**DECLARAÇÃO DE VARIÁVEL**

let nome ="Rafael"

**let**: A palavra-chave **let** é usada para declarar uma variável.

**nome**: É o nome da variável que está sendo declarada. Neste caso, a variável é chamada **nome**.

**"Rafael"**: É o valor atribuído à variável **nome**. Neste caso, é uma string contendo o nome "Rafael".

**IMPRESSÃO**

let nome ="Rafael"

console.log(nome)

**console.log** é uma função que imprime informações no console. Neste caso, estamos imprimindo o valor da variável **nome** no console.

**DIFERENÇA ENTRE VAR, LET, CONST**

var nome ="Rafael" //Escopo global

let nome ="Rafael" //Escopo por bloco

const nome ="Rafael" //Escopo por bloco, sem poder redefinir

**var**: A palavra-chave **var** tem escopo de função ou escopo global, o que significa que a variável **nome** será acessível em todo o escopo da função ou globalmente, dependendo de onde a declaração ocorre.

**let** possui escopo de bloco, o que significa que a variável **nome** só é acessível dentro do bloco em que foi declarada. Isso é especialmente útil em loops, condicionais e outras estruturas de controle de fluxo.

**const**: A palavra-chave **const** declara uma variável de escopo de bloco que não pode ser reatribuída. Ou seja, uma vez que um valor é atribuído a uma variável constante, esse valor não pode ser alterado. Além disso, **const** também tem escopo de bloco, assim como **let**.

É importante notar que ao usar **const**, você não pode reatribuir um novo valor à variável. No entanto, se a variável é um objeto ou um array, as propriedades ou elementos internos podem ser modificados, mas a atribuição da variável a um novo objeto ou array não será permitida.

**OPERADORES MATEMÁTICOS**

let num1 = 5, num2 = 10

soma = num1 + num2 //Soma

console.log(soma)

mult = soma\*2 //Multiplicação

console.log(mult)

div = mult / soma //Divisão

console.log(div)

rest = mult % soma //Resto da divisão

console.log(rest)

num1++ //Incrementa de 1 em 1

console.log(num1)

num2-- //Decrementa de 1 em 1

console.log(num2)

num1 += 5 //Soma 5 na variável num1(num1 = num1 + 5)

console.log(num1)

num2 -= 5 //Subtrai 5 na variável num1(num1 = num1 - 5)

console.log(num2)

Duas variáveis, **num1** e **num2**, são declaradas e inicializadas com os valores 5 e 10, respectivamente.

A soma dos valores de **num1** e **num2**, e o resultado é atribuído à variável **soma**.

A multiplicação do valor de **soma** por 2 é armazenada na variável **mult**.

A divisão do valor de **mult** pelo valor de **soma** é armazenada na variável **div**.

O operador **%** retorna o resto da divisão entre **mult** e **soma**, que é armazenado na variável **rest**.

A operação **num1++** incrementa o valor de **num1** em 1

A operação **num2--** decrementa o valor de **num2** em 1

A operação **num1 += 5** adiciona 5 ao valor de **num1** (é uma forma abreviada de **num1 = num1 + 5**).

A operação **num2 -= 5** subtrai 5 do valor de **num2** (é uma forma abreviada de **num2 = num2 - 5**).

**MAIOR, MENOR, IGUAL, DIFERENTE**

let num1 = 10, num2 = 5, num3 = 10

maior = num1 > num2 // Maior do que ...

console.log(maior)

menor = num1 < num2 // Menor do que ...

console.log(menor)

maiorIgual = num1 >= num3 // Maior ou igual que ...

console.log(maiorIgual)

menorIgual = num2 <= num3 // Menor ou igual que ...

console.log(menorIgual)

Igual = num1 == num3 // Igual a ...

console.log(Igual)

Diferente = num1 != num3 //Diferente de ...

console.log(Diferente)

Três variáveis (**num1**, **num2** e **num3**) são declaradas e inicializadas com os valores 10, 5 e 10, respectivamente.

