

# *Desenvolvimento de interfaces Web*

**HTML**



**CSS**



**JS**



***com Javascript***

Introdução.....	1
Capítulo 1. Conhecendo o html.....	2
1.1.- O que é o HTML.....	2
1.2. A história do HTML.....	2
1.3. Tags Html.....	2
1.4. Tags semânticas ou estruturais.....	3
1.4.1. Exercícios propostos.....	5
Capítulo 2. CSS.....	6
2.1 O que é CSS.....	6
2.2 Como incorporar o estilo a página html.....	6
2.3. Regras CSS.....	7
Capítulo 3. Conhecendo o Javascript.....	7
3.1. História do Javascript.....	8
3.2. Características do JavaScript.....	9
3.3. Variáveis Javascript.....	8
3.4.1. Operadores Aritméticos.....	8
3.4.2. Operadores Unários.....	9
3.4.3. Operadores de atribuição compostos.....	10
3.4.4. Operadores relacionais.....	11
3.4.5. Operadores lógicos.....	12
3.4.6. Operador ternário.....	13
3.5. Exercícios propostos.....	14
Capítulo 4. Git & GitHub.....	17
4.1. Principais comandos Git.....	17
4.2. Exercícios propostos.....	19
Capítulo 5 Controle de fluxo.....	20
5.1. Estruturas de decisão.....	21
5.1.1. Estrutura de decisão if.....	21
5.1.1. Estrutura de decisão if-else.....	22
5.1.2. Estrutura de decisão if-else-if.....	23
5.1.3. Estrutura de decisão if-else aninhado.....	24
5.2. Estrutura de decisão SWITCH.....	25
5.2. Estruturas de repetição.....	26
5.2.1. Estrutura de repetição While.....	26
5.2.2. Estrutura de repetição Do While.....	27
5.2.2. Estrutura de repetição For.....	29
5.3. Exercícios propostos.....	30



## **Introdução**

Javascript tornou-se a linguagem mais utilizada para desenvolvimento de páginas web e é bastante popular no desenvolvimento mobile. Estando no top 3 de todas as listas de linguagens das melhores linguagens de programação vem conquistando, a cada dia que passa, uma maior parcela do mercado.

Essa apostila tem como objetivo ajudar ao aluno a aprender a desenvolver aplicativos web com Javascript usando Node.js e React.js, utilizando o editor de texto Visual Studio Code, disponibilizado de maneira gratuita pela Microsoft corporation que vem com integração ao prompt de comando do windows e propicia uma melhor experiência na criação de códigos.

Vamos aprender também os conceitos da programação Declarativa e as diferenças para com o paradigma imperativo não restringindo o aprendizado somente ao Javascript. Teremos também módulos de HTML 5 e CSS 3 que são essenciais na construção de páginas web bem como aprender a usar o Git e o GitHub que vão nos ajudar no controle de versionamento dos nossos projetos bem como propiciar um repositório na nuvem.

## 1. Conhecendo o HTML.

### 1.1. O que é HTML?

HTML é a linguagem na qual a maioria dos sites é escrita. HTML é usado para criar páginas e torná-las funcionais.

O código usado para torná-los visualmente atraentes é conhecido como CSS e vamos nos concentrar nisso em um tutorial posterior. Por enquanto, vamos nos concentrar em ensinar a você como construir, em vez de projetar.

### 1.2. A História do HTML

O HTML foi criado por Tim Berners-Lee, Robert Cailliau e outros a partir de 1989. Significa Hyper Text Markup Language.

Hipertexto significa que o documento contém links que permitem ao leitor pular para outros lugares no documento ou para outro documento. A versão mais recente é conhecida como HTML5.

Uma linguagem de marcação é uma maneira pela qual os computadores se comunicam para controlar como o texto é processado e apresentado. Para fazer isso, o HTML usa duas coisas: tags e atributos.

### 1.3. tags HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estrutura padrão</title>
    <meta charset="utf-8">
  </head>
  <body>
  </body>
</html>
```

São responsáveis por criar os elementos na página. Cada tag especifica um elemento e também pode ser dividida em: Elementos de bloco, ou de linha. Para simbolizar uma tag nós a representamos da seguinte maneira: `<span></span>`.

A maioria das tags deve ser aberta `<h1>` e fechada `</h1>` para funcionar.

**`<html></html>`**– Essa é a tag principal do HTML a que irá conter todas as outras dentro de si.

**`<head></head>`**– É o cabeçalho da estrutura das nossas páginas. O head vai conter informações especiais como o encoding da página a importação de estilos, título entre outras coisas.

**`<title></title>`**– É o título que aparecerá na sua página HTML.

**`<meta/>`**– É a tag que seta o encoding da página para nós brasileiros usaremos o **UTF-8**, que conterà todos os símbolos da norma ABNT-2.

**`<body></body>`**– Como o próprio nome diz é o corpo da nossa página vai conter tudo o que for apresentado pelo Browser.

#### **1.4. Tags semânticas ou estruturais.**

**`<header></header>`** : Enquanto o head é o cabeçalho da estrutura. Header é o cabeçalho da página. E nele estarão contidas a `<nav>` e a logo do site,

**`<nav></nav>`**– Essa tag define um conteúdo de navegação. Portanto, é muito utilizado em conjunto com listas e na criação de menus;

**`<main></main>`** É a tag que vai conter o conteúdo principal, assim como as tags **`<section>`**, **`<article>`** e **`<aside>`**

**<section></section>** Define uma sessão do article.

**<article></article>** Se trata do conteúdo, notícia de destaque, na página.

**<aside></aside>** é uma sessão lateral que pode conter uma navegação dentro do article ou opções de navegação da página.

**<footer></footer>** Representa o rodapé, geralmente contém informações como informações e links para outros sites relacionados a empresa.

**<div></div>** Especifica divisões dentro da página e podem englobar outras tags delimitando assim um container.



## 1.5. Exercícios propostos

1) Faça uma página html somente com uma estrutura básica.

2) Qual das opções abaixo serve para setar o encoding?

- a) head
- b) body
- c) html
- d) meta
- e) footer

**3) Qual dessas tags abriga todas as outras dentro de si:**

- a) head
- b) body
- c) html
- d) meta
- e) footer

**4) Dentre essas tags em qual delas que poderemos colocar a tag title?**

- a) head
- b) body
- c) html
- d) meta
- e) footer

**5) É uma tag semântica que fica no rodapé da página html:**

- a) head
- b) body
- c) html
- d) meta
- e) footer

**6) É a principal tag html.**

- a) head
- b) body
- c) html
- d) meta
- e) footer

**7) Construa um formulário HTML de acordo com o que foi visto nas aulas deste módulo.**



## 2. CSS

CSS é usado para controlar o estilo de um documento da web de uma forma simples e fácil.

CSS é a sigla para "Cascading Style Sheet". Este tutorial cobre as versões CSS1, CSS2 e CSS3, e dá uma compreensão completa de CSS, desde seus conceitos básicos até conceitos avançados.

### 2.1. Por que aprender CSS?

Cascading Style Sheets, carinhosamente conhecido como CSS, é uma linguagem de design simples destinada a simplificar o processo de tornar as páginas da web apresentáveis.

CSS é uma OBRIGAÇÃO para estudantes e profissionais que trabalham para se tornarem um grande Engenheiro de Software, especialmente quando estão trabalhando no Domínio de Desenvolvimento Web. Vou listar algumas das principais vantagens de aprender CSS:

Crie um site impressionante - CSS controla a aparência e o comportamento de uma página da web. Usando CSS, você pode controlar a cor do texto, o estilo das fontes, o espaçamento entre os parágrafos, como as colunas são dimensionadas e dispostas, quais imagens ou cores de fundo são usadas, designs de layout, variações na exibição para diferentes dispositivos e tamanhos de tela bem como uma variedade de outros efeitos.

### 2.2. Como incorporar um estilo a uma página.

Incorpore o CSS ao seu HTML usando a tag link e passando um rel="stylesheet"

```
<!DOCTYPE html>
<html>
  <head>
    <title>Aula01</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="./css/estilo.css">
  </head>
  <body>
    <header>

    </header>
    <nav>
```

### 2.3. Regras CSS

O próximo passo é aplicar as regras de CSS a nossa folha de estilo.

