Semana 8

Fundamentos do Javascript: Objetos

Estrutura de objetos:

// Com ARRAY

const listaCPFs = ['21728172838','38738749827','29348274892'];

const cliente = ['nome','Andre','idade',36];

// Com OBJETO

const cliente2 = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

Acessando dados de um objeto:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

console.log(`Meu nome  é: ${cliente.nome}`);

console.log(cliente.idade);

console.log(cliente.cpf.substring(0,3));

console.log(cliente.email);

Meu nome  é: Andre

36

345

andre@email.com

na informação de cpf, colocamos para mostrar somente os 3 primeiros dígitos do cpf, com a ajuda do método substring(começa, termina não incluso)

acessando dados com colchetes:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

const chaves = ['nome', 'idade', 'cpf', 'email'];

console.log(cliente[chaves[0]]);

Andre

Utilizando a notação de colchetes para passar a variável

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

//const chaves = ['nome', 'idade', 'cpf', 'email'];

// console.log(cliente[chaves[0]]);

const chave = 'nome';

console.log(cliente[chave]);

podendo passar como variável dentro dos colchetes

pegando com ajuda do foreach:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

const chaves = ['nome', 'idade', 'cpf', 'email'];

chaves.forEach(chave => {

    console.log(cliente[chave]);

})

Quando se tenta acessar um atributo que não existe a resposta será underfined

console.log(cliente['conta']);

adicionando e alterando:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

cliente.telefone = 30302020;

console.log(cliente);

{

  nome: 'Andre',

  idade: 36,

  cpf: '34565436789',

  email: 'andre@email.com',

  telefone: 30302020

}

cliente.telefone = 898989898;

se a chave já exista no objeto o valor será alterado para o valor informado, logo o valor de telefone ficará sendo 8989.....,logo esse campo será atualizado

delete cliente.telefone

caso quisermos deletar um campo do objeto

Tipo de dados e valores:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com',

    fones: ['20202020', '30303030']

}

cliente.fones.forEach(fone => console.log(fone));

objetos em objetos:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com',

    fones: ['20202020', '30303030']

}

cliente.dependentes = {

    nome: 'Sara',

    parentesco: 'filha',

    dataNascimento: '20/03/2011'

}

console.log(cliente);

{

  nome: 'Andre',

  idade: 36,

  cpf: '34565436789',

  email: 'andre@email.com',

  fones: [ '20202020', '30303030' ],

  dependentes: { nome: 'Sara', parentesco: 'filha', dataNascimento: '20/03/2011' }

}

cliente.dependentes.nome = 'Sara Silva';

alterando o valor do valor da propriedade nome do objeto que está dentro de outro objeto

Listas de objetos:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com',

    fones: ['20202020', '30303030'],

    dependentes: [{

        nome: 'Sara',

        parentesco: 'filha',

        dataNascimento: '20/03/2011'

    }]

}

cliente.dependentes.push({

    nome: 'Samia Maria',

    parentesco: 'filha',

    dataNascimento: '04/01/2014'

})

console.log(cliente);

{

  nome: 'Andre',

  idade: 36,

  cpf: '34565436789',

  email: 'andre@email.com',

  fones: [ '20202020', '30303030' ],

  dependentes: [

    { nome: 'Sara', parentesco: 'filha', dataNascimento: '20/03/2011' },

    {

      nome: 'Samia Maria',

      parentesco: 'filha',

      dataNascimento: '04/01/2014'

    }

  ]

}

Incluindo valor em dependente

const filhaMaisNova =

cliente.dependentes.filter(dependente => dependente.dataNascimento === '04/01/2014')

console.log(filhaMaisNova[0].nome);

// Samia Maria

Filtrando para pegar somente o objeto que contenham o objeto dataNascimento dentro de dependente que tenha o valor ‘04/01/2014’

const filhaMaisNova =

cliente.dependentes.filter(dependente => dependente.dataNascimento === '04/01/2014')

console.log(filhaMaisNova);

[

  {

    nome: 'Samia Maria',

    parentesco: 'filha',

    dataNascimento: '04/01/2014'

  }

]

Aqui estamos pegando o valor completo sem passar nenhum parâmetro dentro do colchetes

Funções:

const cliente = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com',

    fones: ['20202020', '30303030'],

    dependentes: [

        {

            nome: 'Sara',

            parentesco: 'filha',

            dataNascimento: '20/03/2011'

        },

        {

            nome: 'Samia Maria',

            parentesco: 'filha',

            dataNascimento: '04/01/2014'

        }

    ],

    saldo: 100,

    // METODOS

    depositar: function(valor) {

        this.saldo += valor

    }

}

console.log(cliente.saldo); // 100

cliente.depositar(30)       // depositou 30

console.log(cliente.saldo); // 130

para criar um novo objeto através de outro objeto:

const objPersonagem = {

    nome: "Gandalf",

    classe: "mago",

    nivel: "20"

   }

   const objPersonagem2 = Object.create(objPersonagem)

   objPersonagem2.nome = "Gandalf, o Cinzento"

   console.log(objPersonagem.nome) //Gandalf

   console.log(objPersonagem2.nome) //Gandalf, o Cinzento

se não utilizar esse método, quando copiamos um objeto ao fazer uma alteração em algum objeto o objeto gêmeo também sofrerá a alteração

const objPersonagem = {

    nome: "Gandalf",

    classe: "mago",

    nivel: "20"

}

const objPersonagem2 = objPersonagem

objPersonagem2.nome = "Gandalf, o Cinzento"

console.log(objPersonagem.nome) //Gandalf, o Cinzento

console.log(objPersonagem2.nome) //Gandalf, o Cinzento

For ... in

Percorrer um objeto sem precisar do índice:

for (let info in cliente) {

    console.log(info);

}

// nome

// idade

// cpf

// email

// fones

// dependentes

// saldo

// depositar

para cada propriedade dentro de cliente jogue na variável info

for (let info in cliente) {

    relatorio += `${info} = ${cliente[info]}\n`

}

nome = Andre

idade = 36

cpf = 34565436789

email = andre@email.com

fones = 20202020,30303030

dependentes = [object Object],[object Object]

saldo = 100

depositar = function(valor) {

        this.saldo += valor

    }

for (let info in cliente) {

    if (typeof cliente[info] === 'object' || typeof cliente[info] === 'function') {

        continue

    } else {

        relatorio += `${info} = ${cliente[info]}\n`

    }

}

console.log(relatorio);

nome = Andre

idade = 36

cpf = 34565436789

email = andre@email.com

saldo = 100

limitando para somente aparecer os valores string e number, o uso do CONTINUE faz o código passar por ele sem acontecer nada ao código

Métodos do objeto:

const propsClientes = Object.keys(cliente)

console.log(propsClientes);

[

  'nome',

  'idade',

  'cpf',

  'email',

  'fones',

  'dependentes',

  'saldo',

  'depositar'

]

O uso do método Keys, vai criar um array a partir das chaves do objeto

function oferecerSeguro(obj) {

    const propsClientes = Object.keys(obj)

    if(propsClientes.includes('dependentes')) { //verifica sem tem esse valor dependente

        console.log(`Oferta de seguro de vida para ${obj.nome}`);