A expressão **num1 > num2** é avaliada e o resultado (**true** ou **false**) é atribuído à variável **maior**. Essa linha compara se **num1** é maior que **num2**.

A expressão **num1 < num2** é avaliada e o resultado (**true** ou **false**) é atribuído à variável **menor**. Essa linha compara se **num1** é menor que **num2**.

A expressão **num1 >= num3** verifica se **num1** é maior ou igual a **num3.**

A expressão **num2 <= num3** verifica se **num2** é menor ou igual a **num3**

A expressão **num1 == num3** verifica se **num1** é igual a **num3**. Importante notar que **==** verifica apenas a igualdade de valor, não considerando o tipo.

A expressão **num1 != num3** verifica se **num1** é diferente de **num3**.

**E, OU, NÃO**

let n1 = 10, n2 = 5, n3 = 15, n4 = 2

E = n1 > n2 && n1 > n3 //&& = e

console.log(E)

Ou = n1 > n2 || n1 > n3 //|| = ou

console.log(Ou)

not = !(n1 > n2) //! = não

console.log(not)

Quatro variáveis (**n1**, **n2**, **n3** e **n4**) são declaradas e inicializadas com os valores 10, 5, 15 e 2, respectivamente.

A expressão **n1 > n2 && n1 > n3** verifica se **n1** é maior que **n2** e, ao mesmo tempo, **n1** é maior que **n3**. O operador lógico **&&** (E) retorna **true** se ambas as condições forem verdadeiras.

A expressão **n1 > n2 || n1 > n3** verifica se **n1** é maior que **n2** ou **n1** é maior que **n3**. O operador lógico **||** (OU) retorna **true** se pelo menos uma das condições for verdadeira.

A expressão **!(n1 > n2)** verifica se **n1** não é maior que **n2**. O operador lógico **!** (NÃO) inverte o valor de verdadeiro para falso e vice-versa.

**INCREMENTAÇÃO**

let n1 = n2 = n3 =10

n1++ //Incrementação

console.log(n1)

console.log(++n2) //Impressão de pré-incremento

console.log(n3++) //Impressão de pós-incremento

console.log(n3)

Três variáveis (**n1**, **n2** e **n3**) são declaradas e inicializadas com o valor 10.

**n1++** é um operador de pós-incremento, o que significa que primeiro usa o valor atual de **n1** na expressão e, em seguida, incrementa **n1** por 1. Após essa operação, **n1** será igual a 11.

O operador **++** antes da variável (**++n2**) é um operador de pré-incremento, o que significa que primeiro incrementa **n2** por 1 e, em seguida, usa esse novo valor na expressão. Portanto, **++n2** imprime 11 no console.

**n3++** é um operador de pós-incremento, então primeiro usa o valor atual de **n3** na expressão e, em seguida, incrementa **n3** por 1. Portanto, **n3++** imprime 10 no console. No entanto, se você imprimir novamente o valor de **n3**, será 11, porque o operador de pós-incremento já foi aplicado anteriormente.

**OPERADOR TERNÁRIO**

let num = 10

res = (!(num % 2) ? "Par" : "Impar") //? é um operador ternário

console.log(res)

Uma variável chamada **num** é declarada e inicializada com o valor 10.

Aqui está o uso do operador ternário (**? :**). Vamos quebrar a expressão:

* **num % 2**: Isso calcula o resto da divisão de **num** por 2. Se o resto for 0, significa que **num** é par; se o resto for diferente de 0, **num** é ímpar.
* **!(num % 2)**: O operador **!** inverte o valor booleano resultante do cálculo do resto. Então, se **num** for par, **!(num % 2)** será **true**, e se for ímpar, será **false**.
* **(!(num % 2) ? "Par" : "Impar")**: O operador ternário avalia a condição entre parênteses. Se **(!(num % 2))** for verdadeiro (ou seja, **num** é par), então a expressão retorna "Par"; caso contrário, retorna "Impar".

O resultado final dessa expressão é atribuído à variável **res**.