```
body{
  height: 100%;
  margin: 0px 5px;
  color: ■aliceblue;
  font-size: 1rem;
  font-family: Arial, Helvetica, sans-serif;
  background-color: ■rgb(95,165,210);
}

main{
  display: flex; flex-direction: row;
}

a{
  color: ■aliceblue;
}
```

## 3. Conhecendo o JavaScript.

### 3.1. História do Javascript.

Na década de 90 a empresa Netscape dominava o setor de navegadores web com o Netscape Navigator inspirado no Mosaic da NCSA. Surgiu então a necessidade de fazer com que as páginas web fossem mais interativas fazendo a autenticação de dados passados por usuários em formulário entre outras coisas. A proposta inicial era fazer uma linguagem Scheme baseada em Lisp, contudo esse projeto foi posto de lado.

O Javascript foi desenvolvido por Brendan Eich e lançado em 1995 com o nome Livescript pela Netscape Corporation no navegador Netscape 2.0. Com a ascensão da linguagem Java a Netscape renomeou seu projeto de Javascript.

Com a Microsoft lançando seu próprio navegador com a linguagem Jscript a Netscape resolveu, para não perder o Javascript, lançar uma padronização e para isso contou com a parceria da empresa europeia ECMA, cedendo assim seu código a eles e criando a ECMAScript que é a linguagem de padronização do Javascript.

Em 2002 os ex funcionários da Netscape fundaram o Mozilla Firefox. Agora tínhamos o internet explorer, firefox e um novo browser o Google Chrome que trouxe o engine V8, em 2009 que tem o código aberto e roda o Javascript.

### 3.2. Características da linguagem Javascript.

O Javascript é uma linguagem fracamente tipada é de alto nível e leve. É uma linguagem que usa vários paradigmas, ou seja, tem suporte tanto para programação funcional quanto a programação orientada a objetos. Podendo seus códigos serem programados de maneira declarativa.

Programação imperativa consiste no programador estabelecer o passo a passo de um algoritmo utilizando as estruturas de controle e descrevendo como o programa deve responder a cada situação de maneira procedural.

Programação declarativa foca mais na solução desses problemas do que nos procedimentos em si utilizando métodos e funções para obter o resultado desejado, deixando o código mais legível e menos verboso, o que vai facilitar a manutenção e o entendimento do código.

### 3.3 Variáveis JavaScript.

O é uma linguagem fracamente tipada, o que significa que não precisamos declarar o tipo das variáveis na hora da criação das mesmas e durante a execução do programa elas podem mudar de tipo. O javascript reconhece como valores os seguintes tipos:

- Strings: São os valores passados de maneira textual dentro de aspas duplas, ou simples.
- Number: São números, inteiros (3, 19, 462) e decimais como (3,9).
- Boolean: Valores lógicos true, ou false.
- Null: é o tipo primitivo de nulo.
- Undefined: valor indefinido.

```
>> let nome = "Ana", idade=35, altura=1.72, doador= true  
  
console.log(nome);  
console.log(idade);  
console.log(altura);  
console.log(doador);
```

#### 3.4.1. Operadores aritméticos.

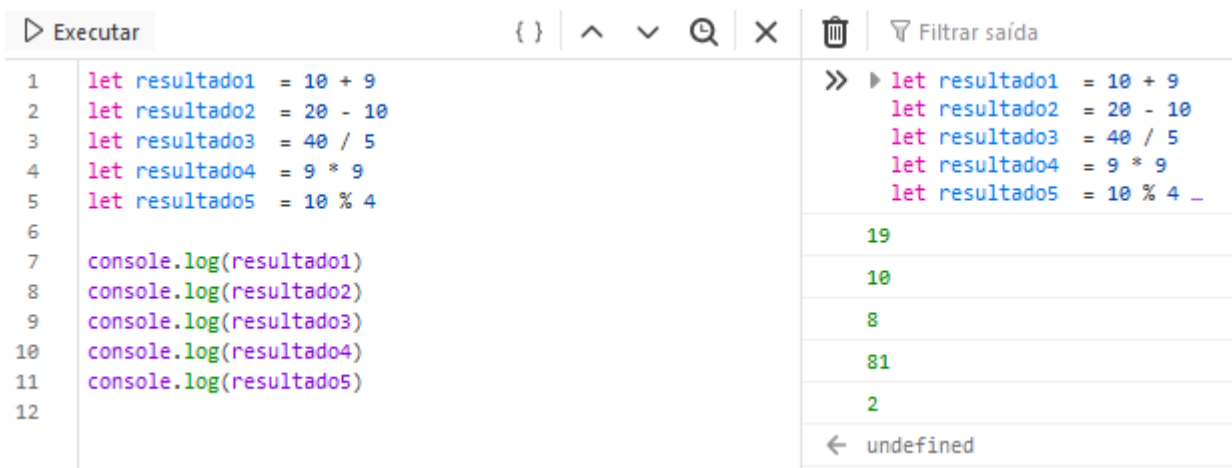
Como o próprio nome diz esses operadores representam as operações básicas da matemática como soma ( + ), subtração ( - ), multiplicação ( \* ), divisão ( / ) e etc. Podemos usar mais de um desses operadores em uma sentença matemática para

chegarmos ao resultado pretendido, as vezes apenas um deles é suficiente para resolver cálculos simples.

A lei de precedência da matemática também se aplica a programação. Então primeiro serão resolvidos os cálculos entre parênteses, raízes e exponenciais, multiplicação e divisão na ordem que aparecem, soma e subtração na ordem que aparecem, sempre da esquerda para a direita.

Operador de módulo % é utilizado quando desejamos saber o resto de uma divisão. Por exemplo 19 % 2 retorna o resultado 1 já que 19 / 2 é igual a 9 e resta 1. 25%7 retorna 4 que é o resto da divisão. É bem simples de entender e muito útil para determinadas situações. Segue abaixo uma tabela com o resumo dos operadores aritméticos vistos nesse tópico.

operador	função	exemplo
+	0	variavel1 + variavel2
-	0	variavel1 - variavel2
*	0	variavel1 * variavel2
/	0	variavel1 / variavel2
%	0	variavel1 % variavel2



```
1 let resultado1 = 10 + 9
2 let resultado2 = 20 - 10
3 let resultado3 = 40 / 5
4 let resultado4 = 9 * 9
5 let resultado5 = 10 % 4
6
7 console.log(resultado1)
8 console.log(resultado2)
9 console.log(resultado3)
10 console.log(resultado4)
11 console.log(resultado5)
12
```

Output:

```
>> ▶ let resultado1 = 10 + 9
    let resultado2 = 20 - 10
    let resultado3 = 40 / 5
    let resultado4 = 9 * 9
    let resultado5 = 10 % 4 _
19
10
8
81
2
← undefined
```

### 3.4.2. Operadores unários.

Operadores unários são de simples entendimento. Eles servem para modificar uma variável ao mesmo tempo que atribuem o novo valor em uma só tacada. Para fazermos

uso dos operadores unários lançamos mão dos operadores aritméticos só que dessa vez, ao invés, de termos dois ou mais componentes temos apenas um.

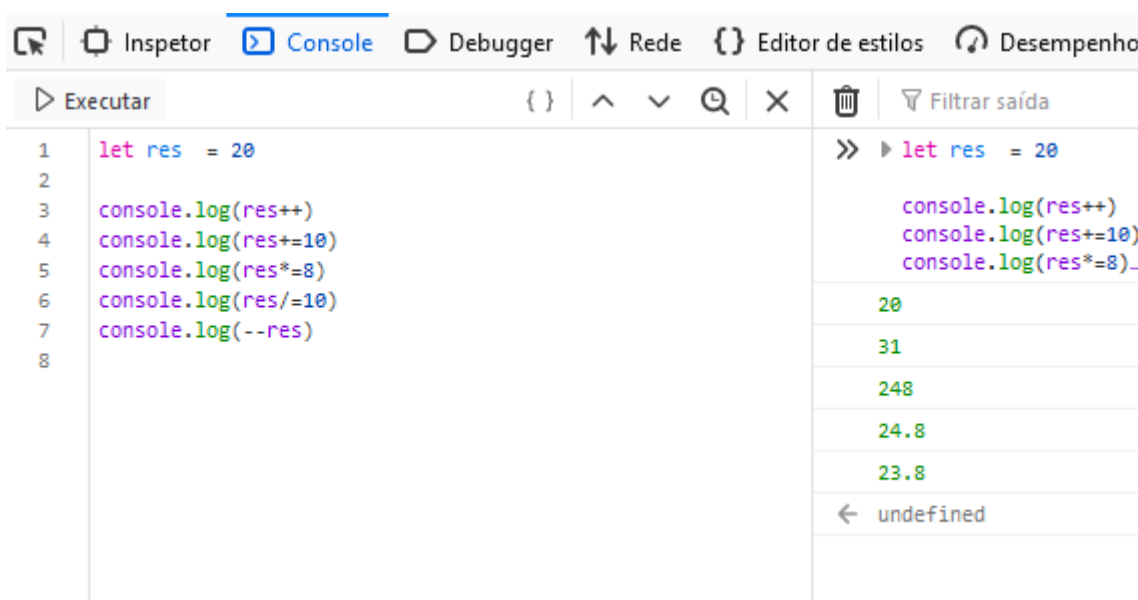
Imagine que temos uma variável chamada **num** que foi atribuído o valor de 10. Quando fazemos **num++** esse valor será incrementado de uma unidade e passará a ser 11. Se fizermos **num--** esse valor será decrementado de uma unidade e voltará ao valor 10, chamamos essa operação de pós incremento e pós decremento.

Também poderíamos fazer **++num** e **--num** que chamamos de pré incremento e pré decremento. A diferença entre o pré e o pós incremento e decremento é que o pré primeiro modifica a variável e depois realiza as operações da linha de código na qual se apresenta, como por exemplo, um **print** e o pós primeiro realiza o **print** e depois altera o valor da variável.

O operador unário **!** faz com que os valores booleanos seja invertido, ou seja, um valor **true** se tornará **false** e vice-versa. Será muito utilizado quando estivermos vendo estruturas de decisão.

### 3.4.3. Operadores de atribuição compostos.

Os operadores de atribuição compostos em termos de aplicabilidade são iguais aos unários de decremento e incremento só que além de apenas somar ou subtrair uma unidade agora podemos usar outros números e outras operações como multiplicação e divisão assim como veremos no exemplo a seguir. Essa é uma ferramenta poderosa para deixar o código mais limpo e simples. Existem cerca de 11 operadores de atribuição compostos, mas veremos somente os mais importantes como **+=**, **-=**, **\*=** e **/=**. Lembrando que esses operadores servem quando se quer mudar o valor de uma variável sem ter que ficar repetindo a mesma desnecessariamente.



