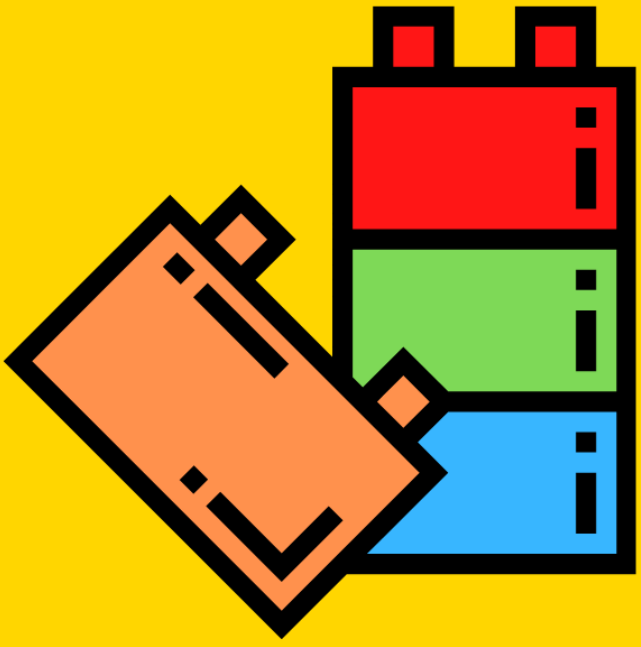
7 EXERCÍCIOS RESOLVIDOS

de Lógica de Programação com JavaScript



CodDev

Paulo Bezerra

SOBRE O AUTOR

Paulo Bezerra é programador a 12 anos e já trabalhou com diversas tecnologias.

Funcionário público federal desde 2012, sentindo a necessidade de compartilhar seus conhecimentos em programação criou o CodDev TV no YouTube onde compartilha conteúdos semanalmente.

Ministra cursos online que vão do básico ao avançado no desenvolvimento de software utilizando JavaScript entre outras tecnologias.

INTRODUÇÃO

Me chamo Paulo, sou programador desde 2008, já trabalho com várias tecnologias e linguagens de programação durante todos estes anos desenvolvendo software. Atualmente sou funcionário público e trabalho no setor de desenvolvimento de software de uma universidade federal.

Talvez você esteja achando estranho um ebook com páginas pretas e fonte mono-espaçada, mas acostume-se, mesmo trabalhando com JavaScript no frontend, se você deseja programar profissionalmente a tela preta e o Courier New[1] são os seus melhores amigos. Eles estão no terminal do Linux, nas IDEs e editores de código.

Devo confessar que a decisão de deixar tudo preto foi um acidente. Quando copiei e colei o primeiro código aqui, ele veio com fundo preto do Visual Studio Code[2], pensei então em pintar tudo de preto. Seguramente estou orgulhoso dessa escolha, pois ela ressalta o código fonte e parece que estou em um editor de código o tempo todo.

Agora vamos falar do conteúdo desse ebook que é provavelmente a tecnologia mais legal que já trabalhei e que oferece um leque gigantesco de opções: **JavaScript**. Muitos que criticam e acham a linguagem meio bagunçada é porque provavelmente não entenderam a brincadeira ou são presos de mais as suas linguagens preferidas. Eu mesmo já tive meu preconceito com JavaScript, mas foi em 2012 quando comecei a conhecer a linguagem mais a fundo e as ferramentas fantásticas que ela fornece que nunca mais parei de estudar e acompanhar a evolução dessa tecnologia.

Se já programa em outras linguagens e está chegando por aqui, peço um favor, se desarme de todo preconceito e besteiras que você já ouviu sobre JavaScript. Tenha a mentalidade de uma criança, com uma chave de fenda na mão, tentando entender um brinquedo novo que acabou de ganhar no natal (talvez não seja toda criança, mas eu fazia isso...).

Resolvi escrever este ebook baseado em exercícios resolvidos e através de exemplos práticos trazer conceitos básicos da linguagem. Não se engane! Ler, ver vídeos, navegar em códigos no Github, etc... é muito importante, mas programação você só aprende na prática! Então após ler um capítulo aqui, busque entender cada linha de código e depois **faça você mesmo**! **PRATIQUE!**

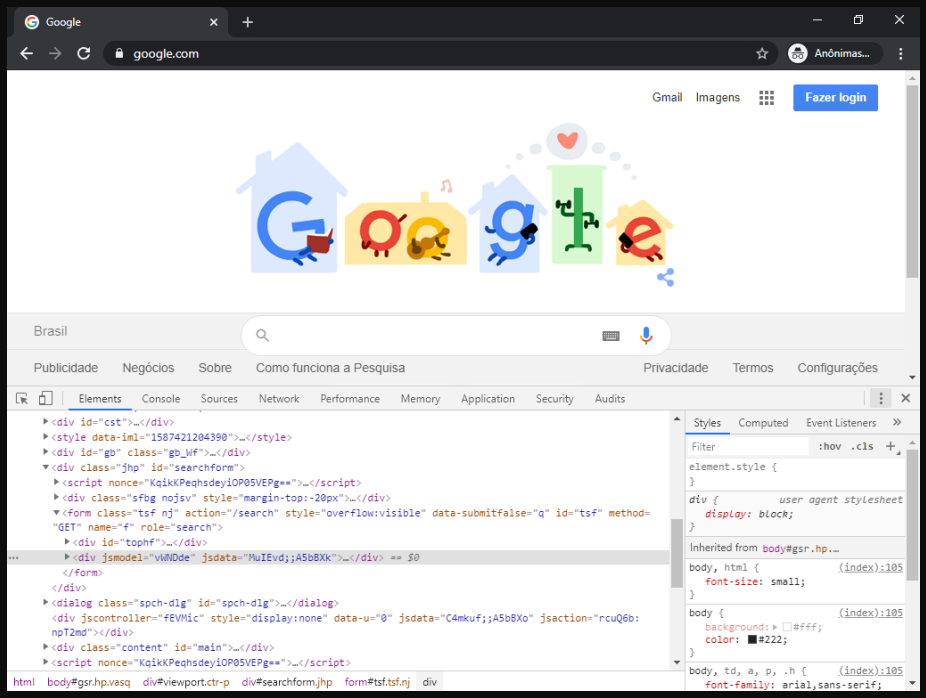
---

[1] Courier New é o tipo de fonte que estou utilizando aqui, é uma fonte mono-espaçada e você vai ver muito dela por ai em ferramentas de desenvolvimento.
[2] Editor de código que usaremos para editar os códigos contidos neste ebook.

AMBIENTE DE DESENVOLVIMENTO

Primeira coisa que você vai precisar é de um navegador. Eu sou um usuário muito antigo do Google Chrome (desde quando ele foi lançado e era uma verdadeira carroça, muito pior que nos dias atuais), aconselho você a utilizar esse navegador, primeiro por ser desenvolvido pela Google e estar sempre à frente da concorrência em relação a implementação de novidades do ECMAScript[3]. Segundo motivo é pelas ferramentas de desenvolvedor que são completas e fantásticas, permitindo testar seu código no navegador com muitos recursos e facilidades. Terceiro seria pela vasta variedade de plugins que podem te auxiliar imensamente nas suas tarefas, fora a possibilidade de desenvolver seus próprios plugins de forma muito simples.

Então sua primeira tarefa, caso não tenha instalado, é instalar o Google Chrome. Depois de instalado, vamos visitar a ferramenta de desenvolvedor, basta clicar com o botão direito do mouse em qualquer parte da página que está aberta e após isso clicar em "Inspecionar" ou teclar Ctrl+Shift+I. Você verá uma tela como da imagem a seguir:

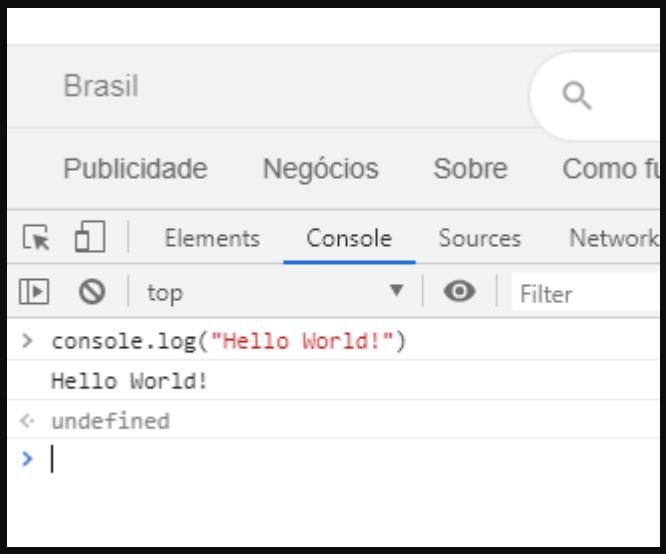


Você tem várias abas com diferentes recursos para testar suas páginas, não tenha medo de mexer e testar, você aprende muito fuçando. Mas o que precisamos ver agora é a aba "Console". Clique nela, digite a linha a seguir e tecle Enter:

```
console.log("Hello World!")
```

[3] Se você não sabe o que é o ECMAScript, pense nele como um rascunho de tudo aquilo que deve ser implementado pelos navegadores na interpretação do JavaScript.