**TYPEOF**

let v1 = 10

let v2 = "10"

let v3 = v1 === v2

let v4 = {nome: "Rafael"}

console.log("Valor: " + v1 + " - Tipo: " + typeof(v1)) //Impressão do valor e tipo da variável

console.log("Valor: " + v2 + " - Tipo: " + typeof(v2)) //Impressão do valor e tipo da variável

console.log("Valor: " + v3 + " - Tipo: " + typeof(v3)) //Impressão do valor e tipo da variável

console.log("Valor: " + v4 + " - Tipo: " + typeof(v4)) //Impressão do valor e tipo da variável

**v1** é uma variável que armazena o número 10.

**v2** é uma variável que armazena a string "10".

**v3** é uma variável que armazena o resultado da comparação estrita (**===**) entre **v1** e **v2**. Neste caso, **v3** será **false** porque **v1** é um número e **v2** é uma string, mesmo que seus valores sejam semelhantes.

**v4** é uma variável que armazena um objeto com uma propriedade chamada **nome** contendo o valor "Rafael".

O operador **typeof** é usado para obter o tipo de uma variável.

**ESPALHADOR**

let n1 = [10, 20, 30]

let n2 = [11, 22, 33, 44, 55]

let n3 = [...n1, ...n2] //Espalhador

console.log("n1: " + n1)

console.log("n2: " + n2)

console.log("n3: " + n3)

Dois arrays são declarados e inicializados com valores. **n1** contém os elementos **[10, 20, 30]** e **n2** contém **[11, 22, 33, 44, 55]**.

O operador de espalhamento (**...**) é utilizado para criar um novo array chamado **n3**. Ele contém todos os elementos de **n1** seguidos pelos elementos de **n2**. Portanto, **n3** terá o valor **[10, 20, 30, 11, 22, 33, 44, 55]**.

**IF, IF ELSE**

let num = 10

if (num > 10) {

    console.log("Numeral maior que 10")

} else if (num > 5){

    console.log("Numeral está entre 6 e 10")

} else {

    console.log("Numeral menor ou igual que 10")

}

console.log("Fim do programa")

Uma variável chamada **num** é declarada e inicializada com o valor 10.

Aqui está a estrutura condicional **if-else**:

* A primeira condição **if** verifica se **num** é maior que 10. Se essa condição for verdadeira, a mensagem "Numeral maior que 10" é impressa no console.
* Se a primeira condição não for verdadeira, a condição **else if** é verificada. Esta condição verifica se **num** é maior que 5. Se essa condição for verdadeira (mas a primeira não foi), a mensagem "Numeral está entre 6 e 10" é impressa no console.
* Se ambas as condições anteriores forem falsas, a parte **else** é executada, e a mensagem "Numeral menor ou igual que 10" é impressa no console.

**SWITCH**

let colocacao = 7

switch (colocacao) {

    case 1:

        console.log("Primeiro Lugar")

        break;

    case 2:

        console.log("Segundo Lugar")

        break

    case 3:

        console.log("Terceiro Lugar")

        break

    case 4: case 5: case 6:

        console.log("Prêmio de participação")

        break

    default:

        console.log("Não subiu ao pódio")

        break;

}

Uma variável chamada **colocacao** é declarada e inicializada com o valor 7.

Aqui está a estrutura **switch**:

* O valor de **colocacao** é avaliado nas diferentes cláusulas **case**.
* Se **colocacao** for igual a 1, a mensagem "Primeiro Lugar" é impressa no console.
* Se **colocacao** for igual a 2, a mensagem "Segundo Lugar" é impressa no console.
* Se **colocacao** for igual a 3, a mensagem "Terceiro Lugar" é impressa no console.
* Se **colocacao** for igual a 4, 5 ou 6, a mensagem "Prêmio de participação" é impressa no console.
* Se **colocacao** não coincidir com nenhum dos casos anteriores, a cláusula **default** é executada, e a mensagem "Não subiu ao pódio" é impressa no console.

Dado que **colocacao** é igual a 7, o código executará a cláusula **default**, imprimindo a mensagem "Não subiu ao pódio" no console.