    }

}

oferecerSeguro(cliente);

// Oferta de seguro de vida para Andre

Primeiro estamos jogando todas as chaves de uma objeto na variável propsClientes, em seguida estamos verificando se esse objeto tem a chave com o nome dependente, caso tenha mostre uma mensagem no log

console.log(Object.values(cliente));

[

  'Andre',

  36,

  '34565436789',

  'andre@email.com',

  [ '20202020', '30303030' ],

  [

    { nome: 'Sara', parentesco: 'filha', dataNascimento: '20/03/2011' },

    {

      nome: 'Samia Maria',

      parentesco: 'filha',

      dataNascimento: '04/01/2014'

    }

  ],

  100,

  [Function: depositar]

]

Assim como temos um método object.keys para pegar as chaves, temos o object.values, para pegar os valores de um objeto

console.log(Object.entries(cliente));

[

  [ 'nome', 'Andre' ],

  [ 'idade', 36 ],

  [ 'cpf', '34565436789' ],

  [ 'email', 'andre@email.com' ],

  [ 'fones', [ '20202020', '30303030' ] ],

  [ 'dependentes', [ [Object], [Object] ] ],

  [ 'saldo', 100 ],

  [ 'depositar', [Function: depositar] ]

]

traz array com várias arrays dentro dele, e cada um sendo um conjunto de chaves e valores, em outra estrutura, array, índice 0 se refere a chave e índice 1 se refere ao valor

Sintaxe de espalhamento:

const clientes = [

    {

        nome: 'Andre',

        cpf: '12345678909',

        dependente: [

            {

                nome: 'Sara',

                parentesco: 'filha',

                dataNasc: '20/03/2011'

            },

            {

                nome: 'Samia',

                parentesco: 'filha',

                dataNasc: '04/01/2014'

            }

        ]

    },

    {

        nome: 'Juliana',

        cpf: '90987654321',

        dependente: [

            {

                nome: 'Sophia',

                parentesco: 'filha',

                dataNasc: '30/08/2020'

            }

        ]

    }

]

const listaDependentes = [...clientes[0].dependente, ...clientes[1].dependente]

console.log(listaDependentes);

[

  { nome: 'Sara', parentesco: 'filha', dataNasc: '20/03/2011' },

  { nome: 'Samia', parentesco: 'filha', dataNasc: '04/01/2014' },

  { nome: 'Sophia', parentesco: 'filha', dataNasc: '30/08/2020' }

]

... spradoperator, ele espalhou o conteúdo de cliente dentro da variável e pegou somente a chave dependente desse objeto

console.table(listaDependentes);

┌─────────┬──────────┬────────────┬──────────────┐

│ (index) │   nome   │ parentesco │   dataNasc   │

├─────────┼──────────┼────────────┼──────────────┤

│    0    │  'Sara'  │  'filha'   │ '20/03/2011' │

│    1    │ 'Samia'  │  'filha'   │ '04/01/2014' │

│    2    │ 'Sophia' │  'filha'   │ '30/08/2020' │

└─────────┴──────────┴────────────┴──────────────┘

console.table faz uma edição para facilitar a leitura do código

Para saber mais sobre JSON:

Formato JSON

{

    "id": 59,

    "titulo": "ECMAScript 6",

    "autor": "Diego Martins de Pinho",

    "categoria": "programação"

}

Formato XML

<livro id="59">

 <titulo>ECMAScript 6</titulo>

 <autor>Diego Martins de Pinho</autor>

 <categoria>programação</categoria>

</livro>

* JSON.parse(): converte JSON para um objeto JavaScript;
* JSON.stringify(): converte um objeto JavaScript para o formato JSON.

const jsonLivro = JSON.stringify({

id: 50,

titulo: "Primeiros Passos com NodeJS",

autor: "João Rubens",

categoria: "programação",

versoes: ["ebook", "impresso"]

})

console.log(jsonLivro)

{"id":50,"titulo":"Primeiros Passos com NodeJS","autor":"João Rubens","categoria":"programação","versoes":["ebook","impresso"]}

Para fazer o inverso

const objLivro = JSON.parse(jsonLivro)

console.log(objLivro)

{

id: 50,

titulo: 'Primeiros Passos com NodeJS',

autor: 'João Rubens',

categoria: 'programação',

versoes: [ 'ebook', 'impresso' ]

}

O que é prototype:

\_\_proto\_\_ -> guarda as definições de todos os arrays criado no Javascript,

Objeto literal

const cliente2 = {

    nome: 'Andre',

    idade: 36,

    cpf: '34565436789',

    email: 'andre@email.com'

}

Propriedades de prototypes:

function Cliente(nome, cpf, email, saldo) {

    this.nome = nome

    this.cpf = cpf

    this.email = email

    this.saldo = saldo

    this.depositar = function(valor) {

        saldo += valor

    }

}

Instanciando um novo cliente com André

function Cliente(nome, cpf, email, saldo) {

    this.nome = nome

    this.cpf = cpf

    this.email = email

    this.saldo = saldo

    this.depositar = function(valor) {

        saldo += valor

    }

}

const andre = new Cliente('Andre', '12345678909', 'andre@email.com', 100);

console.log(andre);

Cliente {

  nome: 'Andre',

  cpf: '12345678909',

  email: 'andre@email.com',

  saldo: 100,

  depositar: [Function (anonymous)]

}

const andre = new Cliente('Andre', '12345678909',

'andre@email.com', 100);

Com o uso do new Cliente, um novo objeto e criado

Manipulando prototype:

function Cliente(nome, cpf, email, saldo) {

    this.nome = nome

    this.cpf = cpf

    this.email = email

    this.saldo = saldo

    this.depositar = function(valor) {

        saldo += valor

    }

}

function ClientePoupanca(nome, cpf, email, saldo, saldoPoup) {

    Cliente.call(this, nome, cpf, email, saldo) // chamando as propriedade de Cliente

    this.saldoPoup = saldoPoup

}

const ju = new ClientePoupanca('Ju', '33344556677', 'ju@email.com', 100, 200);

console.log(ju);

ClientePoupanca {

  nome: 'Ju',

  cpf: '33344556677',

  email: 'ju@email.com',

  saldo: 100,

  depositar: [Function (anonymous)],

  saldoPoup: 200

}

Importante observar que no começo do construtor pegamos as informações de cliente com a ajuda do .call