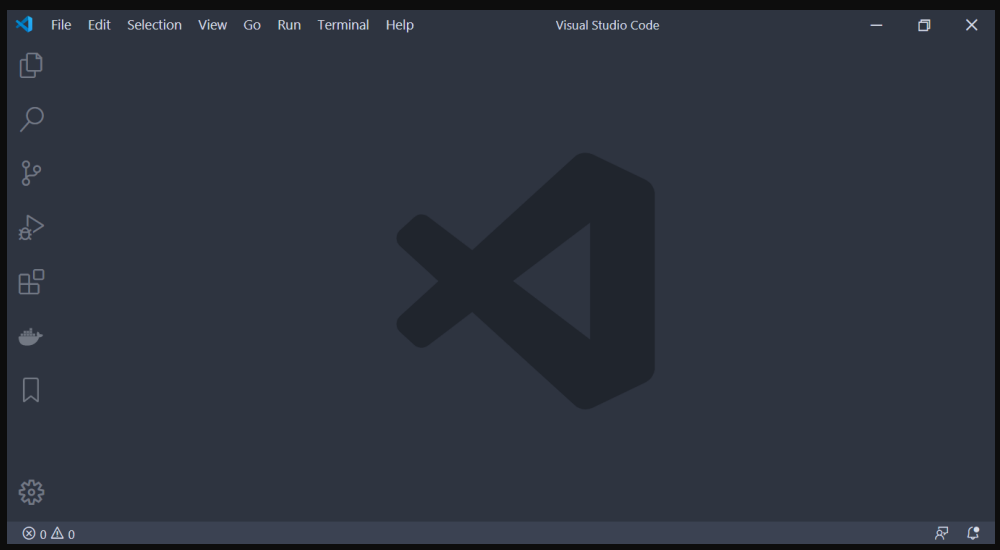
Se você fez tudo certo verá algo parecido com a imagem a seguir:



Parabéns, você acaba de escrever o seu primeiro código utilizando a linguagem JavaScript. O mais legal é que, se já tinha o Google Chrome instalado, fez isso sem precisar instalar nenhuma ferramenta pesada de programação no seu computador. Pelo console, você pode testar todos os conceitos do JavaScript enquanto estuda, o mais legal é esta opção está sempre a mão enquanto você estiver utilizando o navegador.

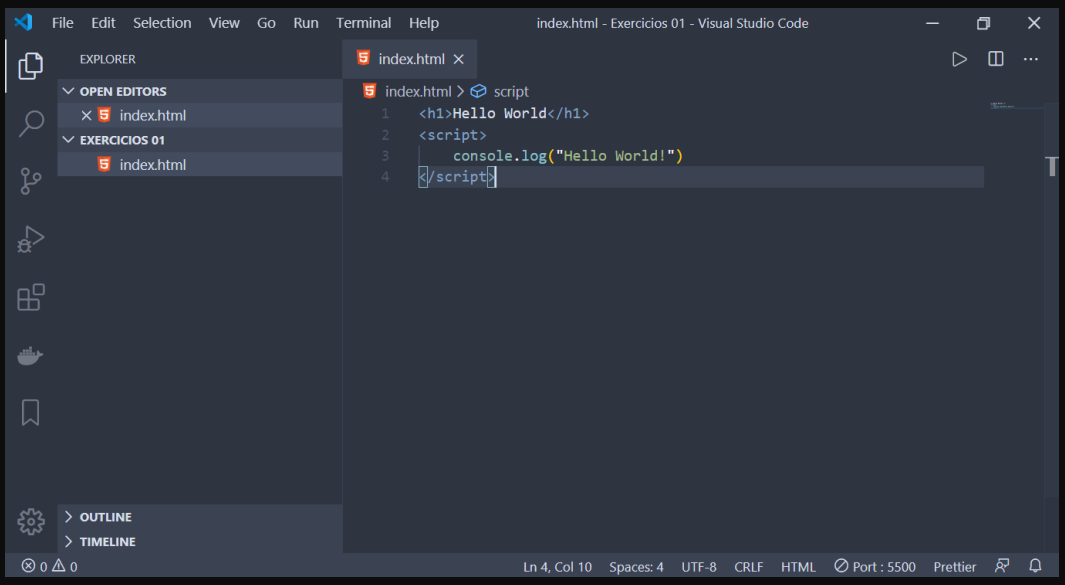
Obviamente que você não vai criar aplicações pelo console do Google Chrome, mas usamos bastante ele para realizar testes no nosso programa utilizando o comando console.log() que vimos anteriormente. Para criação de sites e sistemas completos utilizamos editores de código como é o caso do Visual Studio Code (VSCode) que já citei anteriormente. Procure por VSCode no Google e instale o editor.

O VSCode é desenvolvido pela Microsoft e possui um conceito muito interessante que são as extensões que podem ser instaladas. Como a comunidade de programadores do mundo todo abraçou esta ferramenta, existe extensões para praticamente tudo que você imaginar. Na imagem a seguir você pode ver a tela do VSCode.



É interessante que você clique no botão e extensões do VSCode e busque por "Live Server" e instale. Com esta extensão o VSCode passa a se comportar como um servidor de páginas web e deixa mais simples testar nossos códigos no navegador.

Após fazer a instalação do Live Server, clique em ***File >> Open Folder...*** Sugiro que você crie uma nova pasta vazia e abra ela. Agora clique em ***File >> New File*** para criar nosso primeiro arquivo, de o nome de "index.html" para o arquivo e digite o conteúdo do arquivo como na imagem a seguir:



A seguir o conteúdo do arquivo *Index.html:*

```
<h1>Hello World</h1>
<script>
    console.log("Hello World!")
</script>
```

Agora clique com o botão direito do mouse dobre o arquivo *index.html* no lado esquerdo da tela (no Explorer) e em seguida clique em ***Open With Live Server*** e espere ser aberta uma aba do navegador com a sua página recém-criada. No navegador veja nas ferramentas de desenvolvimento a aba console mostrando a mensagem "Hello World!".

Pronto, se chegou até aqui com sucesso você já entendeu o processo para executar todos os códigos que veremos a seguir. Lembre-se sempre que apenas ler não basta, para comprovar que está entendendo você deve ir ao campo de batalha (VSCode) e fazer o código rodar. Bons estudos!

EXERCÍCIO 01: TROCANDO VARIÁVEIS

Escreva um programa que contenha duas variáveis A e B. O programa deve trocar o valor de A pelo valor de B e vice-versa. Ao final deve escrever no console as variáveis A e B.

Começamos com um clássico! Se alguma vez você fez exercícios de lógica de programação e/ou algoritmos certamente já viu esse aqui e talvez deseje já ir para o próximo exercício. Se estiver iniciando do zero, não deve ter entendido absolutamente nada que o exercício está pedindo. Pois bem, primeiramente temos que entender estes conceitos, então vamos lá!

O que são variáveis?

Variáveis são espaços de memória que utilizamos para armazenas os dados enquanto o programa é executado. Sendo que o programa, no nosso caso aqui, é o JavaScript de uma página web rodando dentro de um navegador no nosso caso o Google Chrome.

Como declaro uma variável?

No JavaScript você utiliza as palavras **var**, **let** ou **const**, para que o interpretador da linguagem entenda que você está declarando uma variável.

Por padrão buscamos sempre declarar variáveis imutáveis, ou seja, que não tem seu valor alterado, portanto utilizamos mais o **const**. Caso precise de uma variável mutável utilize o **let**. Evite o **var**, ele tem problemas de contexto que podem te dar muita dor de cabeça.