```
1 let res = 20
2
3 console.log(res++)
4 console.log(res+=10)
5 console.log(res*=8)
6 console.log(res/=10)
7 console.log(--res)
8
```

Console output:

```
>> let res = 20
console.log(res++)
console.log(res+=10)
console.log(res*=8)
20
31
248
24.8
23.8
← undefined
```

### 3.4.4. Operadores relacionais.

Operadores relacionais vão pegar dois ou mais dados, fazer a comparação entre eles e depois retornar um valor booleano **true** ou **false**. Começemos com

`==` operador relacional igualdade

`!=` operador relacional desigualdade

Esses operadores fazem a comparação entre os valores de duas variáveis e verifica se eles são iguais ou diferentes. Muita atenção o operador de igualdade são dois símbolos de `=`, muito cuidado na hora do código para não confundir e errar. Abaixo temos a tabela verdadeira desses dois operadores.

Operador	igual	diferente
<code>num == num2</code>	0	0
<code>num != num2</code>	0	0

Não podemos usar os comparadores para comparar duas variáveis de tipos primitivos diferentes somente se forem os tipos numéricos, `int`, `float` e `double` entre si, porém `char` e `int`, `String` e `char` não é uma operação que seja possível de ser realizada e retornará um erro em seu código.

`>` maior que

`<` menor que

`>=` maior ou igual que

`<=` menor ou igual que

Assim como a igualdade e a desigualdade esses operadores vão fazer a comparação entre duas variáveis e retornar valores booleanos de verdadeiro ou falso. São amplamente utilizados nas estruturas de decisão, como veremos nos próximos capítulos. Para quem tem dificuldade de saber o que é maior e o que é menor é só você pensar o sinal, se verdadeiro, sempre vai apontar para o menor número, maior `>` menor, menor `<` maior. Abaixo temos uma tabela para facilitar o entendimento.

Valores A e B	<code>A &gt; B</code>	<code>A &lt; B</code>	<code>A &gt;= B</code>	<code>A &lt;= B</code>
2 e 2	0	0	0	0
5 e 8	0	0	0	0
10 e 1	true	false	true	false

Veja a figura abaixo a representação da tabela com duas variáveis  $a = 2$  e  $b = 8$ .

```
let num1 = 10, num2 = 30;

console.log(num1>num2);
console.log(num1>=num2);
console.log(num1<num2);
console.log(num1<=num2);
console.log(num1==num2);
console.log(num1!=num2);
```

```
>> ▶ let num1 = 10, num2 = 30;
      console.log(num1>num2);
      console.log(num1>=num2);
      console.log(num1<num2);_
false
false
true
true
false
true
← undefined
```

### 3.4.5. Operadores lógicos.

Operadores lógicos vão analisar dois booleanos, diferente do que vimos até agora aonde tratávamos com variáveis, e dependendo do valor destes nós teremos uma tabela verdade. Vamos falar de cada um deles.

&& operador “E” lógico. Só retorna verdadeiro caso todas as proposições lógicas sejam verdadeiras, caso contrário retornará falso. Exemplo: Sua mãe te manda para comprar pão e leite. Você só terá sucesso na tarefa se comprar os dois, caso falte algum você levará uma bronca.

|| operador “OU” lógico. O “ou” retornará verdadeiro se qualquer uma, ou todas as proposições lógicas forem verdadeiras. O único caso no qual o “ou” retornará falso é aquele no qual todas as proposições são falsas. Exemplo: Dessa vez sua tarefa é ir na mercearia e trazer dois shampoos da marca A ou B, ou você pode trazer um de cada. Qualquer arranjo será aceito. Sua tarefa só falhará, caso não traga nenhum para casa.

```

1
2 console.log(true&&true);
3 console.log(true&&false);
4 console.log(true||true);
5 console.log(true||false);
6
7
8
9
10
11
12

```

```

>> console.log(true&&true);
    console.log(true&&false);
    console.log(true||true);
    console.log(true||false);

true
false
true
true
← undefined

```

### 3.4.6. Operador ternário.

Esse operador em específico é dominado por qualquer pessoa que tenha feito um curso básico de Excel ou Calc. Operador terciário é utilizado quando temos uma proposição lógica e queremos um determinado resultado para o caso a resposta seja verdadeira e outro para caso seja falsa. Como os operadores unários e de atribuição composta é uma ótima ferramenta a ser utilizada para enxugar o código e deixá-lo fácil de ler quando precisamos de uma estrutura de decisão simples. Vejamos um exemplo na figura abaixo:

```

1 let num1 = 7, num2 = 9
2
3 let resultado = num1>num2?'7 é maior do que 9':'7 é menor do que 9';
4
5 console.log(resultado);
6
7
8
9
10

```

```

>> let num1 = 7, num2 = 9

let resultado = num1>num2?'7 é maior do que 9':'7 é menor do que 9';
console.log(resultado);

7 é menor do que 9
← undefined

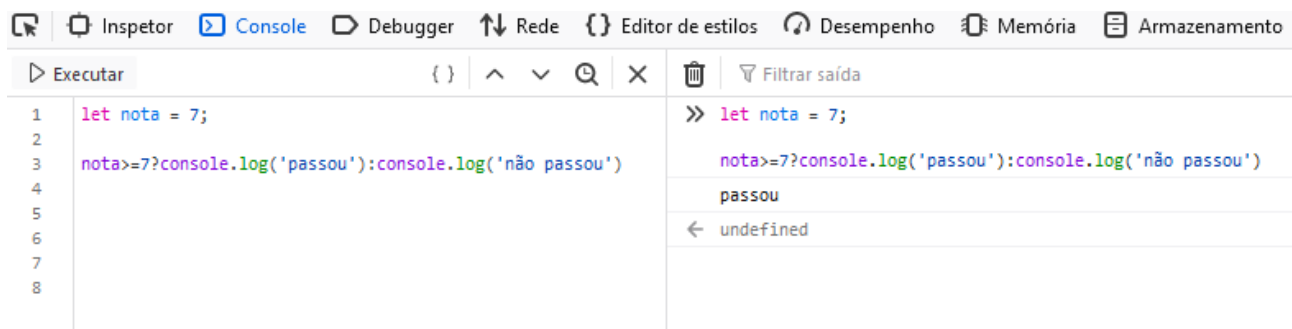
```

Vamos analisar apenas o operador ternário. Na primeira parte temos a proposição lógica que vai pegar num e comparar se é maior do que num2 note que uma parte muito importante é o ponto de interrogação após o teste lógico. Na segunda parte, após o ponto de interrogação temos o valor assumido pelo operador caso o teste retorne verdadeiro e separamos do outro valor por dois pontos. Ficando a estrutura da seguinte maneira.

**(teste lógico) ? (resultado para verdadeiro) : (resultado para falso);**

Vamos fazer um outro exemplo para fixar melhor o aprendizado. Dessa vez usaremos o exemplo clássico do aluno e sua aprovação.