**LOOP FOR**

console.log("Inicio do programa")

for (let i = 1; i <= 100; i++) {

    if(i%2==0) {

        console.log(i + " é par")

    } else {

        console.log(i + " é impar")

    }

}

console.log("Fim do programa")

O loop **for** itera de 1 a 100. Para cada valor de **i** no intervalo especificado, verifica se **i** é par ou ímpar usando a condição **if(i % 2 == 0)**. Se **i** for divisível por 2 (ou seja, o resto da divisão por 2 é 0), então é par e a mensagem é "i é par" é impressa no console. Caso contrário, a mensagem "i é ímpar" é impressa. Este processo se repete para cada valor de **i** no intervalo.

**LOOP FOR IN, FOR OF**

let num = [10, 20, 30, 40, 50]

for (let i = 0; i < num.length; i++) {

    console.log(num[i])

}

for (n in num) { //Imprime as posições

    console.log(n)

}

for (n of num) {

    console.log(n) //Imprime os elementos

}

Uma variável chamada **num** é declarada e inicializada com um array contendo os elementos **[10, 20, 30, 40, 50]**.

O loop **for** utiliza um índice (**i**) para percorrer cada elemento do array **num**. O loop começa com **i** igual a 0 e continua enquanto **i** for menor que o comprimento do array (**num.length**). A cada iteração, o valor do elemento no índice **i** é impresso no console.

O loop **for...in** itera sobre as posições do array **num** (os índices). O loop atribui cada índice a variável **n**, e o valor do índice é impresso no console. Portanto, este loop imprime os índices do array.

O loop **for...of** itera sobre os elementos do array **num**. A cada iteração, o valor do elemento é atribuído à variável **n** e impresso no console. Portanto, este loop imprime os elementos do array.

**WHILE**

let n = 10, fat = 1

while (n >= 1) {

    fat \*= n

    n--

}

console.log(fat)

Duas variáveis são declaradas e inicializadas. **n** é o número para o qual estamos calculando o fatorial (neste caso, 10), e **fat** é a variável que armazenará o resultado do fatorial, inicializada com 1.

O loop **while** continua executando enquanto **n** for maior ou igual a 1. Dentro do loop, o valor de **fat** é multiplicado pelo valor atual de **n**, e em seguida, **n** é decrementado em 1. Isso é feito repetidamente até que **n** seja menor que 1.

O loop está calculando o fatorial de **n**. A cada iteração, o valor de **n** é multiplicado pelo valor acumulado em **fat**. Isso continua até **n** atingir 1, resultando no fatorial de 10.

**WHILE DO**

let n = 10

do {

    console.log("Barcelona")

    n++

} while (n < 10) {

    console.log("Fim do programa")

}

Uma variável **n** é declarada e inicializada com o valor 10.

A estrutura **do...while**, executa o bloco de código dentro do **do** pelo menos uma vez, mesmo que a condição no **while** seja falsa desde o início.

**BREAK, CONTINUE**

let n = 0, max = 1000, pares = 0

while (n < max) {

    console.log("X - " + n)

    if (n > 10) {

        break

    }

    n++

}

console.log("Fim do programa")

for (let i = 0; i < max; i++) {

    if (i%2 == 0) {

        continue

    }

    pares++

}

console.log("quantidade de pares: " + pares)

console.log("Fim do programa")

Três variáveis são declaradas e inicializadas. **n** é um contador inicializado com 0, **max** é o valor máximo (1000), e **pares** é um contador inicializado com 0.

O loop **while** executa enquanto **n** for menor que **max**. A cada iteração, ele imprime no console "X - " seguido do valor de **n**. Se **n** for maior que 10, o loop é interrompido com a instrução **break**. Isso significa que o loop será encerrado assim que **n** ultrapassar 10.

O loop **for** itera de 0 até **max - 1**. Se o valor de **i** for par (**i % 2 == 0**), a instrução **continue** é usada para pular para a próxima iteração sem executar o código abaixo. Portanto, **pares** é incrementado apenas para valores ímpares.