JavaScript não possui tipagem de dados estática, isso quer dizer que não precisa dizer que determinada variável é um texto, a variável passa a ser do tipo texto (String) quando atribuímos um texto a ela. Exemplo de declaração das variáveis do exercício anterior:

```
//Variável A
let a = 10

//Variável B
let b = 20
```

Sabemos que as variáveis terão de ter seus valores trocados, então são mutáveis e por isso declaramos como **let**. Sabemos que são do tipo Number, pois atribuímos valores numéricos inteiros para essas variáveis (10 e 20).

Apenas para esclarecimento, as linhas precedidas por duas barras (//) são comentários e são ignoradas na execução do programa, serve apenas para que o programador explique/documente o código.

Como trocar o valor de uma variável?

Perceba como estamos fazendo para atribuir o valor 10 a variável A. A mesma coisa poderia ser feita se no lugar do 10 tivéssemos a variável B:

```
//Variável A recebe o valor da variável B
a = b
```

Observe que não utilizamos mais a palavra let, pois nesse caso A já é uma variável declarada. Perceba também que ao atribuir o valor de B em A, o valor de A é perdido. Portanto esse não é um caminho para que possamos resolver este exercício, precisamos passar o valor de B para A, mas também precisamos passar o valor de A para B.

Então o que fazer?

Agora que sabemos declarar variáveis e atribuir novos valores (mutação), precisamos pensar em como trocar o valor de A pelo valor de B, e vice-versa, para finalmente poder utilizar console.log e imprimir o resultado do programa.

O enunciado disse para declarar A e B e fazer a troca, ele não nos proibia de ter uma variável auxiliar para nos ajudar nessa troca, e com isso temos a solução.

Imagine que as variáveis são caixas e os valores objetos guardados nas caixas, para que não fique nenhum objeto fora da caixa enquanto é feita a troca, precisamos de uma terceira caixa para conseguir realizar toda essa operação. Com isso chegamos a este código:

```
// Declaramos nossa variável AUX (como uma constante, já que não será mutável)
const aux = a
//Variável A recebe o valor de B
a = b
//Variável B recebe o valor de AUX
b = aux
```

Agora que já fizemos a troca podemos escrever o resultado no console. Para isso vamos juntar os valores de A e B com texto. Essa junção de variáveis com texto chamamos de ***concatenação de strings***.

```
//Mostrando o resultado
console.log("Valor de A: " + a + " - Valor de B: " + b);
```

A seguir o código completo do arquivo exercicio1.html:

```
<script>
    //Variavel A
    let a = 10

    //Variável B
    let b = 20

    //Declaramos a variável AUX (como uma constante já que não será mutável)
    const aux = a
```

```
    //Variável A recebe o valor de B
    a = b
    //Variável B recebe o valor de AUX
    b = aux

    //Monstrando o resoltado
    console.log("Valor de A: " + a + " - Valor de B: " + b);
</script>
```

Agora é hora de digitar esse código no VSCode, rodar ele com o Live Server e ver o resultado da execução na aba Console das Ferramentas de Desenvolvedor do Google Chrome. Depois que conseguir que funcione, passe para o próximo exercício.

EXERCÍCIO 2: CALCULANDO O IMC

O Índice de Massa Corporal (IMC) serve para avaliar o peso do indivíduo em relação a sua altura e assim indicar se a o peso está acima, abaixo ou ideal. Sabendo que a fórmula para o cálculo do IMC é a divisão do PESO pelo quadrado da ALTURA. Escreva um programa que leia o PESO e a ALTURA e apresente o resultado conforme a tabela a seguir:

Classificação	IMC	O que pode acontecer
Muito abaixo do peso	16 a 16,9 kg/m²	Queda de cabelo, infertilidade, ausência menstrual
Abaixo do peso	17 a 18,4 kg/m²	Fadiga, stress, ansiedade
Peso normal	**18,5 a 24,9 kg/m²**	**Menor risco de doenças cardíacas e vasculares**
Acima do peso	25 a 29,9 kg/m²	Fadiga, má circulação, varizes
Obesidade Grau I	30 a 34,9 kg/m2	Diabetes, angina, infarto, aterosclerose
Obesidade Grau II	35 a 40 kg/m2	Apneia do sono, falta de ar
Obesidade Grau III	maior que	Refluxo, dificuldade para se mover, escaras, diabetes, infarto, AVC

Perceba que para resolver esse problema teremos duas variáveis: PESO e ALTURA, isso já sabemos fazer. Mas agora teremos que ler essas informações do usuário, ou seja, o usuário irá informar PESO e ALTURA para que nosso programa faça o cálculo e apresente o resultado. Então vamos por partes, tem algumas coisas novas que precisamos aprender aqui.

Primeiramente, como leio uma entrada do usuário em uma página web?

Temos a possibilidade de utilizar um comando fornecido pelo navegador: ***window.prompt***. Na verdade "comando" não é o termo mais correto, no JavaScript chamamos ***prompt*** de função. Perceba que ***prompt*** está dentro de ***window***, por isso precisamos acessar ***window*** para acessar a função ***prompt***.

O ***window*** é algo que chamamos de ***object***, e no caso do ***object window*** temos um object global, que pode ser acessado em qualquer parte do nosso código. Então para lermos PESO e ALTURA, podemos fazer como a seguir:

```
// Lendo peso do usuário
let peso = window.prompt('Informe o peso:')
// Lendo altura do usuário
let altura = window.prompt('Informe a altura:')
```

E agora, como fazemos o cálculo?

Para calcular é muito simples, precisamos apenas lembrar da fórmula e saber que ***operadores aritméticos*** devemos utilizar. Temos no JavaScript quatro operadores aritméticos: +, -, * e / respectivamente representando adição, subtração, multiplicação e divisão. Sabemos que o IMC é o resultado do PESO dividido pelo quadrado da ALTURA, então temos:

```
// IMC é o resultado da divisão do PESO pelo quadrado da ALTURA
const imc = peso / (altura*altura)
```

Legal, mas talvez se você tentar fazer o calculo nesse momento tenha um pequeno contratempo. Dado que o usuário pode informar o PESO e a ALTURA utilizando **vírgula** como separador decimal e o JavaScript espere por separadores decimais como sendo **ponto** não conseguiremos efetuar o cálculo. Outro fato, é que a função **prompt** retorna dados do tipo texto (String) e não numéricos (Float). Precisamos fazer o tratamento e a conversão corretamente dos dados informados.

```
//Trocando vírgula por ponto
peso = peso.replace(',', '.')
altura = altrua.resplace(',', '.')

//Convertendo String em Float
peso = Number.parseFloat(peso)
altura = Number.parseFloat(altura)

// IMC é o resultado da divisão do PESO pelo quadrado da ALTURA
const imc = peso / (altura*altura)
```

Com isso o JavaScript consegue efetuar o cálculo corretamente. Precisamos agora verificar em qual faixa do IMC o nosso usuário se encontra para escrever uma mensagem apresentando o resultado para ele.

Como testar a classificação do IMC?

Para testar se o valor do IMC está em uma determinada classificação vamos precisar entender um pouco sobre ***operadores lógicos e de comparação.*** Podemos comparar o valor de variáveis utilizando os seguinte operadores: igual (===), menor (<), maior (>), menor ou igual (<=), maior ou igual (>=) e desigual (!==).