```
1 let nota = 7;
2
3 nota >= 7 ? console.log('passou') : console.log('não passou')
4
5
6
7
8
```

Console output:

```
>> let nota = 7;
nota >= 7 ? console.log('passou') : console.log('não passou')
passou
← undefined
```

No exemplo abaixo vemos que o teste lógico recebe uma determinada nota e verifica. Caso a nota seja maior ou igual a sete resultado receberá “aprovado”

### 3.5. Exercícios propostos.

**1) Qual o operador é utilizado para comparar dois valores numéricos e a partir dessa comparação retornar verdadeiro ou falso.**

- a) operador ternário
- b) operador aritmético
- c) operador de atribuição composto
- d) operador lógico
- e) operador unário.

**2) Qual operador que verifica duas ou mais proposições lógicas e verifica se são verdadeiras ou falsas, fazem a relação entre as duas e retorna verdadeiro ou falso, pode ser um E ou um Ou.**

- a) operador ternário
- b) operador aritmético
- c) operador de atribuição composto
- d) operador lógico
- e) operador unário.

**3) Operador que recebe um teste lógico e retorna um valor para falso e um para verdadeiro.**

- a) operador ternário
- b) operador aritmético
- c) operador de atribuição composto

- d) operador lógico
- e) operador unário

**4) Operador que a partir de um único valor possa modificá-lo utilizando os símbolos aritméticos.**

- a) operador ternário
- b) operador aritmético
- c) operador de atribuição composto
- d) operador lógico
- e) operador unário

**5) São os operadores responsáveis por fazer as quatro operações matemáticas.**

- a) operador ternário
- b) operador aritmético
- c) operador de atribuição composto
- d) operador lógico
- e) operador unário

**6) Quem foi o criador do Javascript?**

- a) Brendan Eich
- b) Tim Bernes Lee
- c) Steve Jobs
- d) Bill Gates
- e) Elon Musk

**7) É a característica que o Javascript tem de ter variáveis que podem mudar de tipo e que não precisam ter um tipo declarado na hora da sua criação.**

- a) multiparadigma.
- b) fracamente tipada.
- c) compilada.

- d) interpretada.
- e) alto nível

**8) O fato de Javascript ter suporte a orientação a objeto e a programação funcional significa que ele é:**

- a) multiparadigma.
- b) fracamente tipada.
- c) compilada.
- d) interpretada.
- e) alto nível

**9) Por abstrair todas as conexões do software com o hardware do programador Javascript pode ser considerado:**

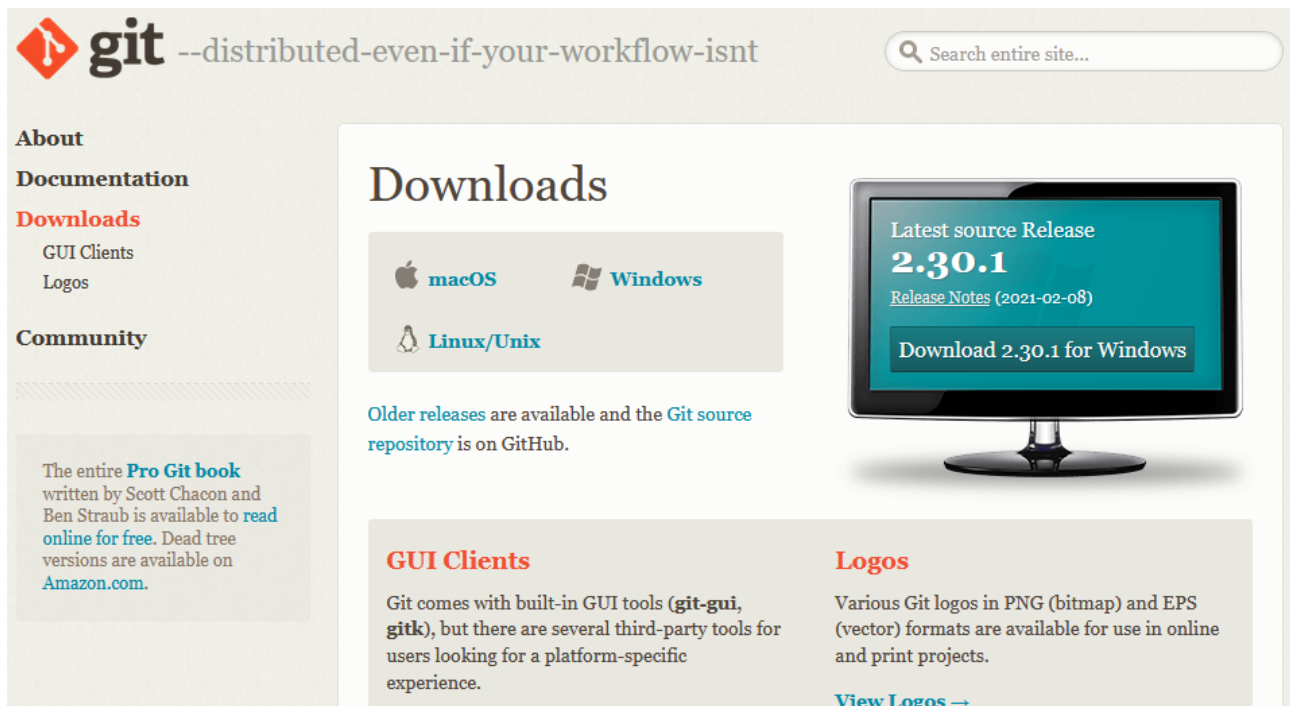
- a) multiparadigma.
- b) fracamente tipada.
- c) compilada.
- d) interpretada.
- e) alto nível

**10) Qual empresa foi responsável pela criação do Javascript?**

- a) Apple
- b) Windows
- c) Netscape
- d) Mozilla
- e) IBM

#### **4. Git & GitHub**

Ambas são ferramentas de controle e versionamento de projetos. Muito utilizadas no mundo da programação para gerência de projetos e ferramentas de colaboração. Para instalar o Git no computador você deve ir ao endereço: <https://git-scm.com/downloads> e baixar o git, logo após instalar.



Siga as instruções de instalação. Você irá escolher o diretório onde será instalado e depois de finalizada a instalação basta abrir o prompt de comando e digitar: **git - - version**

#### 4.1 Lista de principais comando Git.

Setar os dados do usuário:

```
git config --global user.name "João da silva"
git config --global user.email joao@example.com
```

Iniciar um repositório:

```
git init
```

Adicionar arquivos para poder commitar

```
git add <nome do arquivo>
git add *
```

Commitar as mudanças para o cabeçalho, mas não para o repositório online

```
git commit -m "mensagem que identifica o commit"
```

Adicionar a origem de um repositório on line.

```
git remote add origin <endereço do site>
```

Remover a origem do repositório

```
git remote remove origin
```

Adicionar os arquivos do repositório local para o repositório on line.

```
git push origin master
```

Checar o status dos arquivos

```
git status
```

Criar um novo branch ao repositório

```
git checkout -b <nome do branch>
```

Trocar para um branch já existente

```
git checkout <nome do branch>
```

Listar todos os branches

```
git branch
```

Adicionar o branch ao seu repositório remoto

```
git push origin <nome do branch>
```

Adicionar todos os branches para o seu repositório remoto

