate = 0;  
  if (rate) {  
    monthlyRate = rate / 100 / 12;  
  }  
  let monthlyPayment = principal * monthlyRate /  
    (1 - (Math.pow(1/(1 + monthlyRate), years * 12)));  
  return monthlyPayment;  
};
```

Projeto ES6 - Usando Destructuring

- No arquivo js/main.js, modificar o retorno da função calculateMonthlyPayment para:

```
return {principal, years, rate, monthlyPayment,  
monthlyRate};
```

- A instrução é um atalho para:

```
return { principal: principal,  
        years: years,  
        rate: rate,  
        monthlyPayment: monthlyPayment,  
        monthlyRate: monthlyRate };
```

Projeto ES6 - Usando Destructuring

- Abrir o index.html. Adicione o bloco `<h3>` abaixo para mostrar a taxa mensal logo abaixo do pagamento mensal
`<h3>Monthly Rate: </h3>`

Projeto ES6 - Usando Destructuring

- No arquivo main.js, no evento de click do botão, modifique a chamada da função `calculateMonthlyPayment` como abaixo:

```
let {monthlyPayment, monthlyRate} =  
calculateMonthlyPayment(principal, years, rate);
```

- É um atalho para:

```
var mortgage = calculateMonthlyPayment(principal,  
years, rate);  
var monthlyPayment = mortgage.monthlyPayment;  
var monthlyRate = mortgage.monthlyRate;
```

Projeto ES6 - Usando Destructuring

- Na última linha do tratamento de evento do botão adicionar a linha:

```
document.getElementById("monthlyRate").innerHTML =  
(monthlyRate * 100).toFixed(2);
```
- Compilar

```
npm run babel
```

Projeto ES6 - Usando arrow functions

- No arquivo `js/main.js` logo depois da função `calculateMonthlyPayment`, a função `calculateAmortization` definida abaixo:

```
let calculateAmortization = (principal, years,
rate) => {
    let {monthlyRate, monthlyPayment} =
calculateMonthlyPayment(principal, years, rate);
    let balance = principal;
    let amortization = [];
```

Projeto ES6 - Usando arrow functions

- Continuação:

```
for (let m=0; m<12; m++) {  
    let interestM = balance * monthlyRate; //Interest  
    payment for month  
    let principalM = monthlyPayment - interestM;  
    //Principal payment  
    interestY = interestY + interestM;  
    principalY = principalY + principalM;  
    balance = balance - principalM;  
}  
    amortization.push({principalY, interestY, balance});  
}  
return {monthlyPayment, monthlyRate, amortization};  
};
```

Projeto ES6 - Usando arrow functions

- Modifique a assinatura da função `calculateMonthlyPayment` como abaixo:

```
let calculateMonthlyPayment = (principal, years,  
rate) => {
```


Projeto ES6 - Usando arrow functions

- Modifique a assinatura do handler de evento de clique como segue:

```
document.getElementById('calcBtn').addEventListener('click', () => {
```

Projeto ES6 - Usando arrow functions

- Ainda no handler de clique, invoque a função `calculateAmortization` ao invés de `calculateMonthlyPayment`

```
let {monthlyPayment, monthlyRate, amortization} =  
calculateAmortization(principal, years, rate);
```

Projeto ES6 - Usando arrow functions

- Na última linha do handler de clique, logue os dados de amortização no console.
- Esses dados serão mostrados em uma tabela na aplicação posteriormente

```
amortization.forEach(month => console.log(month));
```

Projeto ES6 - Usando arrow functions

- A implementação completa do handler de clique:

```
document.getElementById('calcBtn').addEventListener('click', ()  
=> {  
    let principal = document.getElementById("principal").value;  
    let years = document.getElementById("years").value;  
    let rate = document.getElementById("rate").value;  
    let {monthlyPayment, monthlyRate, amortization} =  
calculateAmortization(principal, years, rate);  
    document.getElementById("monthlyPayment").innerHTML =  
monthlyPayment.toFixed(2);  
    document.getElementById("monthlyRate").innerHTML =  
(monthlyRate * 100).toFixed(2);  
    amortization.forEach(month => console.log(month));  
});
```

Projeto ES6 - Usando arrow functions

- Compile a aplicação novamente:
`npm run babel`

Projeto ES6 - Usando webpack

- Módulos estão disponíveis no JS através de bibliotecas.
- ES6 adicionou suporte nativo de módulos no JS
- Quando uma aplicação ES6 é compilada para ES5, o compilador usa bibliotecas de terceiros para implementar módulos
- Webpack e Browserify são opções populares, babel suporta as duas
- Vamos usar webpack

Projeto ES6 - Usando webpack

- Instalando o loader do babel e o webpack
`npm install babel-loader webpack@3.0.0 --save-dev`

Projeto ES6 - Usando webpack

- No arquivo package.json adicionar a entrada para webpack na seção scripts

```
"scripts": {  
  "babel": "babel --presets es2015 js/main.js -o  
build/main.bundle.js",  
  "start": "http-server",  
  "webpack": "webpack"  
},
```


Projeto ES6 - Usando webpack

- Criar o arquivo webpack.config.js na raiz do projeto. Contendo:

```
var path = require('path');  
var webpack = require('webpack');  
  
module.exports = {  
  entry: './js/main.js',  
  output: {  
    path: path.resolve(__dirname, 'build'),  
    filename: 'main.bundle.js'  
  },  
};
```

Projeto ES6 - Usando webpack

- Continuação:

```
module: {  
  loaders: [  
    {  
      test: /\.js$/,  
      loader: 'babel-loader',  
      query: {  
        presets: ['es2015']  
      }  
    }  
  ],  
},
```

Projeto ES6 - Usando webpack

- Continuação:

```
stats: {  
    colors: true  
  },  
  devtool: 'source-map'  
};
```

Projeto ES6 - Usando webpack

- Webpack usará o babel por debaixo dos panos para compilar a aplicação
- Webpack pode ser usado para construir uma aplicação mesmo que não use o módulos ES6
- Com a adição do webpack, o comando babel não se faz mais necessário
- Lembrete: A configuração feita não funciona para webpack 4.x
- A compilação pode ser feita agora com:
`npm run webpack`

Projeto ES6 - Usando Módulos

- Criar o arquivo mortgage.js no diretório js
- Copie as funções calculateMonthlyPayment e calculateAmortization do arquivo main.js para mortgage.js
- Adicione a palavra "export" em ambas as funções para torná-las disponíveis como parte da API pública do módulo mortgage.js

Projeto ES6 - Usando Módulos

- As funções:
 - `export let calculateMonthlyPayment ...`
 - `export let calculateAmortization ...`

Projeto ES6 - Usando Módulos

- Usando o módulo
 - No arquivo main.js, remover as funções calculateMonthlyPayment e calculateAmortization
 - Adicione o seguinte import na primeira linha do arquivo main.js para importar o módulo mortgage
- ```
import * as mortgage from './mortgage';
```

# Projeto ES6 - Usando Módulos

- Usando o módulo
    - No arquivo main.js, remover as funções calculateMonthlyPayment e calculateAmortization
    - Adicione o seguinte import na primeira linha do arquivo main.js para importar o módulo mortgage
- ```
import * as mortgage from './mortgage';
```


Projeto ES6 - Usando Módulos

- No evento de clique modificar a chamada da função `calculateAmortization` como abaixo:

```
let {monthlyPayment, monthlyRate, amortization} =  
    mortgage.calculateAmortization(principal, years,  
    rate);
```

- Compile e rode a aplicação

Projeto ES6 - Usando Classes

- Como é uma implementação alternativa ao invés de uma continuação lógica da implementação anterior, faça uma cópia do index.html e do main.js, no caso de querer voltar para a versão anterior
- No main.js, remover o import do início do arquivo.

Projeto ES6 - Usando Classes

- Adicione a seguinte definição de classes no início do arquivo:

```
class Mortgage {  
  
    constructor(principal, years, rate) {  
        this.principal = principal;  
        this.years = years;  
        this.rate = rate;  
    }  
  
    ...  
}
```

Projeto ES6 - Usando Classes

...

```
    get monthlyPayment() {  
        let monthlyRate = this.rate / 100 / 12;  
        return this.principal * monthlyRate / (1 -  
(Math.pow(1/(1 + monthlyRate),  
            this.years * 12))));  
    }
```

Projeto ES6 - Usando Classes

...

```
get amortization() {  
    let monthlyPayment = this.monthlyPayment;  
    let monthlyRate = this.rate / 100 / 12;  
    let balance = this.principal;  
    let amortization = [];  
    for (let y=0; y<this.years; y++) {  
        let interestY = 0;  
        let principalY = 0;
```

Projeto ES6 - Usando Classes

...

```
        for (let m=0; m<12; m++) {  
            let interestM = balance *  
monthlyRate;  
            let principalM = monthlyPayment -  
interestM;  
            interestY = interestY + interestM;  
            principalY = principalY +  
principalM;  
            balance = balance - principalM;  
        }
```

Projeto ES6 - Usando Classes

```
...
        amortization.push({principalY,
interestY, balance});
    }
    return amortization;
}
```

Projeto ES6 - Usando Classes

- Modifique o handler de evento de clique como a seguir:

```
document.getElementById('calcBtn').addEventListener('click', () => {  
    let principal =  
document.getElementById("principal").value;  
    let years = document.getElementById("years").value;  
    let rate = document.getElementById("rate").value;  
    let mortgage = new Mortgage(principal, years, rate);  
    document.getElementById("monthlyPayment").innerHTML =  
mortgage.monthlyPayment.toFixed(2);  
    ...  
});
```


Projeto ES6 - Usando Classes

- Modifique o handler de evento de clique como a seguir:

...

```
document.getElementById("monthlyRate").innerHTML =  
(rate / 12).toFixed(2);  
    let html = "";  
    mortgage.amortization.forEach((year, index) =>  
html += `  
        <tr>  
            <td>${index + 1}</td>  
            <td class="currency">$  
{Math.round(year.principalY)}</td>  
            <td class="stretch">
```

Projeto ES6 - Usando Classes

- Modifique o handler de evento de clique como a seguir:

...

```
        </div>
    </td>
    <td class="currency left">$
{Math.round(year.interestY)}</td>
    <td class="currency">$
{Math.round(year.balance)}</td>
    </tr>    `);
    document.getElementById("amortization").innerHTML
= html;
});
```

Projeto ES6 - Usando Classes

- Na linha de comando, digite o seguinte comando para fazer rebuild da aplicação:
- `npm run webpack`

Projeto ES6 - Usando Classes em módulos

- Para criar o módulo:
 - Criar o arquivo mortgage2.js no diretório js
 - Copiar a definição de classe do arquivo main.js para mortgage2.js
 - Adicionar a instrução 'export default' na frente da definição de classe.

```
export default class Mortgage { ...}
```

- Para usar o módulo:
 - No arquivo main.js, remover a definição da classe Mortgage
 - Importe o modulo mortgage no main.js

```
import Mortgage from './mortgage2';
```



Dúvidas?