Em alguns casos precisamos saber se o IMC é maior que X e menor que Y, então precisamos utilizar um operador lógico AND representado no JavaScript por &&. Existe também o operador OR representado por ||.

```
//IMC maior que 15.5 e IMC menor que 24.9
imc >= 18.5 && imc <= 24.9
```

O resultado da operação anterior é um valor booleano, que pode ser verdadeiro (TRUE) ou falso (FALSE), e com isso podemos utilizar uma estrutura condicional ***IF*** para tomada de decisão.

```
// Testa condição
if (CONDICAO) {
    //Excuta isso se a condição for verdadeira
} else {
    //Executa isso se a condição for falsa
}
```

Agora utilizando a comparação e a estrutura condicional podemos retornar à classificação e consequência para nosso usuário:

```
// Testa IMC maior que 16 e menor que 16,9
const classificacao = ''
const consequencia = ''
if (imc >= 16 && imc < 17) {
    classificacao = 'Muito abaixo do peso'
    consequencia = 'Queda de cabelo, infertilidade, ausência menstrual'
} else if (imc >= 17 && imc < 18.5) {
    classificacao = 'Abaixo do peso'
    consequencia = 'Fadiga, stress, ansiedade'
} else if (imc >= 18.5 && imc < 25) {
    classificacao = 'Peso normal'
    consequencia = 'Menor risco de doenças cardíacas e vasculares'
} else if (imc >= 25 && imc < 30) {
    classificacao = 'Acima do peso'
    consequencia = 'Fadiga, má circulação, varizes'
} else if (imc >= 30 && imc < 35) {
    classificacao = 'Obesidade Grau I'
    consequencia = 'Diabetes, angina, infarto, aterosclerose'
} else if (imc >= 35 && imc <= 40) {
    classificacao = 'Obesidade Grau II'
    consequencia = 'Apneia do sono, falta de ar'
} else if (imc > 40) {
    classificacao = 'Obesidade Grau III'
    consequencia = 'Refluxo, dificuldade para se mover, escaras, diabetes, infa
rto, AVC'
}
```

Agora precisamos apresentar o resultado para nosso usuário e podemos fazer isso utilizando mais uma função do objeto global window, usaremos window.alert. Mas para facilitar a concatenação da mensagem, vamos utilizar um ***template string*** que nada mais é do que um texto delimitado por crases (´TEXTO´):

```
//Mostrando resultado utilizando template string
window.alert(`Seu IMC é ${imc} classificado como "${classificacao}" e as conseq
uencias são: ${consequencia}`)
```

A seguir temos o código completo da resolução do exercício 02:

```
<script>
    // Lendo peso do usuário
    let peso = window.prompt('Informe o peso:')

    // Lendo altura do usuário
    let altura = window.prompt('Informe a altura:')
```

```
    //Trocando vírgula por ponto
    peso = peso.replace(',', '.')
    altura = altura.replace(',', '.')

    //Convertendo String em Float
    peso = Number.parseFloat(peso)
    altura = Number.parseFloat(altura)

    // IMC é o resultado da divisão do PESO pelo quadrado da ALTURA
    const imc = peso / (altura * altura)

    // Testa IMC maior que 16 e menor que 16,9
    let classificacao = ''
    let consequencia = ''
    if (imc >= 16 && imc < 17) {
        classificacao = 'Muito abaixo do peso'
        consequencia = 'Queda de cabelo, infertilidade, ausência menstrual'
    } else if (imc >= 17 && imc < 18.5) {
        classificacao = 'Abaixo do peso'
        consequencia = 'Fadiga, stress, ansiedade'
    } else if (imc >= 18.5 && imc < 25) {
        classificacao = 'Peso normal'
        consequencia = 'Menor risco de doenças cardíacas e vasculares'
    } else if (imc >= 25 && imc < 30) {
        classificacao = 'Acima do peso'
        consequencia = 'Fadiga, má circulação, varizes'
    } else if (imc >= 30 && imc < 35) {
        classificacao = 'Obesidade Grau I'
        consequencia = 'Diabetes, angina, infarto, aterosclerose'
    } else if (imc >= 35 && imc <= 40) {
        classificacao = 'Obesidade Grau II'
        consequencia = 'Apneia do sono, falta de ar'
    } else if (imc > 40) {
        classificacao = 'Obesidade Grau III'
        consequencia = 'Refluxo, dificuldade para se mover, escaras, diabetes,
infarto, AVC'
    }

    //Mostrando resultado utilizando template string
    window.alert(`Seu IMC é ${imc} classificado como "${classificacao}" e as co
nsequencias são: ${consequencia}`)
</script>
```

EXERCÍCIO 3: FUNÇÃO PARA CÁLCULO DO IMC

Transforme o código que utilizamos para calcular o IMC em uma função que retorne um objeto contendo o valor do IMC, a classificação e as consequências. Separe essa função em um arquivo calculoIMC.js que possa ser importado como uma dependência de outras aplicações.

Já vimos que o prompt e alert são funções do objeto global window, e agora teremos que criar nossa própria função. Além disso, vamos modularizar esta função, separando-a em um arquivo com extensão .js e permitindo que seja importada em outras aplicações.

Como crio uma função?

Existem algumas formas de criar funções em JavaScript, a primeira e mais simples é criar uma função nomeada como vemos a seguir:

```
// Função nomeada
function calcularIMC(peso, altura) {
    // Bloco de código da função
}
```

O problema da função nomeada é que ela fica exposta para ser chamada em qualquer parte do nosso programa, inclusive pelo console nas Ferramentas de Desenvolvimento do Navegador por usuários que estejam procurando uma forma de hackear nosso site. Então devemos criar funções de forma mais segura.

Uma forma é atribuir a nossa função a uma variável imutável:

```
// Função anônima atribuida a uma variável
const calcularIMC = function(peso, altura) {
    // Bloco de código da função
}
```

E uma terceira forma que podemos utilizar é a de arrow function, onde podemos ter uma sintaxe menos verbosa:

```
// Arrow Function atribuida a uma variável
const calcularIMC = (peso, altura) => {
    // Bloco de código da função
}
```

Como eu retorno o resultado da minha função?

No caso deste exercício, precisamos retornar 3 coisas: IMC, classificação e consequências. Para tanto podemos pegar essas três variáveis e envolve-las em chaves ({}), dessa forma teremos um object sendo retornado contendo os valores processados pela nossa função.

```
//Retornando valores processados pela nossa função
return {imc, classificacao, consequencia };
```

E como podemos exportar nossa função?

Precisamos exportar nossa função para que possamos importá-la em outros pontos do nosso programa:

```
export default calcularIMC
```

Agora que já vimos como criar uma função, como retornar os dados processados e como exportar essa função, temos a seguir o conteúdo completo do arquivo calcularIMC.js:

```
const calcularIMC = (peso, altura) => {
  const imc = peso / (altura * altura);

  let classificacao = "";
  let consequencia = "";
  if (imc >= 16 && imc < 17) {
    classificacao = "Muito abaixo do peso";
    consequencia = "Queda de cabelo, infertilidade, ausência menstrual";
  } else if (imc >= 17 && imc < 18.5) {
    classificacao = "Abaixo do peso";
    consequencia = "Fadiga, stress, ansiedade";
  } else if (imc >= 18.5 && imc < 25) {
    classificacao = "Peso normal";
    consequencia = "Menor risco de doenças cardíacas e vasculares";
  } else if (imc >= 25 && imc < 30) {
    classificacao = "Acima do peso";
    consequencia = "Fadiga, má circulação, varizes";
  } else if (imc >= 30 && imc < 35) {
    classificacao = "Obesidade Grau I";
    consequencia = "Diabetes, angina, infarto, aterosclerose";
  } else if (imc >= 35 && imc <= 40) {
    classificacao = "Obesidade Grau II";
    consequencia = "Apneia do sono, falta de ar";
  } else if (imc > 40) {
    classificacao = "Obesidade Grau III";
    consequencia =
      "Refluxo, dificuldade para se mover, escaras, diabetes, infarto, AVC";
  }

  //Retornando valores processados pela nossa função
  return {imc, classificacao, consequencia };
};

export default calcularIMC;
```

E agora, como chamamos essa função?

Primeiro passo é importar a função no nosso arquivo index.html, isso é bem simples, precisamos acrescentar uma propriedade na tag <script> para dizer que o nosso script é do tipo “module”, e depois importamos a função conforme o código a seguir:

```
<script type="module">
```

```
    import calcularIMC from './calcularIMC.js'
    ...
```

Agora substituímos todo o código que foi movido para dentro da função pela linha a seguir. É importante observar o conceito da ***destructuring*** (desestruturação), que é quando extraímos valores de um objeto para várias variáveis:

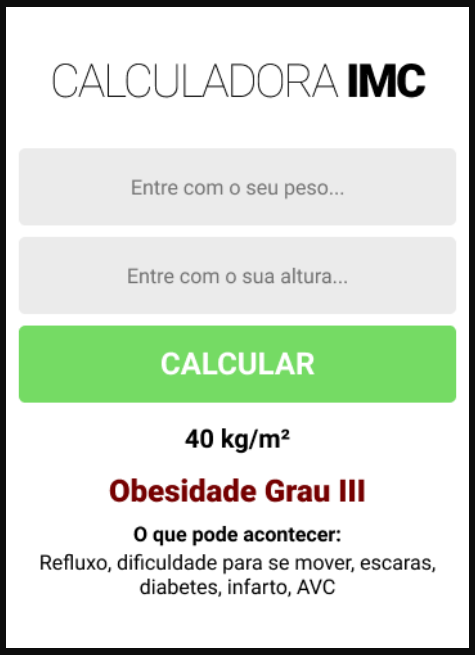
```
const {imc, classificacao, consequencia} = calcularIMC(peso, altura)
```