```
git push - -all origin
```

Deletar um branch do seu repositório remoto

```
git push origin :<nome do branch>
```

## **4.2. Exercícios propostos**

**1) É uma ferramenta de controle de versionamento e gerência de arquivos on line. Que serve também de repositório para os seus projetos.**

- a) Git
- b) Java EE

- c) GitHub
- d) ArrayList
- e) JVM

**2) Qual o comando git para adicionar o endereço do seu repositório on line no seu repositório local.**

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

**3) Qual o comando git para iniciar o repositório local?**

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

**4) Qual o comando que utilizamos para enviar nossos arquivos para o repositório na nuvem?**

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

**5) Comando git utilizado para checar o status dos seus arquivos?**

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

**6) Comando para apagar a origem remota do seu repositório localiza**

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master

e) git status

**7) Comando utilizado para criar um novo branch no repositório.**

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

**8) Comando que mostra todos os branches no repositório.**

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

**9) Deletar o branch no repositório remoto.**

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

**10) Adicionar os arquivos para staging**

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

**Capítulo 2. Controle de fluxo.**

As estruturas de controle definem a **sequência de execução das instruções**. Não é possível implementar um algoritmo que não seja trivial em uma linguagem de programação que não tenha um conjunto mínimo de estruturas de controle. As estruturas de controle podem ser divididas em *seleção*, *repetição* e *sequência*. A sequência é simplesmente definida pela **execução sequencial dos comandos, de cima para baixo**. O controle de fluxo é essencial em qualquer linguagem de programação pois através dele que o programa executa a rotina para o qual foi criado assim como define um programa como imperativo e não declarativo.

Cada programa é feito com um propósito: contar o tempo, classificar produtos, efetuar uma venda, calcular média e várias outras funcionalidades. Para que o programa

execute a sua função usaremos as estruturas de controle. Começaremos pelas estruturas de decisão.

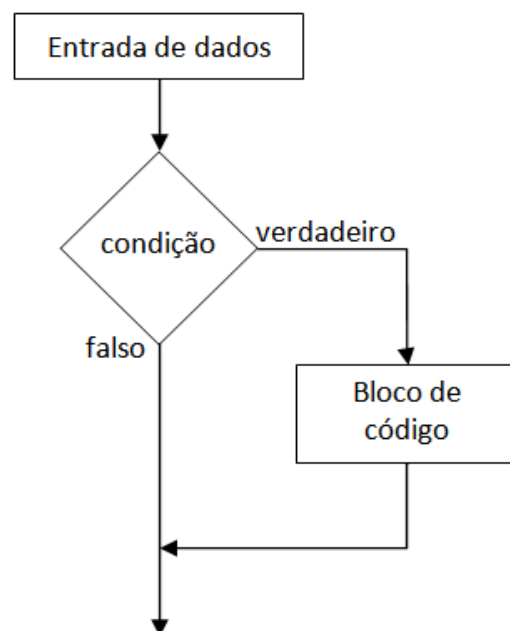
## 2.1. Estruturas de decisão.

Durante o nosso dia, desde a hora que acordamos até a hora de ir dormir efetuamos várias decisões como levantar da cama; tomar café da manhã; o que comer no almoço; fazer exercícios; entrar em uma loja, algumas espontâneas, outras nem tanto.

No mundo dos algoritmos decisões também devem ser tomadas a todo o tempo para que o programa realize a tarefa para o qual ele foi desenvolvido. Para fazer esse controle de que caminho ele deve seguir, qual instrução será executada, executamos as estruturas de decisão. Nesse curso aprenderemos recursividade junto com **IF-ELSE** e o **SWITCH**.

### 2.1.1. Estrutura de decisão if.

A estrutura de decisão if é usada para verificar o teste lógico, que estará após a palavra reservada **if** e entre parênteses, e caso esse teste retorne verdadeiro ele executa um bloco de código. Observação: Se a instrução contida dentro do bloco de código for um só comando você não precisará envolvê-lo em chaves como é mostrado na figura abaixo, pois o compilador entenderá que a próxima instrução é a que deve ser executada, caso o teste retorne o valor verdadeiro.



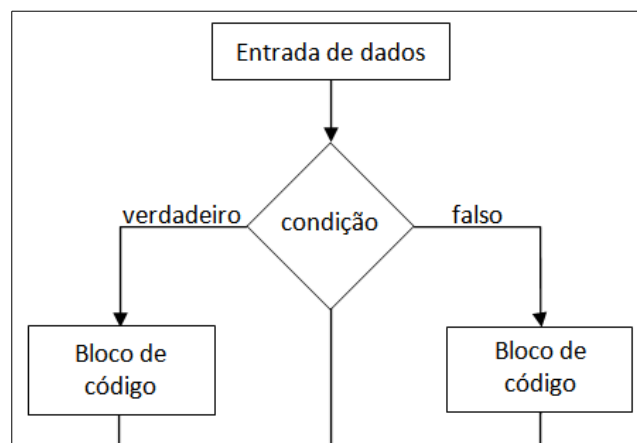
```
1  campanha = true;
2
3  if(campanha){
4      console.log('ding dong')
5  }
6
7
8
9
```



### 5.1.1. Estrutura de decisão if-else

O **if-else** é um avanço da estrutura anterior que tinha apenas um tratamento para o caso verdadeiro. Agora temos o **else** cujo bloco será executado se a condição do **if** não for satisfeita. O fluxograma do **if-else** é mostrado na próxima figura note que na estrutura anterior, caso o programa não entrasse no bloco do **if** ele seguia a execução normalmente agora, se isso acontece, temos um bloco de execução do **else** que será executado no lugar.

O **else** compartilha da mesma particularidade do **if**: caso tenha apenas um comando para executar, você não precisa colocar as chaves para envolver o bloco de código, pois o compilador entenderá que a próxima linha a ser executada pertence ao **else**, dessa maneira o código ficará bem mais limpo e simples para entender. Observação: você pode ter apenas o **if**, porém não pode ter um **else** desacompanhado.



```
int idade = 18;

if(idade>=18)

    System.out.println("Você pode tirar a habilitação.");

else

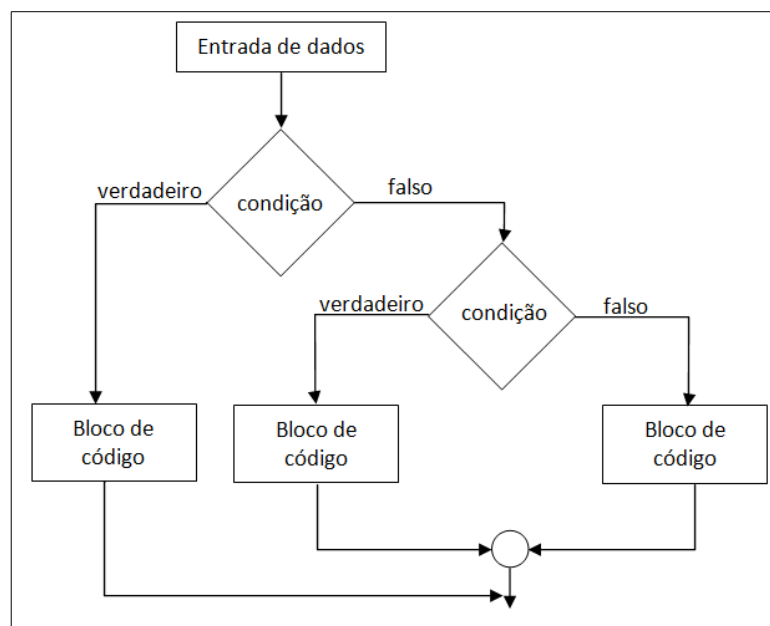
    System.out.println("Você não pode tirar a habilitação.");
```

### 55.1.2. Estrutura de decisão if-else-if

Quando temos mais de um teste lógico que não pode ser simplesmente resolvido com operadores lógicos, o que fazemos, outro if-else? Isso deixaria o código muito carregado, então para resolver esse problema de maneira eficaz e elegante temos os else-if. É uma estrutura que fica entre o if e o else fazendo com que possamos inserir mais um teste lógico e um outro bloco de comandos.

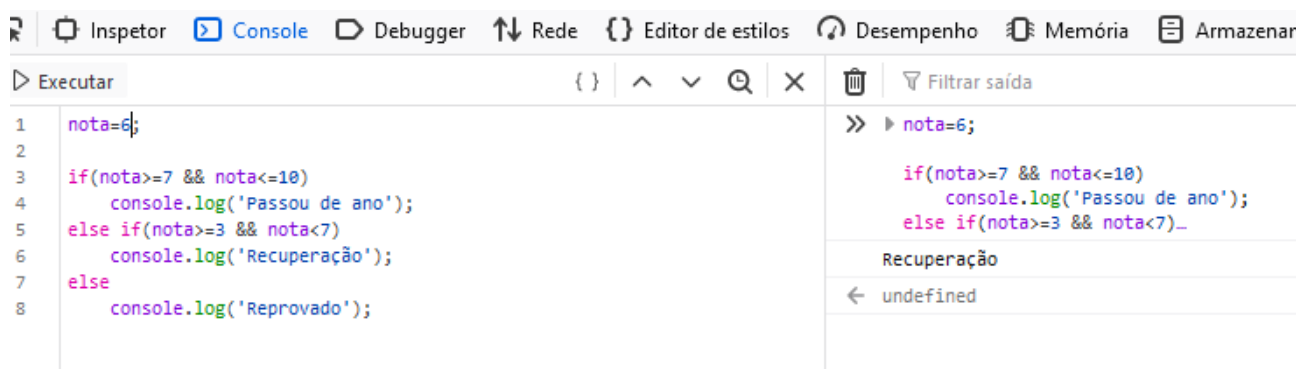
Também segue as mesmas propriedades dos seus antecessores quanto ao não precisar de chaves caso só aja um comando a executar e dentro do teste lógico pode conter operadores relacionais, lógicos e unários para a obtenção do resultado. Tenha muito cuidado de quando fizer um if-else-if de cobrir todas as possibilidades pretendidas para não haver erros de lógica e caso não seja nenhum deles.

O seu código funcionará perfeitamente só com if-else e com o else-if, porém é sempre uma boa prática de programação colocar um else para os casos gerais que não se enquadrem nas proposições lógicas. Na imagem teremos um efeito cascata todos os **ifs** serão executados um por um até que um deles retorne o valor verdadeiro ou caia no else, caso um deles retorne verdadeiro seu bloco de comandos é executado, logo após a execução da estrutura if é terminada.



- Vamos usar e abusar do bom e velho exemplo do aluno e sua nota. No código abaixo temos a declaração da nota, declarada inválida de propósito.
- O primeiro if verifica se a nota é maior do que 7 e menor e igual a 10 (necessário para restringir a nota máxima em 10), caso seja verdade o aluno passa por nota.

- O primeiro else-if analisa se a nota é menor do que 7 e maior ou igual a 3, para que o aluno possa ir para a recuperação.
- O segundo else-if verifica se a nota é menor do que 3 e maior ou igual a 0 (nota mínima) se for verdade então o aluno foi reprovado por nota.
- O else serve para fechar com chave de ouro. Como todas as opções anteriores cobrem todas as possibilidades de notas válidas (de 0 a 10), caso seja passado qualquer outro valor numérico fora desse intervalo o else passará uma mensagem de nota inválida.