**FUNÇÃO**

function nome() { //Declaração da função

    let n1 = 2, n2 = 10, soma = n1 + n2

    console.log(soma)

}

for (let i = 0; i < 10; i++) {

    nome() //Chamada da função

}

Uma função chamada **nome** é declarada. Dentro da função, duas variáveis locais **n1** e **n2** são inicializadas com os valores 2 e 10, respectivamente. Em seguida, uma terceira variável chamada **soma** é criada e armazena a soma de **n1** e **n2**. Finalmente, o resultado da soma é impresso no console.

O loop **for** é utilizado para chamar a função **nome** 10 vezes. O loop começa com **i** igual a 0 e continua enquanto **i** for menor que 10. A cada iteração do loop, a função **nome** é chamada, resultando na impressão de "12" (a soma de **n1** e **n2**) no console.

**FUNÇÃO COM RETORNO**

function canal() {

    let n1 = 10, n2 = 2, res = n1 \* n2

    return res

}

let num = canal()

console.log(num)

console.log(canal())

Uma função chamada **canal** é definida. Dentro da função, duas variáveis locais **n1** e **n2** são inicializadas com os valores 10 e 2, respectivamente. Uma terceira variável chamada **res** armazena o resultado da multiplicação de **n1** por **n2**. A função utiliza a palavra-chave **return** para retornar o valor calculado.

A função **canal** é chamada e o resultado retornado (o valor da multiplicação) é armazenado na variável **num**.

O valor armazenado em **num** é impresso no console. Isso resultará na impressão de "20", que é o resultado da multiplicação de 10 por 2.

A função **canal** é chamada diretamente no argumento do **console.log**. Isso imprimirá diretamente o resultado da função no console, novamente resultando na impressão de "20".

**FUNÇÃO PARAMETRIZADA**

function soma(n1, n2) {

    console.log(n1 + n2)

}

soma(10, 5)

Uma função chamada **soma** é definida. Esta função aceita dois parâmetros, **n1** e **n2**. Dentro do corpo da função, a soma de **n1** e **n2** é calculada e o resultado é impresso no console usando **console.log**.

A função **soma** é chamada com os argumentos 10 e 5. Isso significa que dentro da função, **n1** será 10 e **n2** será 5. A soma de 10 e 5 (10 + 5) é calculada e o resultado, que é 15, é impresso no console.

**PARÂMETRO REST EM FUNÇÃO**

function soma(...valores) {

    let tam = valores.length, res = 0

    for ( let v of valores) {

        res += v

    }

    return res

}

console.log(soma(10, 5))

Uma função chamada **soma** é definida com o uso do operador de propagação/rest **...valores**. Isso permite que a função aceite um número variável de argumentos e os agrupe em um array chamado **valores**. Dentro do corpo da função, um loop **for...of** é usado para iterar sobre os valores do array e realizar a soma deles. O resultado da soma é armazenado na variável **res**, que é, então, retornado pela função.

A função **soma** é chamada com os argumentos 10 e 5. Esses valores são passados para a função e são tratados como elementos do array **valores**. A função realiza a soma desses dois valores e retorna o resultado. O **console.log** imprime esse resultado no console.

**FUNÇÃO ANÔNIMA**

const f = new Function("v1", "v2", "return v1 + v2") //Função Construtor Anônima

console.log(f(10, 5))

Uma função é criada dinamicamente utilizando a função construtora **Function**. Os parâmetros da função são especificados como strings dentro da chamada ao construtor. O último argumento da chamada é o corpo da função, que é **"return v1 + v2"**. Isso cria uma função anônima que recebe dois parâmetros (**v1** e **v2**) e retorna a soma deles.

A função **f** é chamada com os argumentos 10 e 5. A função interna, criada dinamicamente, é executada e retorna o resultado da soma, que é 15. Esse resultado é então impresso no console usando **console.log**.

**ARROW FUNCTION**

const soma = (v1 , v2) => v1 + v2

console.log(soma(10, 5))

A função **soma** é definida usando a sintaxe de função de seta. Esta sintaxe é uma forma mais concisa de definir funções em JavaScript. A função recebe dois parâmetros, **v1** e **v2**, e retorna a soma deles. A função é atribuída à constante **soma**.