Com isso nosso programa continua funcionando da mesma forma, sendo que podemos reutilizar a função que calcula o IMC em outros pontos do nosso programa ou até em outros programas. Veja o código final do arquivo index.html:

```
<script type="module">
    import calcularIMC from './calcularIMC.js'
    // Lendo peso do usuário
    let peso = window.prompt('Informe o peso:')

    // Lendo altura do usuário
    let altura = window.prompt('Informe a altura:')
    console.log(peso, altura)

    //Trocando vírgula por ponto
    peso = peso.replace(',', '.')
    altura = altura.replace(',', '.')

    //Convertendo String em Float
    peso = Number.parseFloat(peso)
    altura = Number.parseFloat(altura)

    const {imc, classificacao, consequencia} = calcularIMC(peso, altura)

    //Mostrando resultado utilizando template string
    window.alert(`Seu IMC é ${imc} classificado como "${classificacao}"
            e as consequencias são: ${consequencia}`)
</script>
```

EXERCÍCIO 4: FORMULÁRIO IMC

Crie um formulário com o layout a seguir:



Ao clicar no botão CALCULAR, o programa deve pegar os dados dos campos do formulário, verificar se estão preenchidos e se são dados numéricos. Feita a validação dos dados deve chamar a função calcularIMC e apresentar os dados conforme o layout.

Anteriormente havíamos criado apenas a tag <script></script> no nosso arquivo index.html, como precisaremos criar vários elementos dessa página agora, vamos precisar escrever a estrutura básica de um arquivo HTML (Hypertext Markup Language). Graças ao Emmet do VSCode, você pode ir na primeira linha do arquivo e digitar uma exclamação (!) e teclar Ctrl+Espaço, então será sugerida a estrutura básica do HTML que precisamos, tecle Enter para aceitar a sugestão:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculadora IMC</title>
</head>
<body>

</body>
</html>
```

Perceba que aproveitei par alterar o conteúdo da tag título para "Calculadora IMC". Feito isso, vamos colar a tag <script> e todo o seu conteúdo antes do fechamento da tag </body>:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculadora IMC</title>
</head>
<body>


    <script type="module">
        import calcularIMC from './calcularIMC.js'
        // Lendo peso do usuário
        let peso = window.prompt('Informe o peso:')

        // Lendo altura do usuário
        let altura = window.prompt('Informe a altura:')
        console.log(peso, altura)

        //Trocando vírgula por ponto
        peso = peso.replace(',', '.')
        altura = altura.replace(',', '.')

        //Convertendo String em Float
        peso = Number.parseFloat(peso)
        altura = Number.parseFloat(altura)

        const {imc, classificacao, consequencia} = calcularIMC(peso, altura)

        //Mostrando resultado utilizando template string
        window.alert(`Seu IMC é ${imc} classificado como "${classificacao}" e a
s consequencias são: ${consequencia}`)
    </script>
</body>
</html>
```

Logo após a abertura da tag <body>, vamos criar todos os elementos necessários para compor o layout solicitado no enunciado do exercício.

```
<body>
  <main>

        <h1>Calculadora <strong>IMC</strong></h1>
        <form>
            <input type="number" step="0.01" name="peso" placeholder="Entre com
 o peso..."/>
            <input type="number" step="0.01" name="altura" placeholder="Entre c
om a altura..."/>
            <button type="submit">Calcular</button>
            <div id="resultado">
                <h3>40kg/m²</h3>
                <h2>Obesidade Grau III</h2>
                <p><strong>O que pode acontecer:</strong></p>
```

```
                <p>Refluxo, dificuldade para se mover, escaras, diabetes, infarto, AVC</p>
            </div>
        </form>
    </main>
```

Perceba que criamos toda nossa estrutura de formulário dentro da tag <main>, e deixamos uma marcada com id "resultado" que é onde colocaremos dinamicamente os dados que vamos obter ao executar nossa função calcularIMC. Vamos nos preocupar agora com a parte estética da nossa aplicação, adicionando um arquivo com o CSS(Cascade Style Sheet) para formatar adequadamente nosso layout.

Crie um arquivo com o nome de style.css e acrescente o link para esse arquivo no <head> de index.html:

```
    <title>Calculadora IMC</title>
    <link rel="stylesheet" href="style.css">
</head>
```

A seguir o conteúdo do arquivo style.css:

```
/*
IMPORTAÇÃO DE FONTE
Fonte importada de fonts.google.com
*/
@import url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap');

/*
RESET CSS
A formatção a seguir é aplicada a todos os elementos da página
*/
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/*
Ajustando página ao tamanho da tela
*/
html,
body {
    height: 100vh;
}

/*
Formatando body para que conteudo da página fique centralizado verticalmente
 */
body {
    font-family: Roboto;
```

```
    display: flex;
    flex-direction: column;
    justify-content: space-around;
}

/*
Conteudo principal foca centralizado verticalmente
*/
main {
    margin: 0 auto;
    min-height: 500px;
    width: 100%;
    max-width: 360px;
    padding: 10px;
    text-align: center;
    display: flex;
    flex-direction: column;
    font-size: 14px;
}

h1 {
    font-weight: 100;
    text-transform: uppercase;
    font-size: 36px;
    margin-bottom: 20px;
    letter-spacing: -0.05em;
}

h1 strong {
    font-weight: 900;
}

input {
    display: block;
    width: 100%;
    font-size: 16px;
    padding: 15px 10px;
    margin: 10px 0;
    border-radius: 5px;
    border: none;
    background-color: #EBEBEB;
    text-align: center;
}

button {
    display: block;
    width: 100%;
    font-size: 16px;
    padding: 15px 10px;
    margin: 10px 0;
    border-radius: 5px;
    border: none;
    background-color: #75DB64;
```

```
    text-transform: uppercase;
    color: #FFFFFF;
    font-size: 24px;
}

h3 {
    font-weight: bold;
    font-size: 18px;
    margin-top: 20px;
}

h2 {
    color: #750000;
    font-size: 24px;
    margin: 10px 0;
    font-weight: 900;
}


#resultado {
    margin-bottom: 20px;
}
```

Precisamos alterar nosso programa para pegar os dados do formulário, calcular e interpolar o resultado no HTML dentro da , a seguir faremos as alterações necessárias na tag <script> da index.html para que nosso programa funcionem conforme o solicitado pelo exercício:

```
<script type="module">
    import calcularIMC from './calcularIMC.js'

    //Recuperando o formulario
    const form = document.forms['formIMC']

    //nosso código ficará escutando o evento submit
    //este evento ocorre ao clicar no button do type="submit"
    form.addEventListener('submit', (event) => {
        //previne comportamneto padrão do navegador ao submeter o form
        event.preventDefault()

        //lendo e o peso e altura do input
        let peso = form.peso.value
        let altura = form.altura.value

        //Se um dos dois campos não for informado encerra execução
        if (!peso || !altura) return;

        //Trocando vírgula por ponto
        peso = peso.replace(',', '.')
        altura = altura.replace(',', '.')
```

```
        //Convertendo String em Float
        peso = Number.parseFloat(peso)
        altura = Number.parseFloat(altura)

        //Chama função para calcular o IMC
        const {imc, classificacao, consequencia} = calcularIMC(peso, altura)

        //Chama função para apresentar a resposta
        apresentarResposta(imc, classificacao, consequencia)
    })


    const apresentarResposta = (imc, classificacao, consequencia) => {
        //Obtemos o elemento utilizando a função document.getElementbyId
        //Em seguida substituimos o conteudo com uma template string formatada
        //Aproveitamos para formatar o imc com apenas 1 casa decimal
        document.getElementById('resultado').innerHTML = `
        <h3>${imc.toFixed(1)}kg/m²</h3>
        <h2>${classificacao}</h2>
        <p><strong>O que pode acontecer:</strong></p>
        <p>${consequencia}</p>
        `
    }
</script>
```

Perceba que utilizamos em vários pontos desse código referência a global ***document***, é através dela que conseguimos manipular o DOM (Document Object Model) que é uma hierarquia de componentes da página, esse tipo de hierarquia, onde elementos filhos se aninham em um elemento pai, dentro da programação chamamos de árvore.

Para que não fique redundante, não apresentaremos o código completo do index.html pois esperamos que neste ponto você já tenha pego a ideia. Se tiver feito tudo certo, e sem erros no código, já deve estar com uma aplicação bem interessante, mas não terminamos ainda, nos próximos exercícios vamos aprender a gravas o histórico de cálculos do IMC e apresentar isso ao usuário.