```
1  nota=6;
2
3  if(nota>=7 && nota<=10)
4      console.log('Passou de ano');
5  else if(nota>=3 && nota<7)
6      console.log('Recuperação');
7  else
8      console.log('Reprovado');
```

Executar

» » nota=6;

if(nota>=7 && nota<=10)  
 console.log('Passou de ano');  
else if(nota>=3 && nota<7)\_

Recuperação

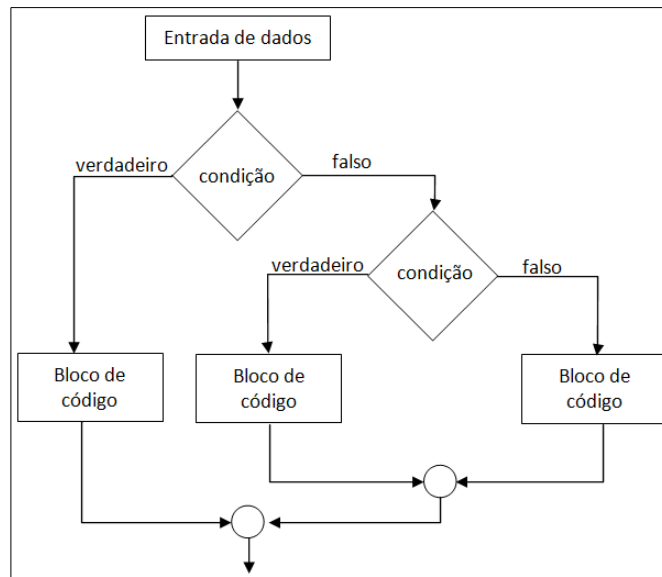
← undefined

### 5.1.3. Estrutura de decisão if-else aninhado.

Chamamos de if-else aninhado quando temos uma condição dentro de uma condição. O diagrama de fluxo do if-else aninhado é bem semelhante ao do if-else-if a diferença é que, ao invés, de termos mais um teste lógico principal temos um dentro de outro, logo ele só será executado se o código entrar naquele bloco de comandos.

Continuemos com o nosso exemplo super didático do nosso aluno em sua busca pela aprovação. Dessa vez o aluno teve uma nota 6, o que foi suficiente para fazer a recuperação. Dentro da recuperação temos uma análise da nota da recuperação, se tirar um 7 ele passará, caso contrário, ficará reprovado. Levando em consideração esse exemplo podemos observar que temos uma estrutura de decisão dentro da outra referente a ele ter a sua segunda nota analisada para só então o programa calcular qual será o resultado final.

Apesar dessas estruturas serem muito úteis, não é recomendado usar de maneira indiscriminada. Os riscos de uma má utilização do IF é a perda da linearidade do código. Quanto mais IFs tivermos será mais difícil de acompanhar o fluxo do programa, isso tornará mais árdua a tarefa de dar manutenção e até mesmo de ser lido por terceiros. Estamos falando aqui não de banir o uso dessa estrutura, mas sim do uso racional e consciente para que ela seja um aliado e não um empecilho.



#### 5.1.4. Estrutura de decisão SWITCH

Quando temos a análise de uma variável (byte, short, int, char, String) para a partir dela decidirmos que rumo nosso programa tomará podemos usar a estrutura de decisão **SWITCH**. Que é a mais indicada para a construção de menus e afins. O seu diagrama é bem simples. Temos a análise de uma variável e a comparação com cada um dos casos contidos nela. Se o programa encontrar compatibilidade com algum caso a análise termina e o mesmo será executado. Se não encontrar nenhuma ocorrência então a opção padrão (DEFAULT) será executada. Note que ao fim de cada caso usamos a palavra reservada **break** para que o switch seja encerrado, uma vez que na ausência dela todos os casos serão executados até chegar ao default.

```
let opcao = 1;
switch(opcao){

    case 1 : console.log("ensino médio"); break;

    case 2: console.log("graduação"); break;

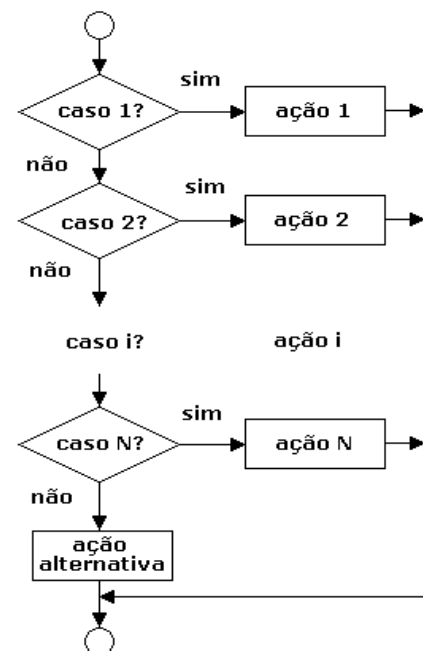
    case 3: console.log("mestrado"); break;

    case 4: console.log("doutorado"); break;

    default: console.log("escolaridade não definida.");