ClientePoupanca.prototype.depositarPoup = function(valor) {

    this.saldoPoup += valor

}

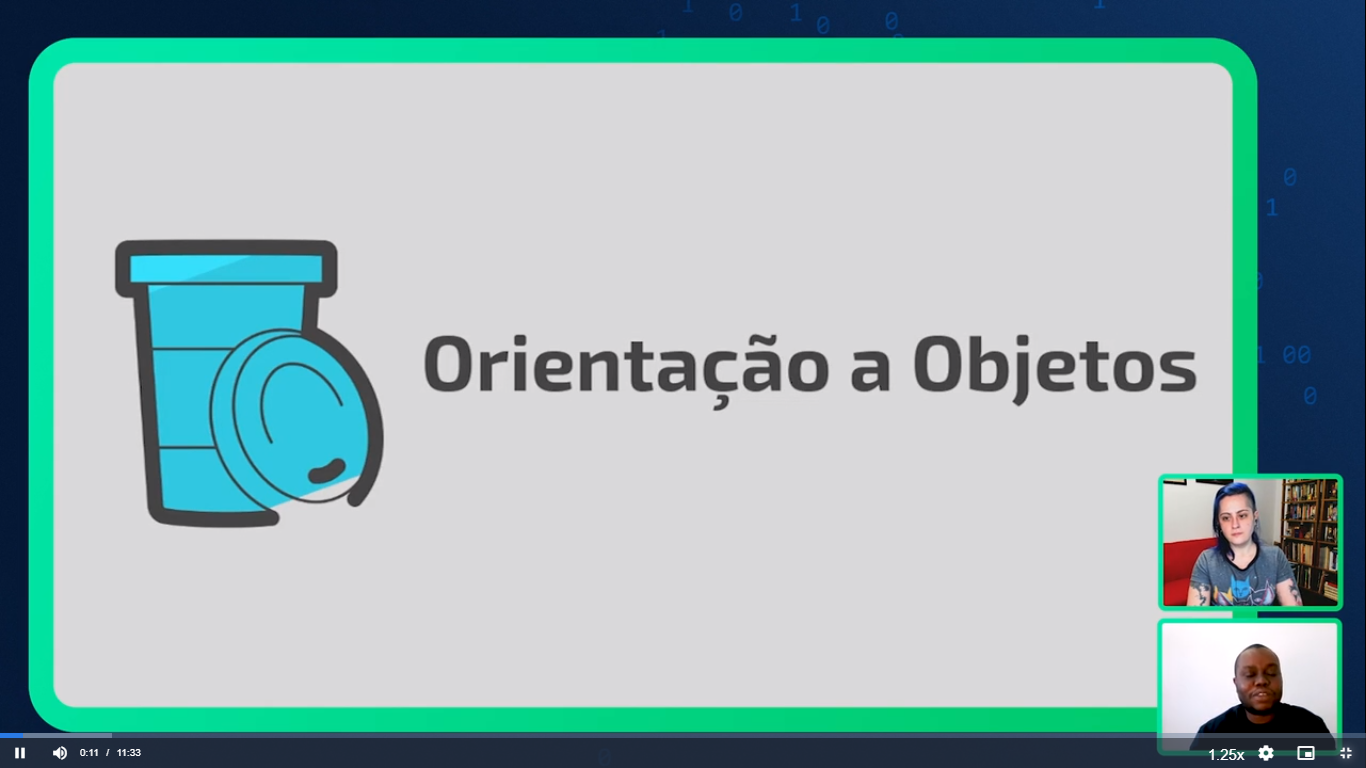