EXERCÍCIO 5: HISTÓRICO DO IMC

Necessitamos apresentar um histórico da evolução do IMC do nosso usuário, para que isso seja possível precisamos armazenas cada cálculo em uma lista e apresentar a listagem do histórico quando solicitado. Para cada vez que o usuário clicar no botão CALCULAR, grave no locaStorage do navegador o peso, a altura, o imc, a classificação, a consequência e a data atual. Por enquanto, se preocupe com o armazenamento desses dados a apresentação será tratada em outro exercício.

Primeiramente o que é localSorage?

É um espaço de armazenamento que o navegador nos fornece para que possamos armazenar dados no formato de texto. Mas perceba que iremos precisar armazenas dados no formato de lista, uma lista com o histórico de todas os cálculos de IMC realizados pela aplicação.

Três conceitos importantes sobre estruturas de dados em JavaScript precisarão ser trabalhados neste ponto. O primeiro é o conceito de ***Object***, que é uma forma de agrupar variáveis sobre determinado conceito do nosso sistema, no nosso caso teremos uma listas de "Cálculos", onde cada um destes "Cálculos" é um Object.

Podemos fabricar Objects, utilizando Class, onde cada objeto descendente da classe herda sua estrutura de dados e comportamentos (funções).

O terceiro conceito que precisamos entender é como criar listas, que no JavaScript chamamos de Array. Este Array é o que armazenaremos no localStorage.

Sei que isso pode estar confuso, mas vai ficar mais simples dando uma olhada no código. Começaremos então, criando nossa fábrica de cálculos, para isso criaremos um novo arquivo imc.js:

```
import calcularIMC from "./calcularIMC.js";

class IMC {

  // Função chamada a cada vez que pedimos que um novo cálculo seja criado
  constructor(peso, altura) {
    // this._peso e this._altura são properties do objeto que esta sendo criado
    // o underline antes do nome da propert simboliza
    // que ela só deve ser acessada dentro da classe
    this._peso = peso;
    this._altura = altura;

    //Armazenaremos a data de criação desse calculo
```

```
    this._data = new Date();
  }

  //Função para calcular
  calcular() {
    //Chamada pra função calcularIMC
    const { imc, classificacao, consequencia } = calcularIMC(
      this.peso,
      this.altura
    );

    //Atribuímos valores a properts do objeto.
    this._imc = imc;
    this._classificacao = classificacao;
    this._consequencia = consequencia;
  }

  //Como não queremos que as properties com "_" sejam manipuladas externamente
  //Fornecemos GETTERS somente leitura para nossas variáveis
  get peso() {
    return this._peso;
  }

  get altura() {
    return this._altura;
  }

  get imc() {
    return this._imc;
  }

  get classificacao() {
    return this._classificacao;
  }

  get consequencia() {
    return this._consequencia;
  }

  get data() {
    return this._data;
  }
}

export default IMC;
```

Com isso precisaremos fazer alterações no index.html, para que passe a utilizar objetos da classe IMC para efetuar os cálculos, apresentamos a seguir apenas o trecho de código alterado:

```
  ...

  //Convertendo String em Float
```

```
  peso = Number.parseFloat(peso)
  altura = Number.parseFloat(altura)

  //Instanciando um novo objeto daclasse IMC
  const imc = new IMC(peso, altura)

  //Chama função para calcular o IMC
  imc.calcular(peso, altura)

  //Chama função para apresentar a resposta
  apresentarResposta(imc)
})


const apresentarResposta = (imc) => {
  //Obtemos o elemento utilizando a função getElementbyId da global document
  //Em seguida substituimos o conteudo com uma template string formatada
  //Aproveitamos para formatar o imc com apenas 1 casa decimal
  document.getElementById('resultado').innerHTML = `
  <h3>${imc.imc.toFixed(1)}kg/m²</h3>
  <h2>${imc.classificacao}</h2>
  <p><strong>O que pode acontecer:</strong></p>
  <p>${imc.consequencia}</p>
  `
}
```

Agora que temos objetos da classe IMC em nossa aplicação, precisamos criar uma lista com esses objetos para assim armazenar nosso histórico. Pra isso criaremos um arquivo chamado imcHistory.js

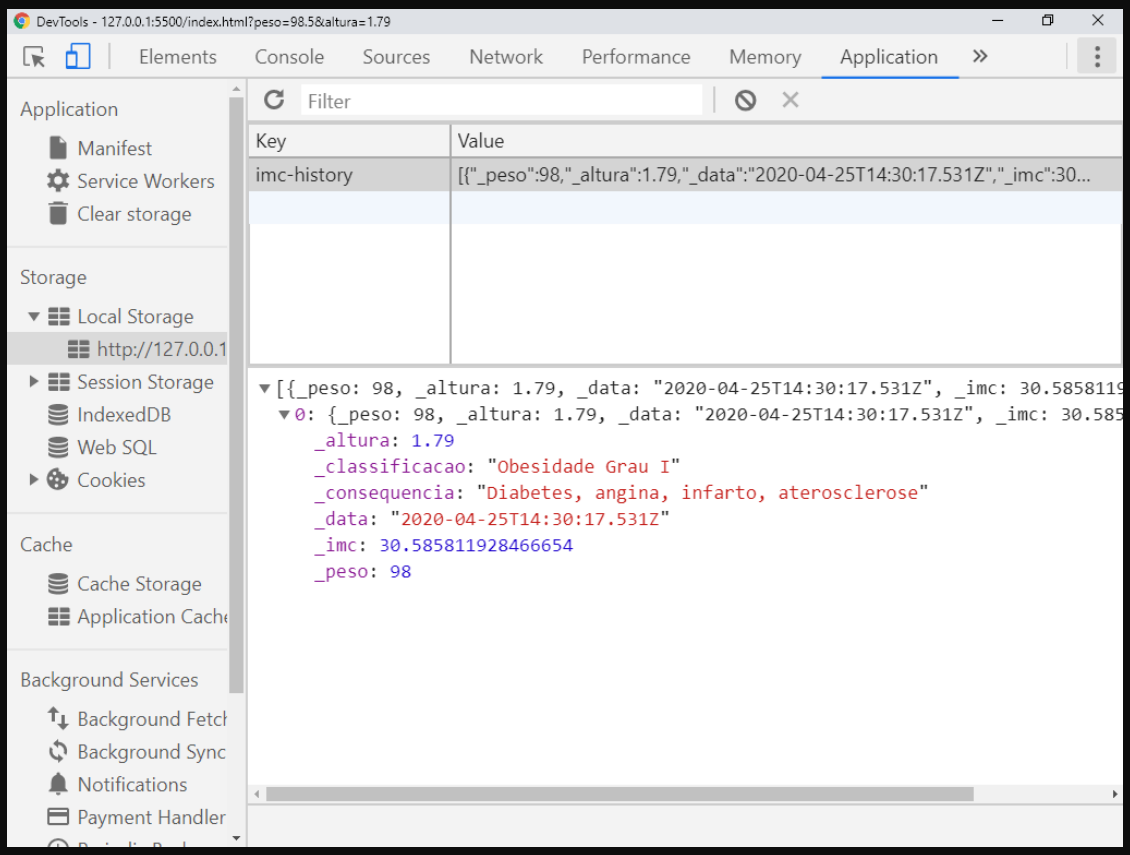
```
class IMCHistory {
  constructor() {
    //Cria lista atribuindo array vazio
    this._history = [];

    //Busca no localStorage o histórico
    const historyJson = localStorage.getItem("imc-history");

    //Caso o histórico ainda não exista encerra construtor
    if (historyJson === null) return;

    //Faz o parse do JSON para this._history
    this._history = JSON.parse(historyJson);
  }

  add(imc) {
    //inclui novo objeto na lista
    this._history.push(imc);

    //transforma this._history em JSON
    const historyJson = JSON.stringify(this._history);

    //escreve lista no localStorage
```

```
    localStorage.setItem("imc-history", historyJson);
  }
}
```

Mais uma vez precisamos alterar o index.html para incluir o novo objeto no nosso histórico:

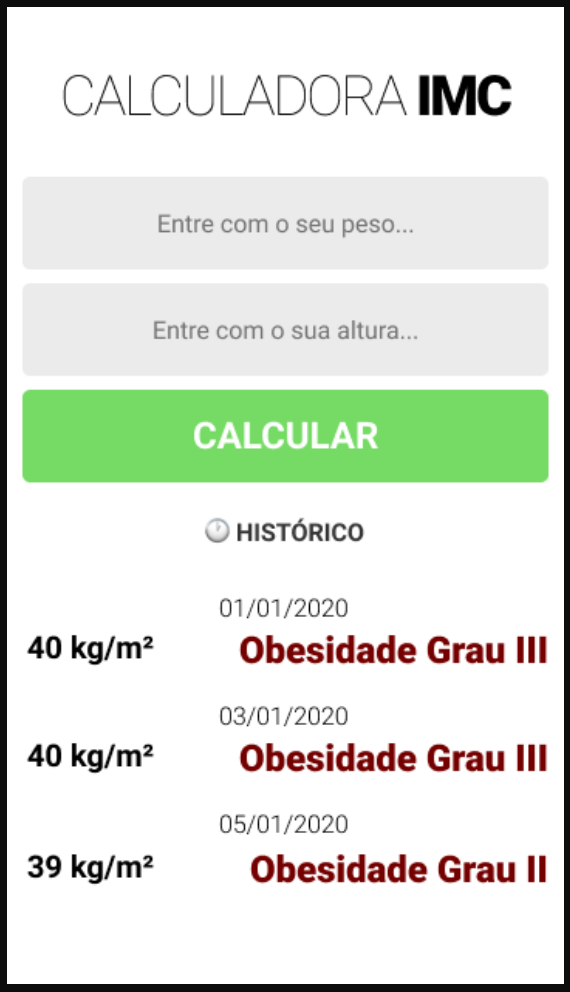
```
        ...
        //Chama função para apresentar a resposta
        apresentarResposta(imc)

        //Salvando dados no localStorage
        const history = new IMCHistory()
        history.add(imc)
    })
```

Se você abrir as ferramentas de desenvolvedor e observar na aba application conforma a imagem a seguir, porá ver que os valores do histórico já estão sendo armazenador.



EXERCÍCIO 6: LISTANDO O HISTÓRICO

Leia do localStorage os dados do histórico e apresente uma listagem em tela conforme o layout a seguir:



Vamos precisar de uma novo botão "HISTÓRICO", e quando acionado ele vai mostrar ou esconder o histórico de leituras do IMC. Abaixo do botão vamos incluir uma div com o id "historico" que vai receber dinamicamente os dados do histórico. Primeiramente faremos a alteração do HTML:

```
</form>
        <button id="btnHistory">&#128336; Histórico</button>
        <div id="history">
            <div class="imc">
                <p>01/01/2020</p>
                <h3>40 kg/m²</h3>
                <h2>Obesidade Grau III</h2>
            </div>
            <div class="imc">
                <p>01/01/2020</p>
                <h3>40 kg/m²</h3>
                <h2>Obesidade Grau III</h2>
            </div>
            <div class="imc">
```

```
                <p>01/01/2020</p>
                <h3>40 kg/m²</h3>
                <h2>Obesidade Grau III</h2>
            </div>
        </div>
    </main>
```

Agora acrescentamos o trecho a seguir no arquivo style.css:

```
button#btnHistory {
    background: none;
    font-size: 16px;
    color: #222222;
    padding: 0;
    margin-top: 20px;
    font-weight: bold;
}

#history {
    display: flex;
    flex-direction: column;
    margin-top: 20px;
}

.imc {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-bottom: 10px;
}

.imc p {
    width: 100%;
}

.imc h2, .imc h3 {
    margin: 0;
}
```

Agora vamos acrescentar ao imcHistory.js um GETTER para pegar a lista do histórico de leituras.

```
get history() {
    return this._history;
}
```

Finalmente faremos a alteração no index.html para que mostre o histórico ao clicar no botão:

```
    //Recuperando botão do histórico
    const btnHistory = document.getElementById("btnHistory")
    //Adicionando escutador ao evento click do botão de hitórico
    btnHistory.addEventListener('click', () => {
        const history = new IMCHistory()
        apresentarHistorico(history)
```

```
    })

    const apresentarHistorico = (history) => {
        //Percorre a lista de historico com a função reduce do array
        //A função reduce retorna um valor unico após percorrer todo o array
        //No nosso código utilizamos reduce para construir uma template com tod
os os históricos
        //Por fim atribuimos o template a div id="history"
        document.getElementById('history').innerHTML = history.history.reduce((
template, hist) => {
            //Transforma string em data
            const data = new Date(hist._data)
            //Monta o template formatando os dados de data e imc
            return template + `
                <div class="imc">
                    <p>${data.toLocaleDateString()}</p>
                    <h3>${hist._imc.toFixed(1)}Kg/m²</h3>
                    <h2>${hist._classificacao}</h2>
                </div>
                `
        }, '')
    }
</script>
```

Poderíamos ter percorrido o array utilizando uma estrutura de repetição como for como veremos a seguir:

```
let template = ''
for (let i = 0; i < history.history.length; i++) {
    const hist = history.history[i]
    const data = new Date(hist._data)
    template = template + `
            <div class="imc">
                <p>${data.toLocaleDateString()}</p>
                <h3>${hist._imc.toFixed(1)}Kg/m²</h3>
                <h2>${hist._classificacao}</h2>
            </div>
            `
}
document.getElementById('history').innerHTML = template
```

Ou ainda utilizando while:

```
let template = ''
let i = 0
while (i < history.history.length) {
    const hist = history.history[i]
    const data = new Date(hist._data)
    template = template + `
        <div class="imc">
            <p>${data.toLocaleDateString()}</p>
            <h3>${hist._imc.toFixed(1)}Kg/m²</h3>
            <h2>${hist._classificacao}</h2>
```

```
        </div>
    `

    i++
}
document.getElementById('history').innerHTML = template
```

Manteremos a versão do código com a utilização da função ***reduce*** presente no array do JavaScript. Todas as versões do código vistas anteriormente são equivalentes, o importante é entender que existem várias formas de percorrer um array, mas a forma mais otimizada e segura atualmente na linguagem JavaScript é utilizando as funções do próprio array como: reduce, forEach, find, filter, etc.

EXERCÍCIO 7: CRIANDO TRANSIÇÃO COM CSS

Acrescente um efeito de transição ao clicar nos botão CALCULAR e HITÓRICO, para que os dados sejam mostrados de forma mais suave. Faça com que ao clicar no botão HITÓRICO pela segunda vez os dados sejam escondidos também com uma transição suave.

Precisaremos agora, combinar o click do botão com efeitos de transição do CSS, isso é possível adicionando e retirando classes aos elementos que desejamos fazer a transição. Incluiremos algumas estilizações ao nosso arquivo style.css;

```
.fade-in {
    visibility: hidden;
    opacity: 0;
    transition: visibility 0.3s linear, opacity 0.1s linear;
}

.visible {
    visibility: visible;
    opacity: 1;
}
```

Agora precisamos acrescentar a classe "fade-in" nas divs do resultado e do histórico:

```
<div id="resultado" class="fade-in"></div>
<div id="history" class="fade-in"></div>
```

Falta apenas escrever o código para acrescentar dinamicamente a classe 'visible' nas funções de apresentação do resultado e do histórico. Vamos começar pela função do resultado:

```
const apresentarResposta = (imc) => {
  //Obtemos o elemento utilizando a função getElementbyId da global document
  const div = document.getElementById('resultado')
  if (div.classList.contains('visible')) {
    div.classList.remove('visible')
  }

  //Substituimos o conteudo com uma template string formatada
  //Aproveitamos para formatar o imc com apenas 1 casa decimal
  div.innerHTML = `
    <h3>${imc.imc.toFixed(1)}kg/m²</h3>
    <h2>${imc.classificacao}</h2>
    <p><strong>O que pode acontecer:</strong></p>
    <p>${imc.consequencia}</p>
  `
  setTimeout(() => div.classList.add('visible'), 500)
}
```

E finalizamos com a alteração da função de apresentação do histórico:

```
const apresentarHistorico = (history) => {
    const div = document.getElementById('history')
    if (div.classList.contains('visible')) {
        div.classList.remove('visible')
        //Como solicitado no enunciádo do exercício apenas escondemos
        return
    }
    //Percorre a lista de historico com a função reduce do array
    //A função reduce retorna um valor unico após percorrer todo o array
    //No nosso código utilizamos reduce para construir uma template com todos o
s históricos
    //Por fim atribuimos o template a div id="history"
    div.innerHTML = history.history.reduce((template, hist) => {
        //Transforma string em data
        const data = new Date(hist._data)
        //Monta o template formatando os dados de data e imc
        return template + `
            <div class="imc">
                <p>${data.toLocaleDateString()}</p>
                <h3>${hist._imc.toFixed(1)}Kg/m²</h3>
                <h2>${hist._classificacao}</h2>
            </div>
            `
    }, '')

    setTimeout(() => div.classList.add('visible'), 500)
}
```

CONCLUSÃO

Se você chegou até aqui, quero te dar os PARABÊNS! Você acabou de criar um webapp completo para cálculo do IMC. Certamente se dedicou bastante em entender cada linha de código e com isso já tem uma noção do JavaScript e de tudo que dá para desenvolver no frontend de aplicações web utilizando recursos poderosos que demonstramos aqui.