A função **soma** é chamada com os argumentos 10 e 5. A função retorna a soma desses dois valores (10 + 5), que é 15. O **console.log** imprime esse resultado no console.

**FUNÇÃO DENTRO DE FUNÇÃO**

const soma = (...valores) => {

    const somar = val => {

        let res = 0

        for (v of val) {

            res += v

        }

        return res

    }

    return somar(valores)

}

console.log(soma(10, 5, 15))

valor = [10, 5, 15]

console.log(soma(...valor))

A função **soma** é definida usando a sintaxe de função de seta e o operador de propagação (**...valores**) para aceitar um número variável de argumentos. Dentro da função **soma**, há outra função chamada **somar**, que recebe um array de valores (**val**) e realiza a soma deles usando um loop **for...of**. A função **somar** é, então, chamada com os valores recebidos pela função **soma**, e o resultado é retornado.

A função **soma** é chamada com os argumentos 10, 5 e 15. Internamente, a função **somar** é chamada com esses valores, resultando na soma (10 + 5 + 15), que é 30. O **console.log** imprime esse resultado no console.

Um array chamado **valor** é criado com os elementos 10, 5 e 15. A função **soma** é então chamada novamente, desta vez usando o operador de propagação (**...valor**) para passar os elementos do array como argumentos individuais. Isso resulta na chamada **soma(10, 5, 15)**, que novamente retorna a soma (10 + 5 + 15), que é 30. O **console.log** imprime esse resultado no console.

**FUNÇÃO GERADORA**

function\* perguntas() {

    const nome = yield 'Qual seu nome?'

    const esporte = yield 'Qual seu esporte favorito?'

    return 'Seu nome é ' + nome + ', seu esporte favorito é ' + esporte

}

const itp = perguntas()

console.log(itp.next().value)

console.log(itp.next('Rafael').value)

console.log(itp.next('Futebol').value)

Uma função geradora chamada **perguntas**. Dentro dessa função, há duas expressões **yield** que pausam a execução da função e retornam os valores das perguntas. Quando a função é retomada, os valores fornecidos na próxima chamada **next** serão atribuídos às variáveis **nome** e **esporte**, respectivamente. A função retorna uma mensagem combinando o nome e o esporte fornecidos.

A função geradora **perguntas** é inicializada, criando um iterador chamado **itp**.

A primeira chamada de **itp.next()** inicia a execução da função geradora até o primeiro **yield**. Neste caso, ela imprime no console a pergunta "Qual seu nome?" e pausa a execução até que a próxima chamada **next** seja feita. O valor retornado é a string "Qual seu nome?".

A segunda chamada **itp.next('Rafael')** retoma a execução da função geradora e fornece o valor 'Rafael' para a variável **nome**. A função então imprime no console a pergunta "Qual seu esporte favorito?" e pausa novamente. O valor retornado é a string "Qual seu esporte favorito?".

A terceira chamada **itp.next('Futebol')** retoma a execução, fornecendo o valor 'Futebol' para a variável **esporte**. A função agora executa até o final e retorna a mensagem combinando nome e esporte. O valor retornado é a string "Seu nome é Rafael, seu esporte favorito é Futebol".

**METODO MAP**

const cursos = ['HTML', 'CSS', 'Javascript', 'PHP', 'React']

cursos.map((el, i) => {

    console.log("Curso: " + el + " - Posição do curso: " + i)

})

Um array chamado **cursos** é definido, contendo cinco elementos que representam nomes de cursos.

O método **map** é chamado no array **cursos**. O método **map** itera sobre cada elemento do array e executa uma função de callback para cada elemento. A função de callback recebe dois parâmetros: **el** (elemento atual) e **i** (índice do elemento).

Dentro da função de callback, o **console.log** é utilizado para imprimir uma mensagem para cada curso, incluindo o nome do curso (**el**) e a posição do curso no array (**i**). A mensagem é concatenada usando strings e os valores de **el** e **i**.