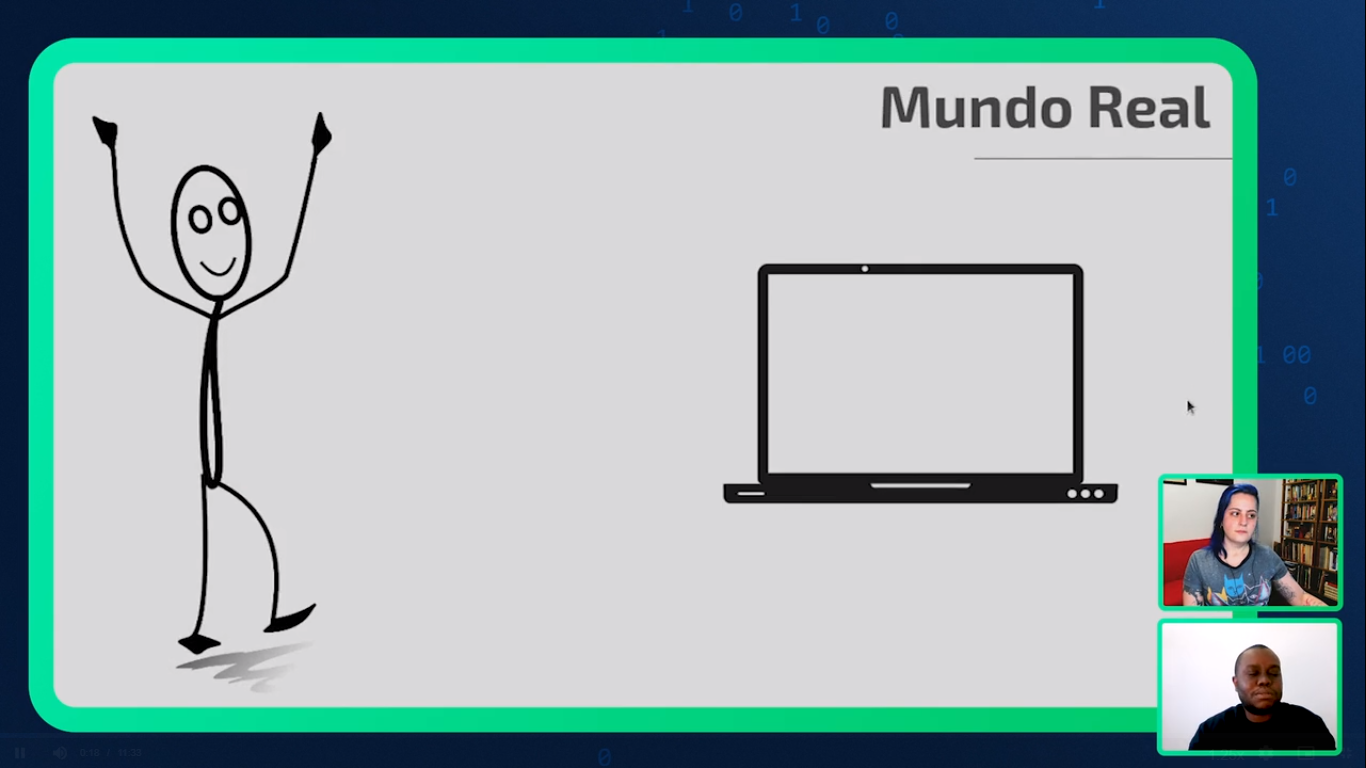
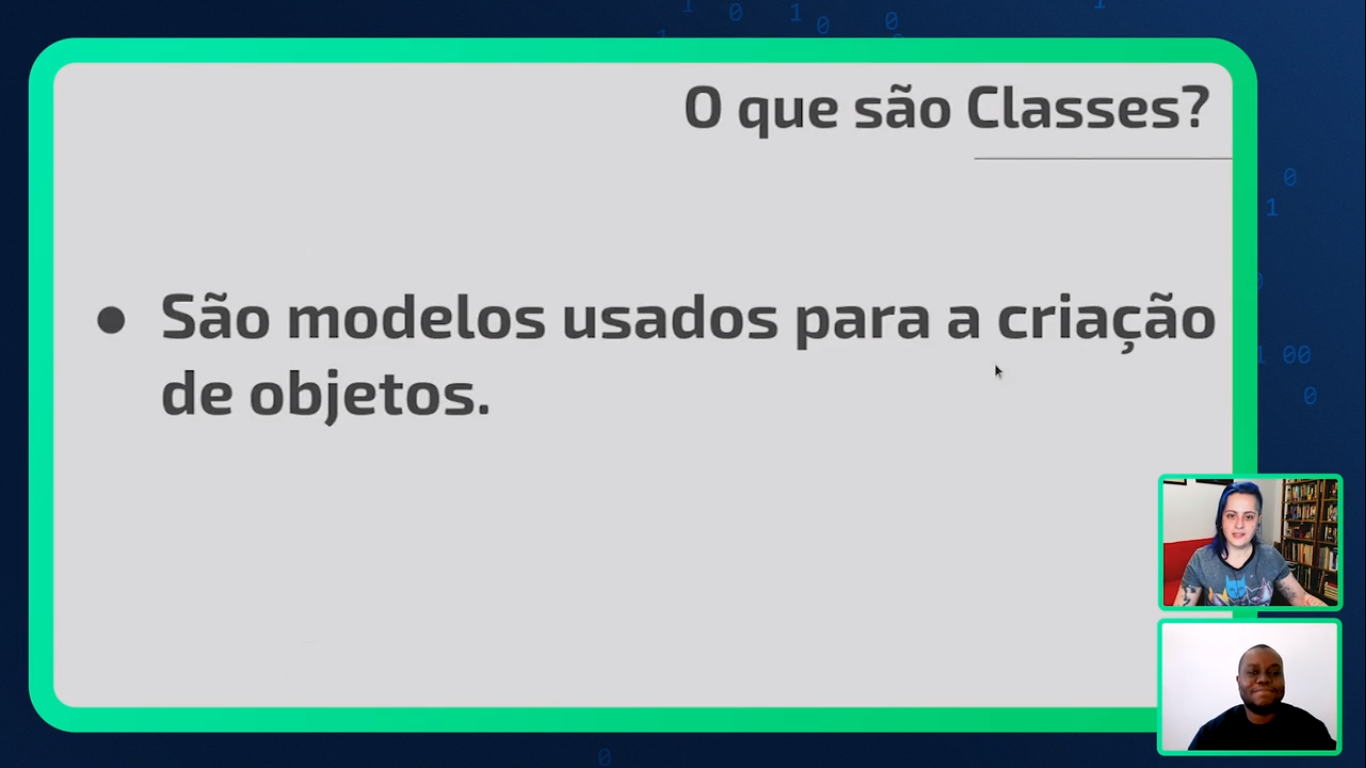
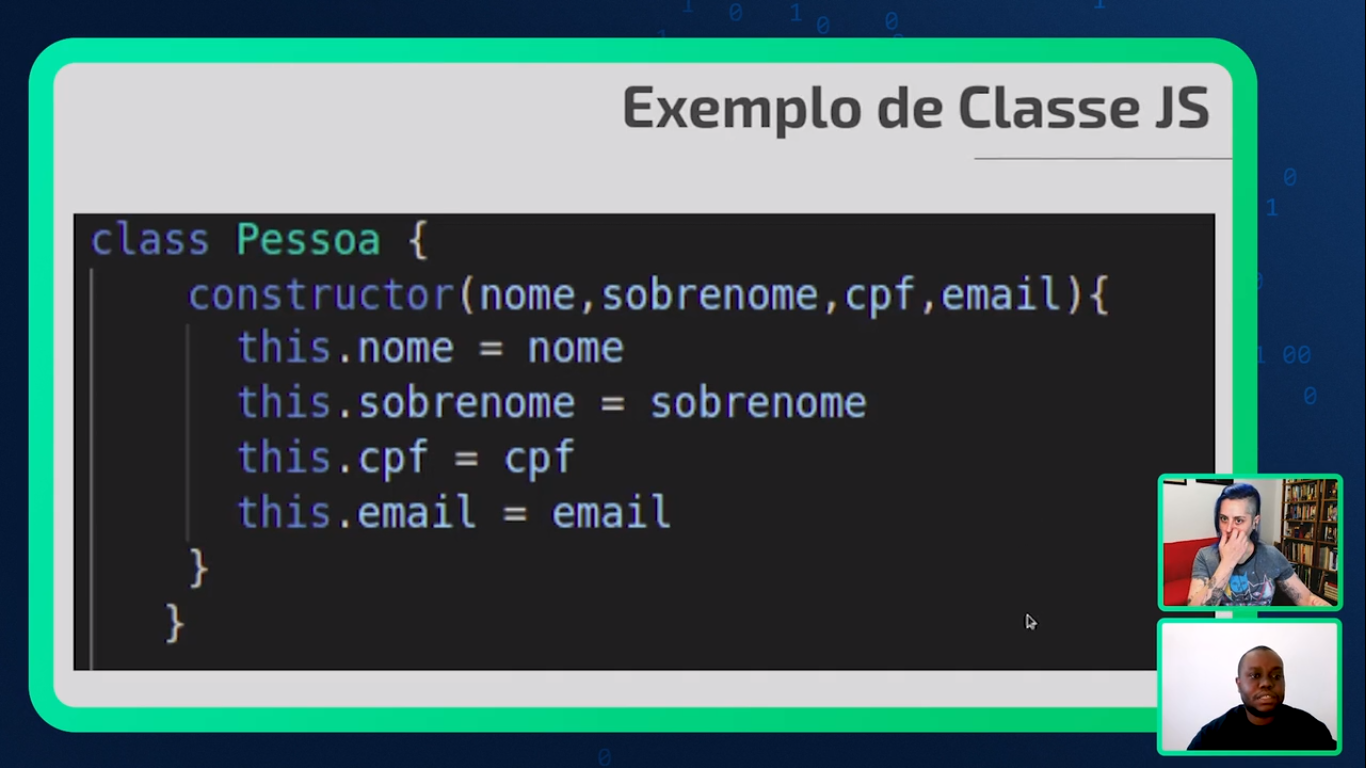