}
```

A observação que fizemos sobre o IF também é válida para o switch. Deve-se usá-lo da maneira correta, ou o código ficará difícil de manter e acompanhar. No próximo exemplo nós temos um código no qual a



variável de decisão é o carácter. Para definir qual é o tipo de cadastro na corrida e qual será o brinde que o participante receberá.

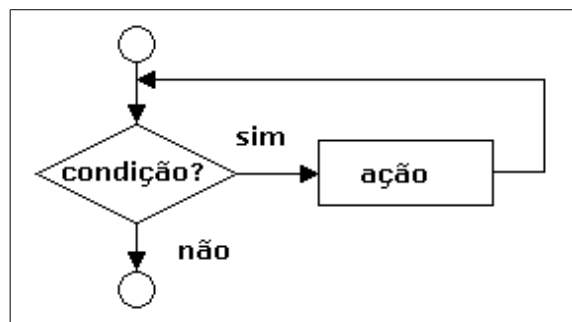
## 5.2. Estruturas de repetição.

As vezes no nosso algoritmo nos deparamos com situações onde precisamos repetir algum comando, várias vezes, repetidamente. Então você, como programador, tem duas escolhas: escrever o mesmo comando, de novo, de novo e de novo, ou usar uma estrutura que repetirá esse comando quantas vezes for preciso. Essas são as estruturas de repetição, também conhecidas como laços (loops). Elas vão executar um comando até que atinja o número de repetições desejado (por contador), ou até que receba algum sinal dizendo que não é mais necessário executar aquele comando (flag). Vamos começar vendo recursividade. Para entender um pouco mais das outras.

O último exemplo de recursividade faz o cálculo de um número elevado a um determinado expoente.

### 5.2.2. Estrutura de repetição WHILE.

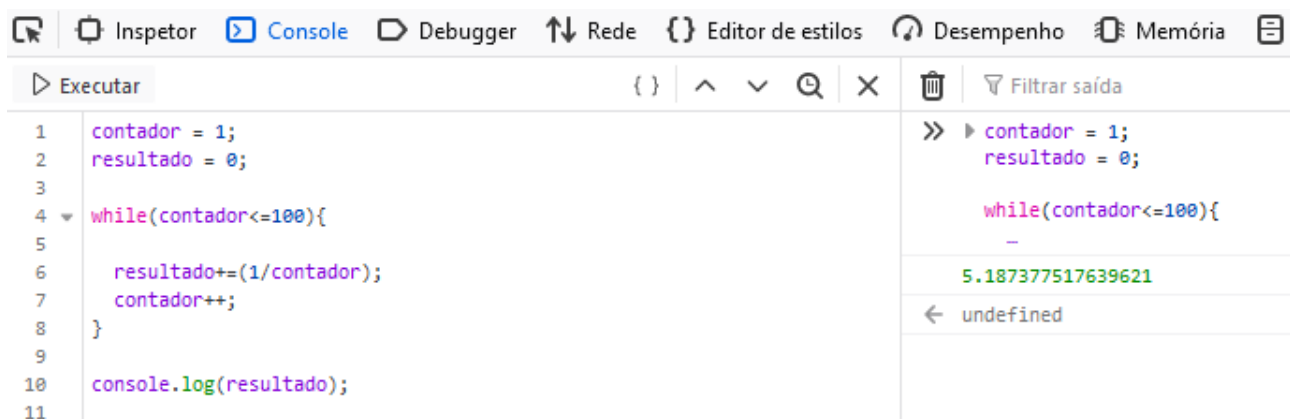
O diagrama de fluxo do while é bem simples. Ele vai primeiro analisa uma expressão, se ela for verdadeira ele executa o bloco de código contido nele e retorna para o início, fazendo isso até que o teste lógico retorne falso, quando isso acontecer o laço será encerrado. O maior cuidado que devemos ter com laços, é deixar bem claro o número de repetições ou a condição de término do mesmo, caso contrário nós teremos um loop infinito fazendo o programar executar indefinidamente.



Nosso primeiro exemplo de while é um programa que soma todas as frações de 1 até 1/100. Imagina o trabalho que daria para fazer soma por soma. Demoraria muito tempo para fazer uma expressão  $1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 \dots 1/100$ . Tal expressão seria impraticável por ser muito extensa e fácil de errar enquanto é digitada. Com as estruturas de repetição escrevemos a soma uma vez e repetimos cem vezes.

Antes de mais nada criamos duas variáveis o contador e o resultado. Depois na nossa estrutura **WHILE** dizemos para repetir o comando até o contador chegar no cem. Dentro do bloco de execução repare que usamos **resultado += ((double) 1/contador);** explicaremos cada um separadamente.

A expressão **resultado += x** é a forma mais simplificada de escrever **resultado = resultado + x**. Como contador é um inteiro então a divisão **1/ contador** tende a retornar um valor inteiro por isso colocamos o **(double)** esse recurso é chamado de **casting** e sua função é transformar esse resultado de inteiro para um número de ponto flutuante do tipo **double**. Logo após temos uma parte vital do programa que é incrementar o contador com **contador++** sem esse trecho do código o contador não mudaria de valor e entraríamos em um loop infinito. Por último temos a impressão do resultado.

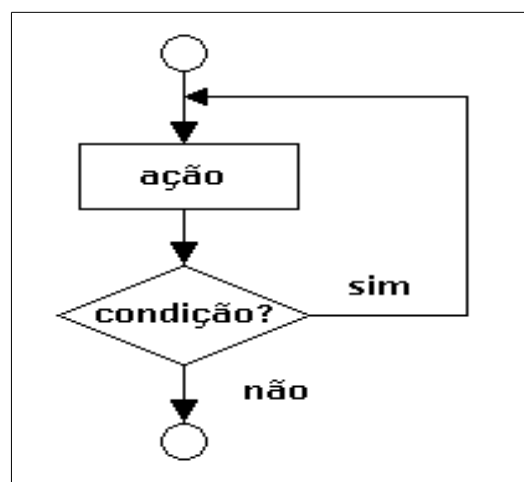


```
1 contador = 1;
2 resultado = 0;
3
4 while(contador<=100){
5
6     resultado+=(1/contador);
7     contador++;
8 }
9
10 console.log(resultado);
11
```

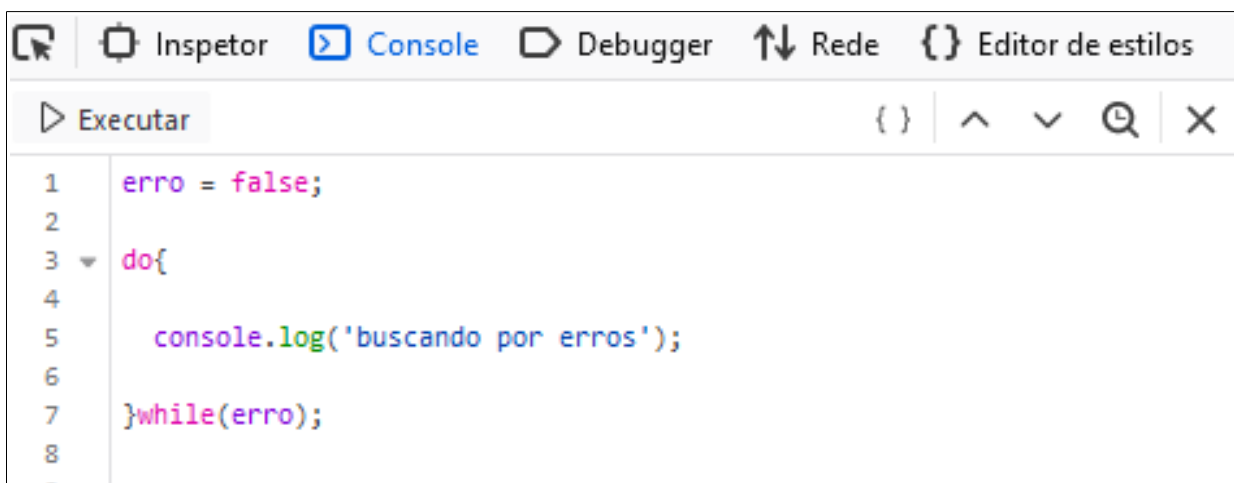
The console output shows the final value of **resultado** as **5.187377517639621**.

### 5.2.3. Estrutura de repetição DO WHILE.

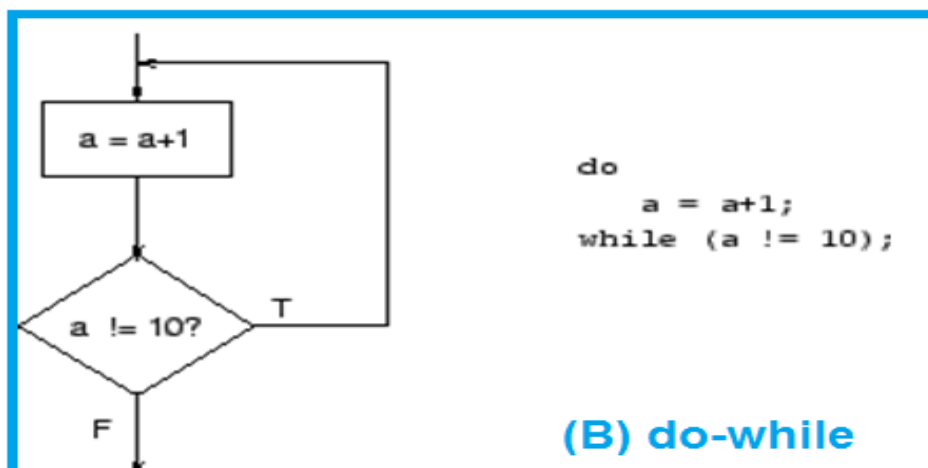
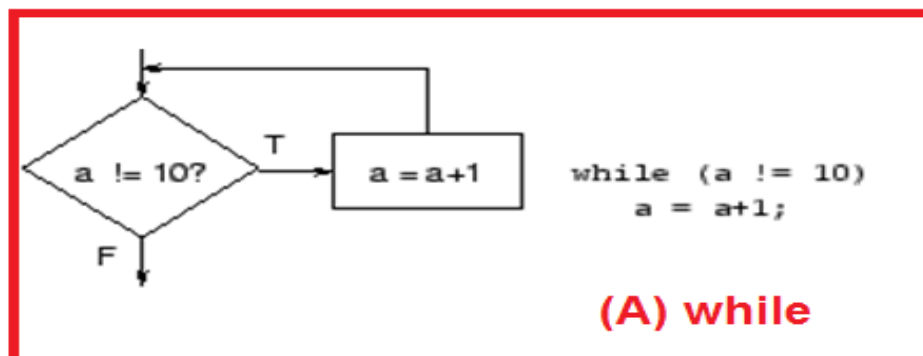
A expressão **do while** é semelhante em todos os aspectos ao **while** exceto por um detalhe. O laço vai executar, pelo menos uma vez, suas instruções, independente do valor do seu teste lógico e só depois faz a verificação para saber se o laço continuará sendo executado, enquanto o **while** só vai ser executado se sua condição for verdadeira. Qual a aplicabilidade para tal estrutura? Imagine um sistema que ao ser ligado, mesmo que não tenha nenhum problema aparente, faça uma checagem de segurança preventiva. Logo abaixo temos o diagrama de fluxo **do do while**.



O nosso exemplo com o do while é meramente figurativo. Ele não faz escaneamento nenhum, é só para demonstrar que mesmo que a flag erro essa falsa, e a condição verificando se ela é verdadeira, nosso laço vai executar pelo menos uma vez. Depois da primeira execução a estrutura vai identificar a condição como falsa e vai parar sua execução. Também mostramos os dois diagramas de fluxo para vocês verem de maneira mais clara a diferença entre os dois.