**OPERADOR THIS**

function aluno(nome, nota) {

    this.nome = nome

    this.nota = nota

    this.dados\_arrow = function() {

        setTimeout(() => {

            console.log(this.nome)

            console.log(this.nota)

        }, 2000)

    }

}

const al1 = new aluno("Rafael", 100)

al1.dados\_arrow()

A função construtora **aluno** é definida para criar objetos aluno. A função aceita dois parâmetros, **nome** e **nota**, e atribui esses valores aos membros **nome** e **nota** do objeto criado. Além disso, a função construtora possui um método **dados\_arrow**, que utiliza uma função de seta dentro do **setTimeout**. A função de seta preserva o valor de **this**, então mesmo que a função seja chamada em um contexto diferente, ela ainda referencia o objeto aluno atual.

A função construtora é utilizada para criar um objeto aluno chamado **al1** com nome "Rafael" e nota 100.

O método **dados\_arrow** do objeto **al1** é chamado. Este método utiliza **setTimeout** para adicionar um atraso de 2000 milissegundos antes de imprimir o nome e a nota do aluno no console. A função de seta dentro do **setTimeout** permite que **this** continue referenciando o objeto aluno (**al1**), mesmo após o atraso.

**DOM (getElementsById)**

const dc1 = document.getElementById("c1")

const dc2 = document.getElementById("c2")

const dc3 = document.getElementById("c3")

const dc4 = document.getElementById("c4")

const dc5 = document.getElementById("c5")

const dc6 = document.getElementById("c6")

const arrayElement = [dc1, dc2, dc3, dc4, dc5, dc6]

arrayElement.map((e) => {

    e.innerHTML = "Barcelona"

    console.log(e)

})

Seis variáveis ​​( **dc1** a **dc6**) são declaradas e utilizadas para armazenar referências a elementos HTML encontrados no documento usando a função **getElementById**. Cada elemento é identificado pelo seu ID correspondente ("c1" a "c6").

Uma array chamada **arrayElement** é criada, contendo as referências dos elementos HTML armazenados nas variáveis dc1​​a dc6.

O método **map** é utilizado em array **arrayElement** para iterar sobre cada elemento do array. Para cada elemento e, o conteúdo interno ( **innerHTML**) é definido como "Barcelona". Além disso, a função **console.log** é utilizada para imprimir cada elemento no console.

Portanto, para cada elemento HTML identificado pelos IDs "c1" a "c6", o conteúdo interno é definido como "Barcelona", e os elementos são impressos no console. Essa abordagem é comum ao trabalho com manipulação do DOM em JavaScript.

**DOM (getElementsByTagName)**

const colecaoHTML = [...document.getElementsByTagName("div")]

console.log(colecaoHTML)

colecaoHTML.map((e) => {

    console.log(e)

})

O método **getElementsByTagName()** seleciona todos os elementos com a tag **div** no documento HTML. O resultado é armazenado em uma lista chamada **colecaoHTML**.

A função **map()** percorre cada elemento da lista **colecaoHTML**. Para cada elemento, a função imprime o conteúdo do elemento no console do navegador.

**DOM (getElementsByClassName)**

const cursosTodos = [...document.getElementsByClassName("curso")]

const cursosC1 = [...document.getElementsByClassName("c1")]

const cursosC2 = [...document.getElementsByClassName("c2")]

const cursoEspecial = document.getElementsByClassName("curso")[6]

console.log(cursosTodos)

console.log(cursosC1)

console.log(cursosC2)

console.log(cursoEspecial)

cursosC2.map((el) => {

    el.classList.add("destaque")

})

O método **getElementsByClassName()** seleciona todos os elementos com a classe "curso" no documento HTML. O resultado é armazenado em uma lista chamada **cursosTodos**.

Seleciona especificamente o sétimo elemento (índice 6) que possui a classe "curso". O resultado é armazenado na variável **cursoEspecial**.

A função **map()** percorre cada elemento da lista **cursosC2**. Para cada elemento, a função adiciona a classe "destaque".