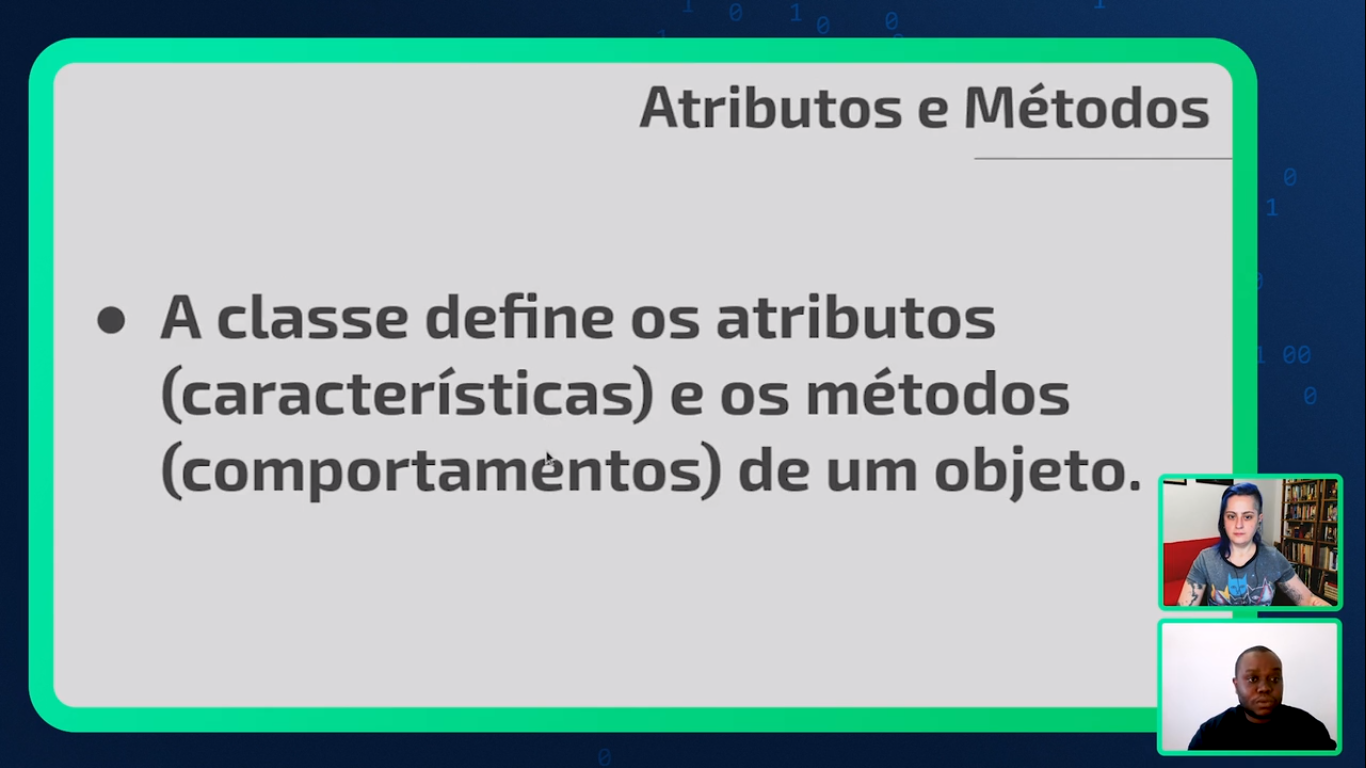
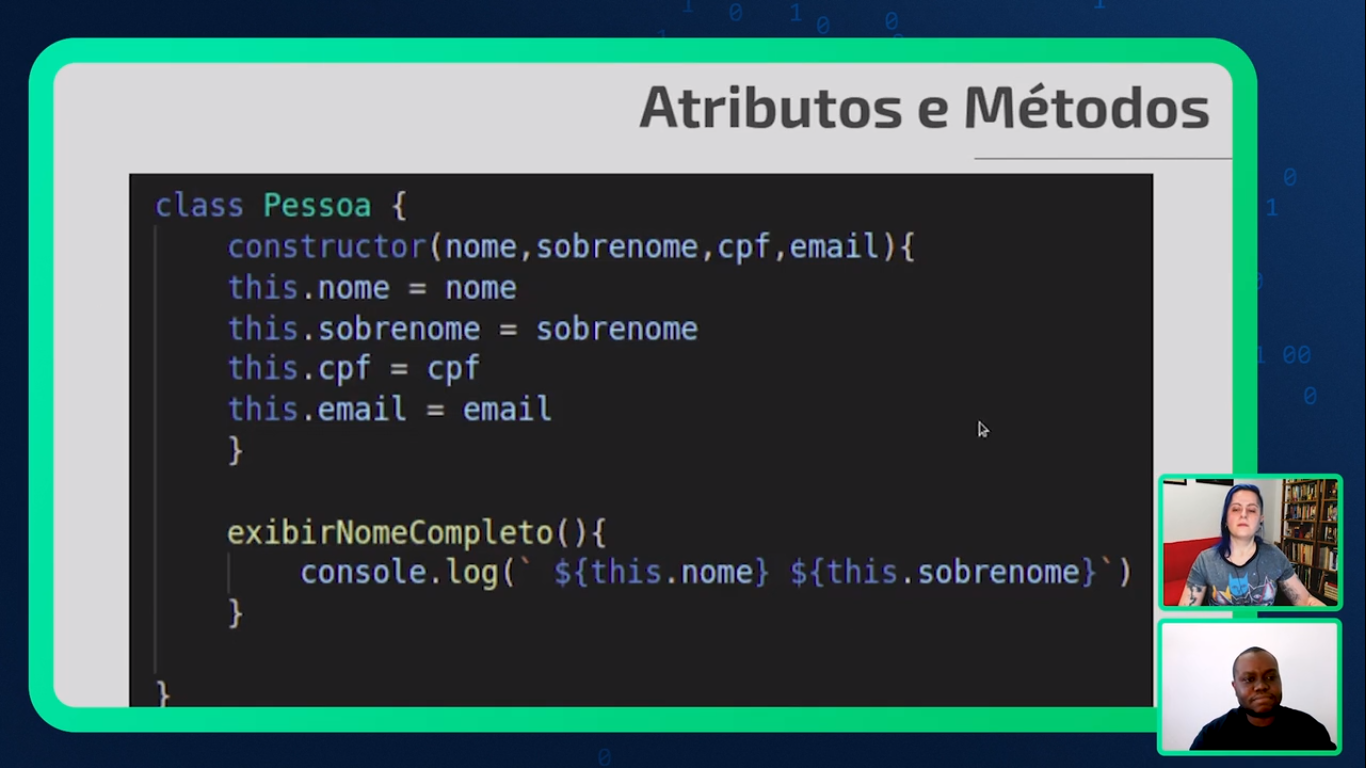
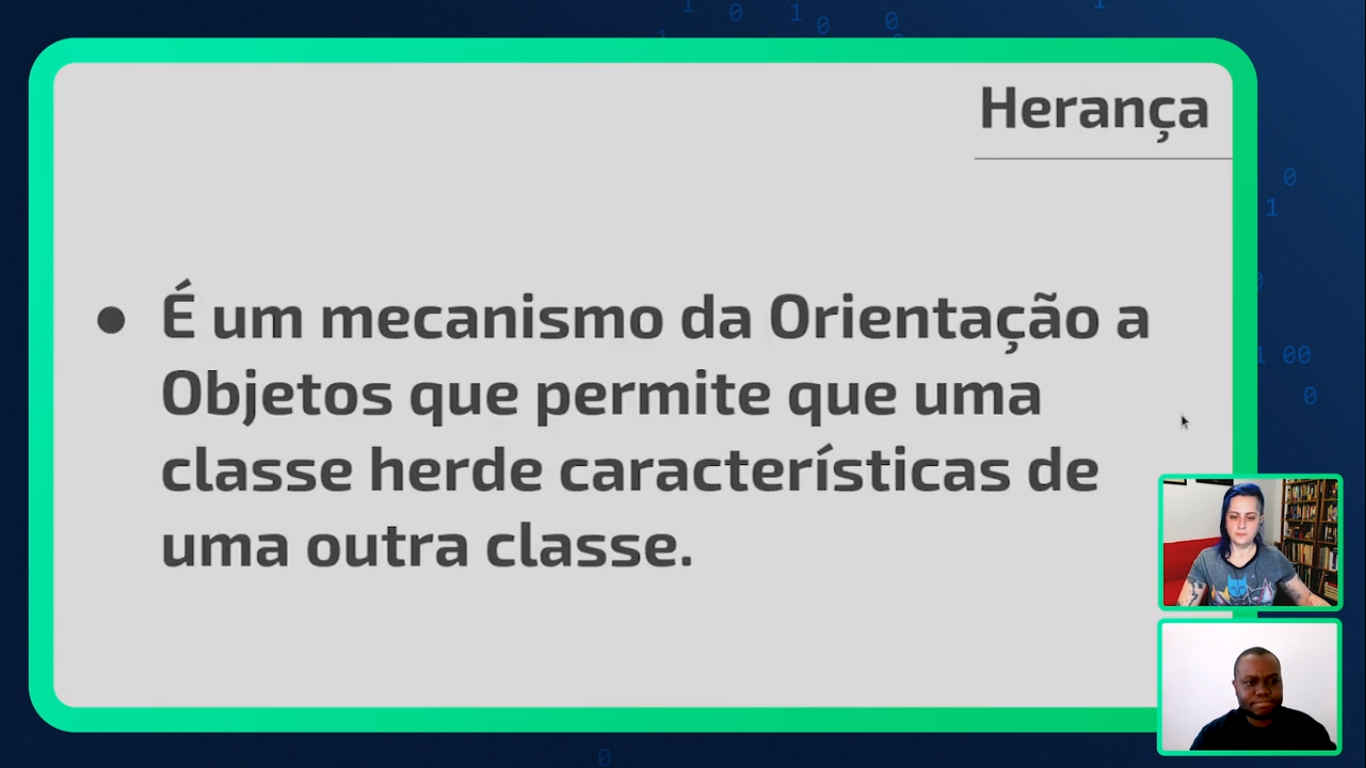
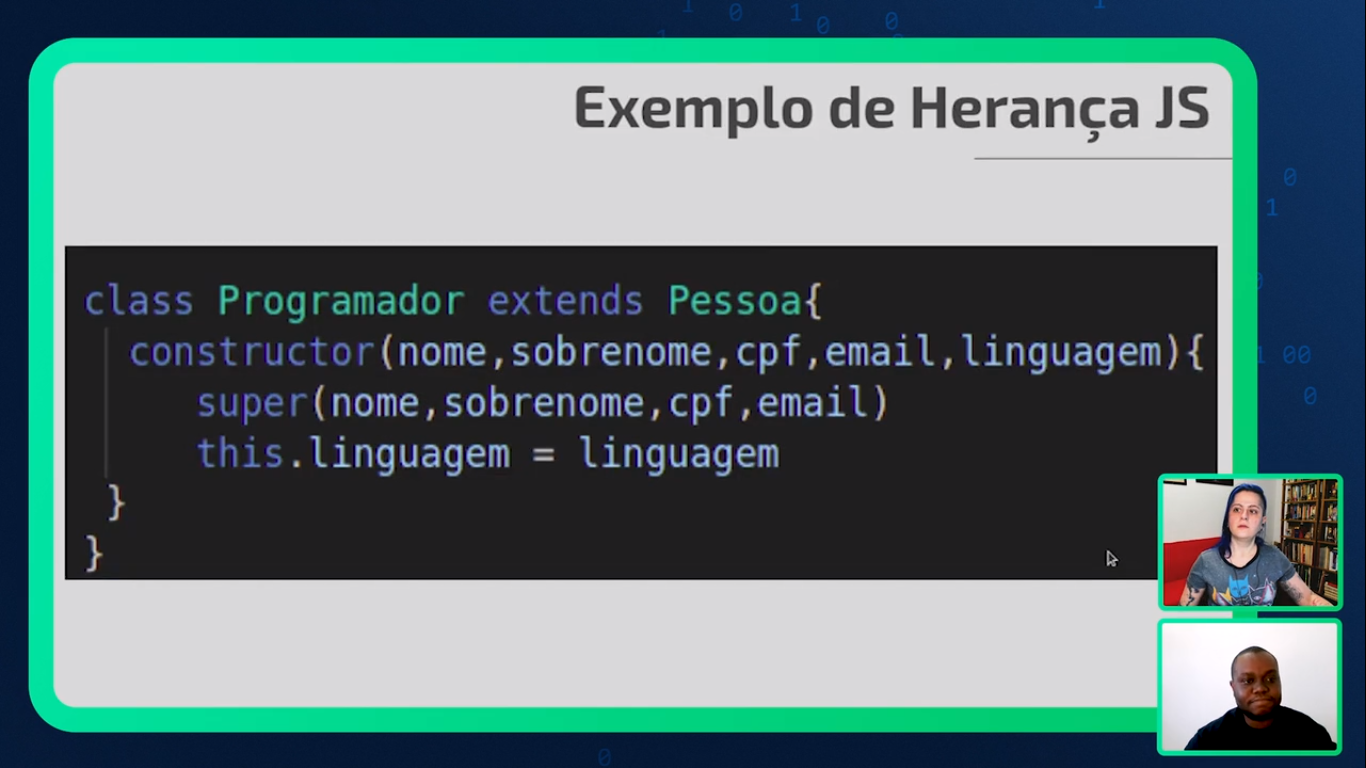