Mas saiba que sua jornada como dev frontend está apenas começando, tem muita coisa para aprender e exercitar.

Se tem dúvidas sobre o conteúdo deste ebook, ou não sabe qual seria seu próximo passo na carreira, ou simplesmente para me dizer o que achou bom e o que poderia ser melhor nesse material está aqui meu WhatsApp:

https://bit.ly/FaleComOPauloBezerra

TODOS OS CÓDIGOS

-Index.html

```
<!DOCTYPE html>
<html lang="pt-BR">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>

    <main>
        <h1>Calculadora <strong>IMC</strong></h1>
        <form id="formIMC">
            <input type="number" step="0.01" name="peso" placeholder="Entre com o peso..."
 />
            <input type="number" step="0.01" name="altura" placeholder="Entre com a altura
..." />
            <button type="submit">Calcular</button>
            <div id="resultado" class="fade-in"></div>
        </form>
        <button id="btnHistory">&#128336; Histórico</button>
        <div id="history" class="fade-in"></div>
    </main>


    <script type="module">
        import IMC from './IMC.js'
        import IMCHistory from './imcHistory.js'

        const form = document.forms['formIMC']

        form.addEventListener('submit', (event) => {
            event.preventDefault()
            let peso = form.peso.value
            let altura = form.altura.value

            if (!peso || !altura) return;

            peso = peso.replace(',', '.')
            altura = altura.replace(',', '.')
            peso = Number.parseFloat(peso)
            altura = Number.parseFloat(altura)
            const imc = new IMC(peso, altura)
            imc.calcular(peso, altura)
            apresentarResposta(imc)
            const history = new IMCHistory()
```

```
            history.add(imc)
        })

        const apresentarResposta = (imc) => {
            const div = document.getElementById('resultado')
            if (div.classList.contains('visible')) {
                div.classList.remove('visible')
            }

            div.innerHTML = `
            <h3>${imc.imc.toFixed(1)}kg/m²</h3>
            <h2>${imc.classificacao}</h2>
            <p><strong>O que pode acontecer:</strong></p>
            <p>${imc.consequencia}</p>
            `
            setTimeout(() => div.classList.add('visible'), 500)
        }

        const btnHistory = document.getElementById("btnHistory")
        btnHistory.addEventListener('click', () => {
            const history = new IMCHistory()
            apresentarHistorico(history)
        })

        const apresentarHistorico = (history) => {
            const div = document.getElementById('history')
            if (div.classList.contains('visible')) {
                div.classList.remove('visible')
                return
            }
            div.innerHTML = history.history.reduce((template, hist) => {
                const data = new Date(hist._data)
                return template + `
                    <div class="imc">
                        <p>${data.toLocaleDateString()}</p>
                        <h3>${hist._imc.toFixed(1)}Kg/m²</h3>
                        <h2>${hist._classificacao}</h2>
                    </div>
                    `
            }, '')

            setTimeout(() => div.classList.add('visible'), 500)
        }
    </script>
</body>

</html>
```

-calcularIMC.js

```
const calcularIMC = (peso, altura) => {
```

```
  const imc = peso / (altura * altura);

  let classificacao = "";
  let consequencia = "";
  if (imc >= 16 && imc < 17) {
    classificacao = "Muito abaixo do peso";
    consequencia = "Queda de cabelo, infertilidade, ausência menstrual";
  } else if (imc >= 17 && imc < 18.5) {
    classificacao = "Abaixo do peso";
    consequencia = "Fadiga, stress, ansiedade";
  } else if (imc >= 18.5 && imc < 25) {
    classificacao = "Peso normal";
    consequencia = "Menor risco de doenças cardíacas e vasculares";
  } else if (imc >= 25 && imc < 30) {
    classificacao = "Acima do peso";
    consequencia = "Fadiga, má circulação, varizes";
  } else if (imc >= 30 && imc < 35) {
    classificacao = "Obesidade Grau I";
    consequencia = "Diabetes, angina, infarto, aterosclerose";
  } else if (imc >= 35 && imc <= 40) {
    classificacao = "Obesidade Grau II";
    consequencia = "Apneia do sono, falta de ar";
  } else if (imc > 40) {
    classificacao = "Obesidade Grau III";
    consequencia =
      "Refluxo, dificuldade para se mover, escaras, diabetes, infarto, AVC";
  }

  return {
    imc,
    classificacao,
    consequencia,
  };
};

export default calcularIMC;
```

-imc.js

```
import calcularIMC from "./calcularIMC.js";

class IMC {
  constructor(peso, altura) {
    this._peso = peso;
    this._altura = altura;

    this._data = new Date();
  }

  calcular() {
```

```
    const { imc, classificacao, consequencia } = calcularIMC(
      this.peso,
      this.altura
    );

    this._imc = imc;
    this._classificacao = classificacao;
    this._consequencia = consequencia;
  }

  get peso() {
    return this._peso;
  }

  get altura() {
    return this._altura;
  }

  get imc() {
    return this._imc;
  }

  get classificacao() {
    return this._classificacao;
  }

  get consequencia() {
    return this._consequencia;
  }

  get data() {
    return this._data;
  }
}

export default IMC;
```

-imcHistory.js

```
class IMCHistory {
  constructor() {
    this._history = [];
    const historyJson = localStorage.getItem("imc-history");
    if (historyJson === null) return;
    this._history = JSON.parse(historyJson);
  }

  add(imc) {
    this._history.push(imc);
    const historyJson = JSON.stringify(this._history);
```

```
    localStorage.setItem("imc-history", historyJson);
  }

  get history() {
    return this._history;
  }
}

export default IMCHistory
```

-style.css

```
@import url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap');

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

html,
body {
    height: 100vh;
}

body {
    font-family: Roboto;
    display: flex;
    flex-direction: column;
    justify-content: space-around;
}

main {
    margin: 0 auto;
    min-height: 500px;
    width: 100%;
    max-width: 360px;
    padding: 10px;
    text-align: center;
    display: flex;
    flex-direction: column;
    font-size: 14px;
}

h1 {
    font-weight: 100;
    text-transform: uppercase;
    font-size: 36px;
    margin-bottom: 20px;
```

```
    letter-spacing: -0.05em;
}

h1 strong {
    font-weight: 900;
}

input {
    display: block;
    width: 100%;
    font-size: 16px;
    padding: 15px 10px;
    margin: 10px 0;
    border-radius: 5px;
    border: none;
    background-color: #EBEBEB;
    text-align: center;
}

button {
    display: block;
    width: 100%;
    font-size: 16px;
    padding: 15px 10px;
    margin: 10px 0;
    border-radius: 5px;
    border: none;
    background-color: #75DB64;
    text-transform: uppercase;
    color: #FFFFFF;
    font-size: 24px;
}

h3 {
    font-weight: bold;
    font-size: 18px;
    margin-top: 20px;
}

h2 {
    color: #750000;
    font-size: 24px;
    margin: 10px 0;
    font-weight: 900;
}

#resultado {
    margin-bottom: 20px;
}

button#btnHistory {
    background: none;
    font-size: 16px;
```

```
    color: #222222;
    padding: 0;
    margin-top: 20px;
    font-weight: bold;
}

#history {
    display: flex;
    flex-direction: column;
    margin-top: 10px;
}

.imc {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-bottom: 10px;
}

.imc p {
    width: 100%;
}

.imc h2, .imc h3 {
    margin: 0;
}

.fade-in {
    visibility: hidden;
    opacity: 0;
    transition: visibility 0.3s linear, opacity 0.1s linear;
}

.visible {
    visibility: visible;
    opacity: 1;
}
```

CODDEV.COM.BR