```
1 erro = false;
2
3 do{
4     console.log('buscando por erros');
5 }while(erro);
```

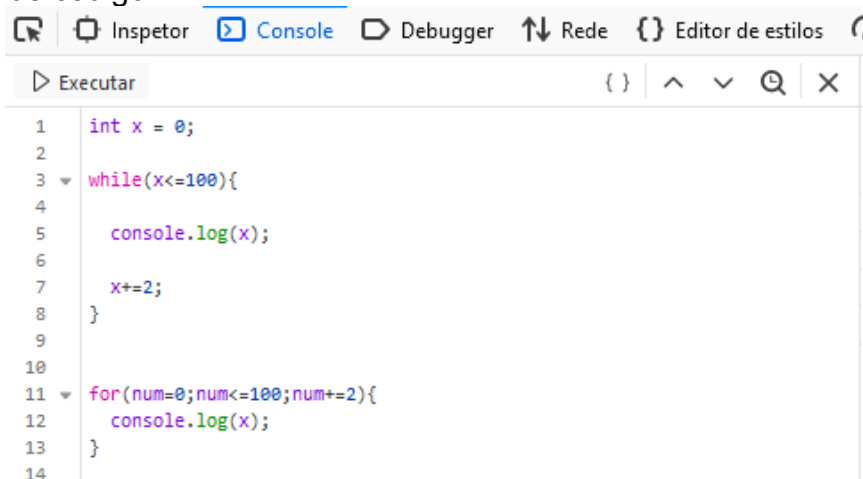


#### 5.2.4. Estrutura de repetição FOR.

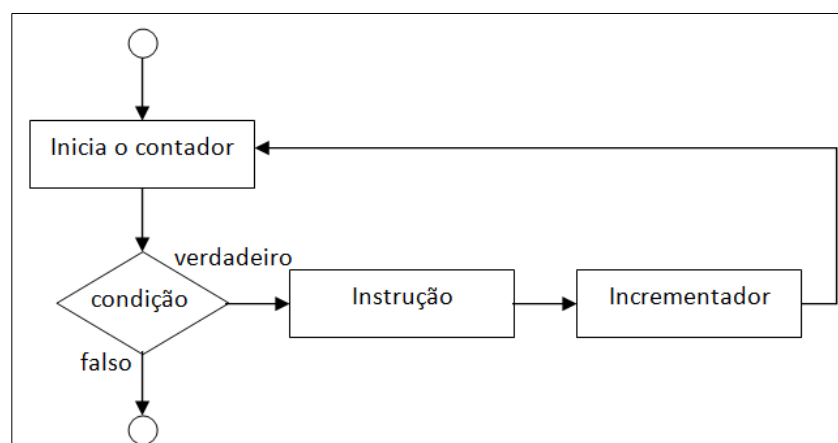
Enquanto no **while** precisávamos ficar declarando dentro do corpo a incrementação do contador, ou mudança da flag, tínhamos que declarar as variáveis de controle e só cabia a estrutura o teste lógico para o controle do laço. No **for** já temos todos esses dados declarados em apenas uma linha, ou seja, bem mais fácil de acompanhar as condições sobre as quais o laço está sujeito.

Essa organização do for faz dele o candidato perfeito para trabalharmos com Arrays, tanto é que foi criado o `forEach` só para essa finalidade.

Antes de falarmos mais sobre o for. Observe apenas a próxima figura para ver a diferença de verbosidade entre ele e o while e compare os dois quanto a clareza e simplicidade de código.

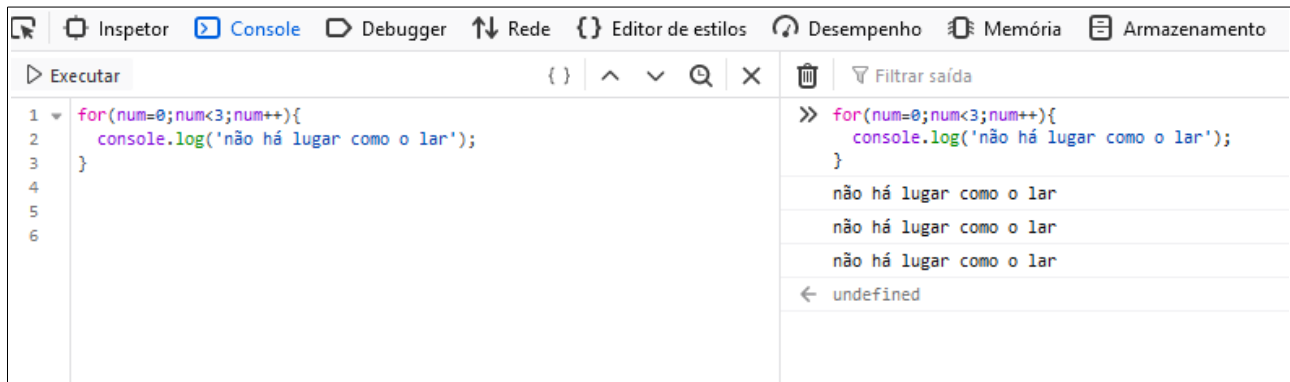


Os dois códigos têm a mesma função: imprimir os números pares de 0 a 100. Como podem ver o segundo consegue entregar em apenas três linhas de código o que o primeiro precisou de cinco. Parece pouca coisa, mas em um código cada linha a menos faz diferença na hora de dar manutenção ou ler o mesmo. Verbosidade em excesso apenas deixa o seu código menos funcional. Agora que vimos a diferença, vamos conhecer mais a fundo o **FOR**.





Como podemos ver na figura abaixo na declaração do for já temos a inicialização da variável, a condição de parada e o incremento, tudo junto e logo após temos o corpo onde ficarão os comandos a serem executados durante o laço. Com num inicializado em 0, executando enquanto ele for menor do que 3 e incrementando de 1 a cada volta, teremos três execuções.



O for melhorado foi implementado a partir do Java 5 para servir como facilitador da exploração de Arrays, por enquanto veremos apenas essa versão. Com o lançamento do Java 8 tivemos as lambdas expressions para adicionar mais algumas funcionalidades como o map e filter, veremos mais durante o capítulo de Arrays. Note que esse for melhorado basicamente pega um array e o quebra em várias partes menores que você especifica no cabeçalho do for e que pode tratar como quiser.

### 5.3.5 Exercícios propostos.

#### 1) Responda com V para verdadeiro e F para falso.

- ☐ O if é uma estrutura de repetição.
- ☐ O while só executa seu bloco interno caso a condição nele seja verdadeira.
- ☐ O do-while irá executar pelo menos uma vez independente da sua condição ser verdadeira ou falsa.
- ☐ Não existe a necessidade do break em uma estrutura switch case.
- ☐ No cabeçalho do for podemos inicializar a variável, ter a condição do laço e o incremento em um único lugar.

#### 2) Faça um laço que imprima os primeiros 10 números múltiplos de 8.

**3) É a estrutura de decisão que analisa uma determinada variável e de acordo com o seu resultado executa algum bloco de código específico.**

- a) for
- b) if else
- c) while
- d) do while
- e) switch

**4) O que acontece caso um laço não possa alcançar a sua condição de parada?**

- a) o programa não roda.
- b) laço infinito.
- c) a VM desliga automaticamente a aplicação.
- d) acontece um erro.
- e) nada acontece.

**5) Supondo que a estrutura switch case esteja implementada de maneira correta e completa. Ao analisar uma variável o switch não encontrou nenhuma combinação com os seus cases, nesse caso será executado o:**

- a) break
- b) default
- c) o último case.
- d) N.R.A.

**6) Qual dessas estruturas é a mais indicada para o manejo de arrays?**

- a) while
- b) do while
- c) if else
- d) switch
- e) for

**7) Faça um programa que resolva uma equação do 2º grau.**

**8) Faça um programa que imprima os 10 primeiros números da sequência de Fibonacci**

**9) Desenvolva um programa que receba um número e diga se ele é um quadrado perfeito, ou não.**

**10) Com os seus conhecimentos adquiridos até agora faça um programa que receba dois números e verifique se eles são primos entre si.**