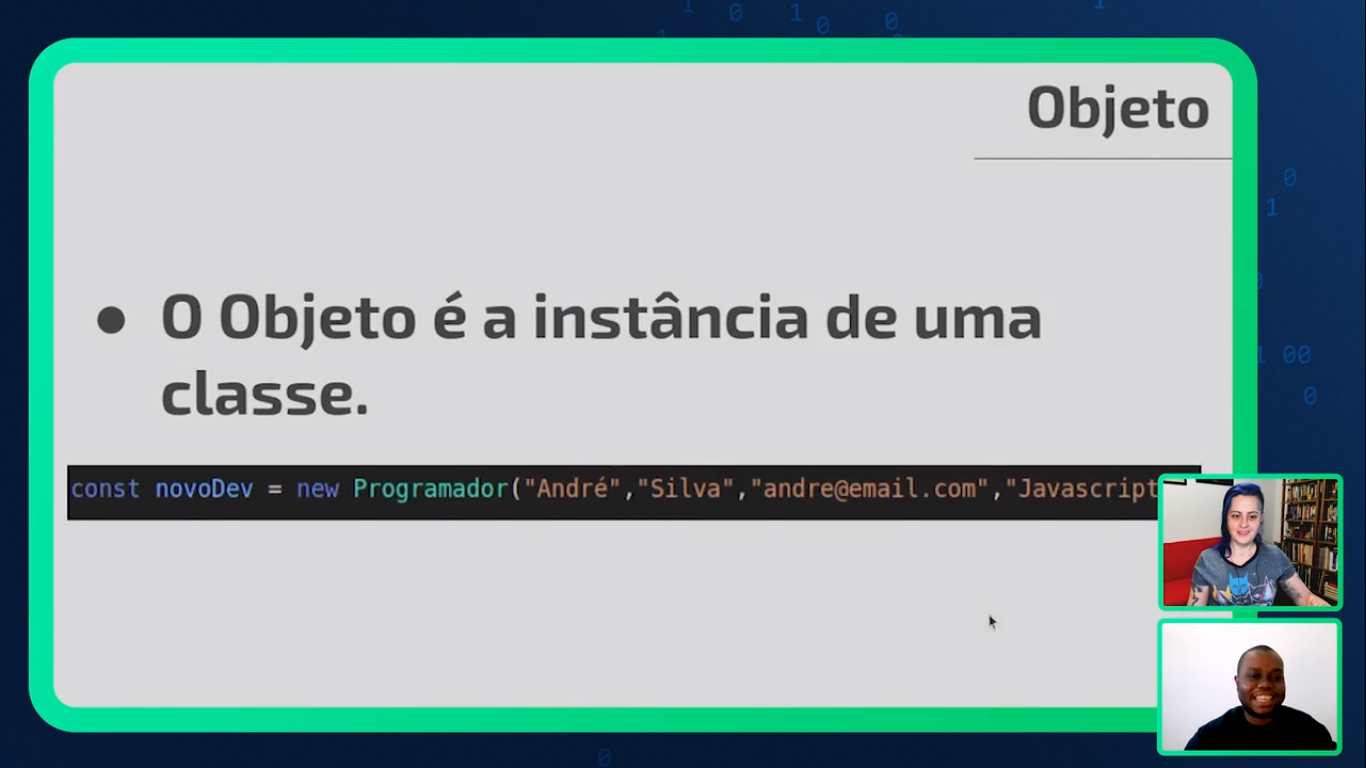
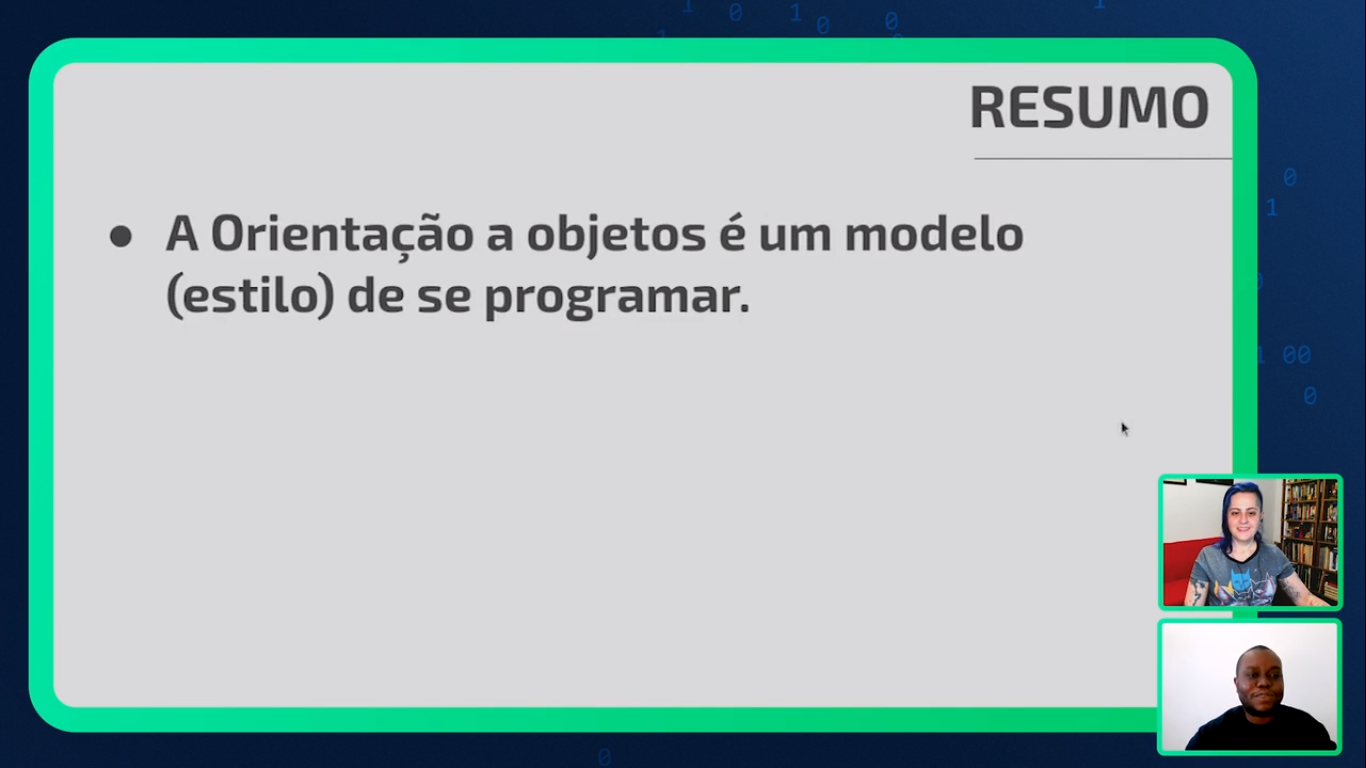
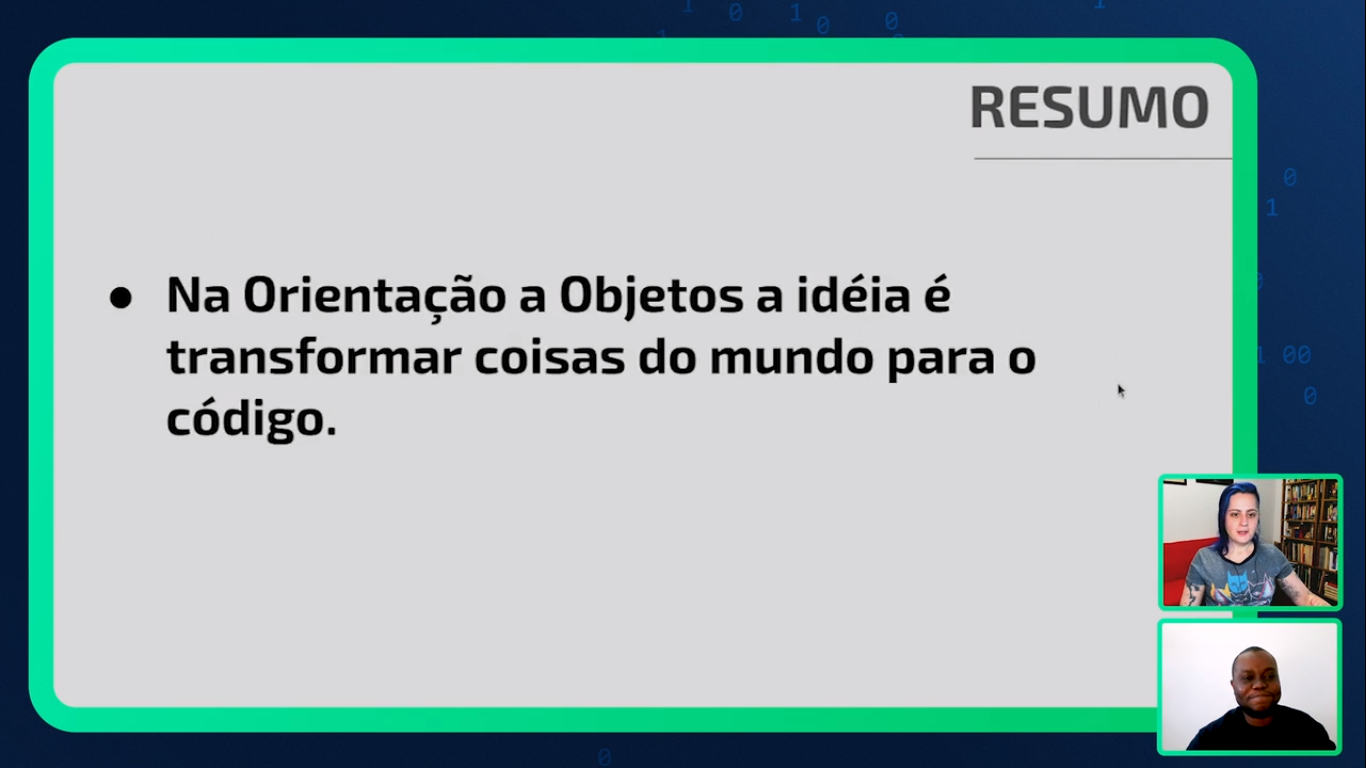
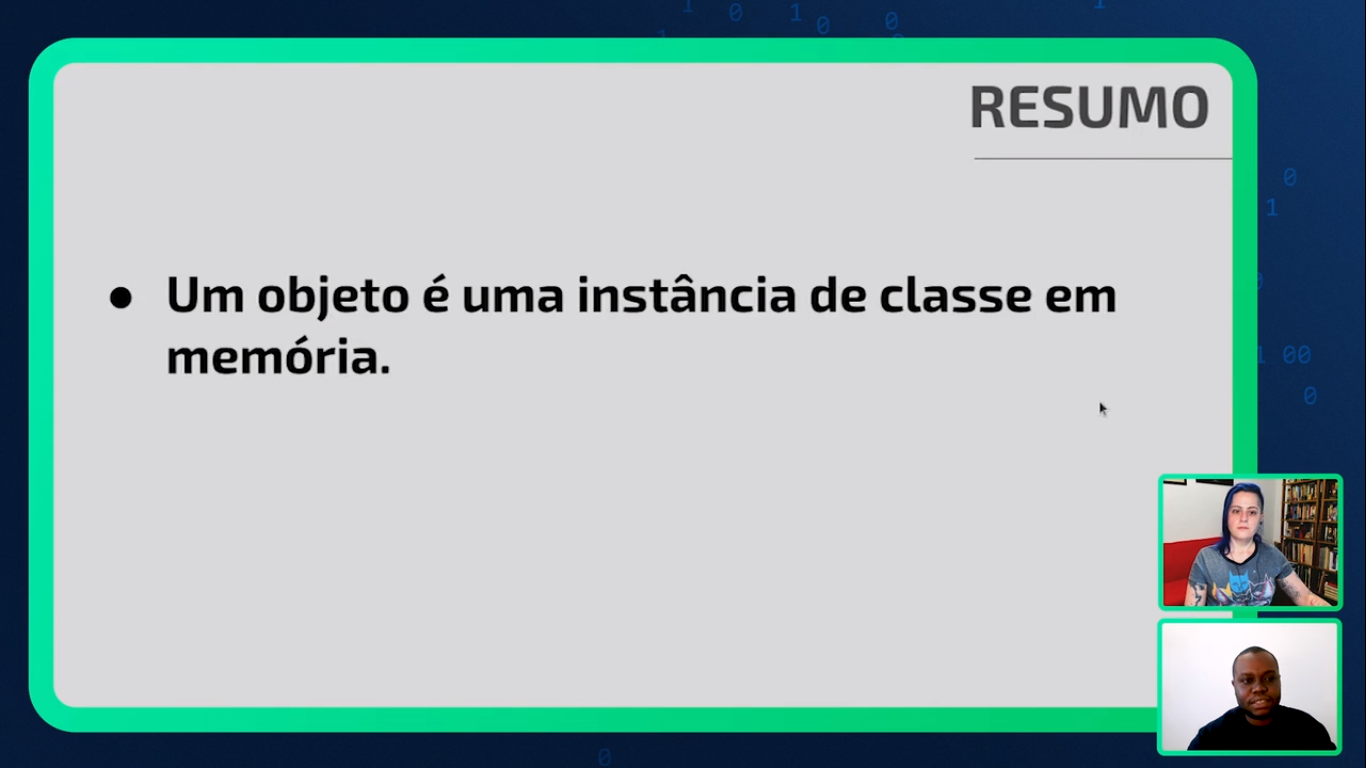
Dessa maneira estou definindo no prototypo um novo método, um novo comportamento para o ClientePoupanca

console.log(ju.saldoPoup); // 200

ju.depositarPoup(30)       // deposita 30

console.log(ju.saldoPoup); // 230

Conceito de OO (orientação a objeto):

Criando uma classe:

class Cliente {

    // Atributos - caracteristicas

    constructor(nome, email, cpf, saldo) {

        this.nome = nome

        this.email = email

        this.cpf = cpf

        this.saldo = saldo

    }

    // Metodos - comportamento

    depositar(valor) {

        this.saldo += valor

    }

    exibirSaldo() {

        console.log(this.saldo)

    }

}

const andre = new Cliente('Andre','andre@email.com','12345678909',100);

console.log(andre);

Cliente {

  nome: 'Andre',

  email: 'andre@email.com',

  cpf: '12345678909',

  saldo: 100

}

andre.depositar(50) // deposita 50

andre.exibirSaldo() // 150

Metodos e propriedades:

class Cliente {

    // Atributos - características

    constructor(nome, email, cpf, saldo) {

        this.nome = nome

        this.email = email

        this.cpf = cpf

        this.saldo = saldo

    }

    // Métodos - comportamento

    depositar(valor) {

        this.saldo += valor

    }

    exibirSaldo() {

        console.log(this.saldo)

    }

}

class ClientePoupanca extends Cliente{

    constructor(nome, email, cpf, saldo, saldoPoup){

        super(nome, email, cpf, saldo)

        this.saldoPoup = saldoPoup

    }

    depositarPoup(valor) {

        this.saldoPoup += valor

    }

}

const andre = new ClientePoupanca('Andre','andre@email.com','2255887744',100, 200);

console.log(andre);

ClientePoupanca {

  nome: 'Andre',

  email: 'andre@email.com',

  cpf: '2255887744',

  saldo: 100,

  saldoPoup: 200

}