

The SENAIE logo is centered in a red rectangular box. The word "SENAIE" is written in white, bold, sans-serif capital letters. On each side of the word, there are three horizontal bars: two short bars on top and one longer bar below them, creating a stylized "E" shape on both sides.

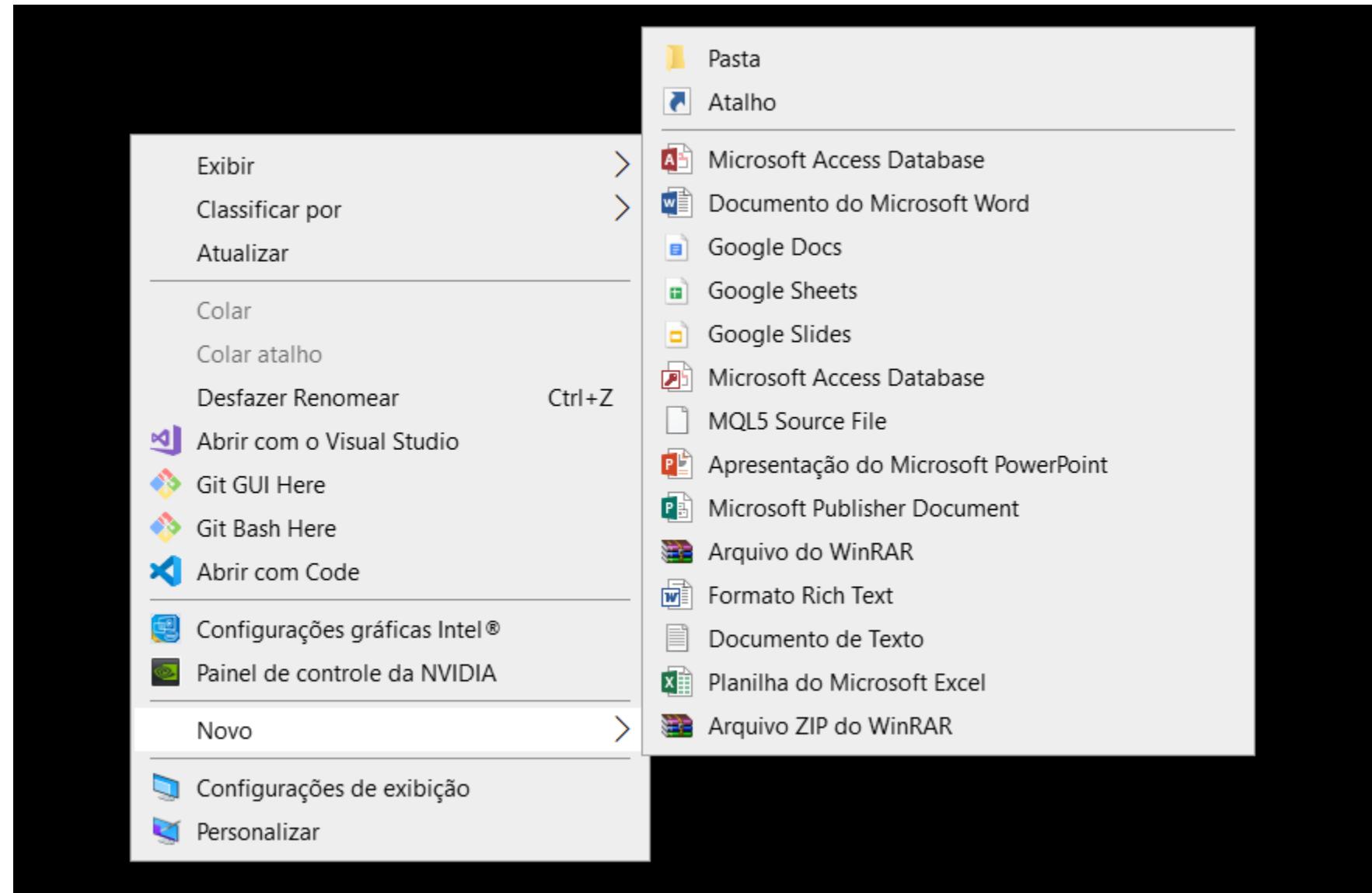
SENAIE

# Fundamentos de Informática



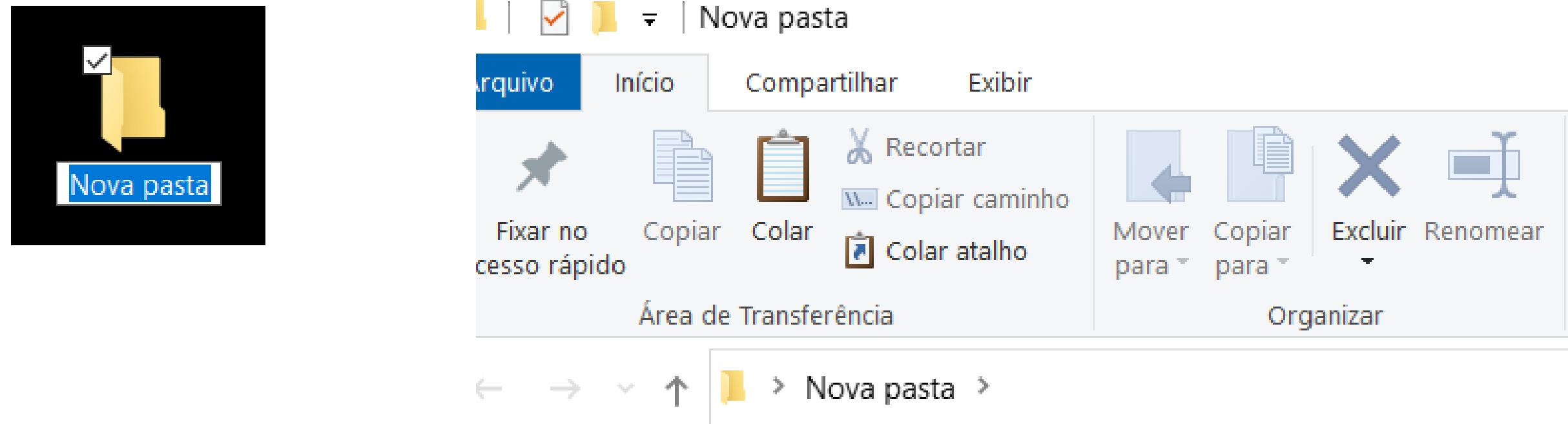
**SENAI**

# Manipulação de arquivos



- 
- Criar Arquivos
  - Deletar Arquivos
  - Mover Arquivos
  - Renomear Arquivos

# Manipulação de pastas



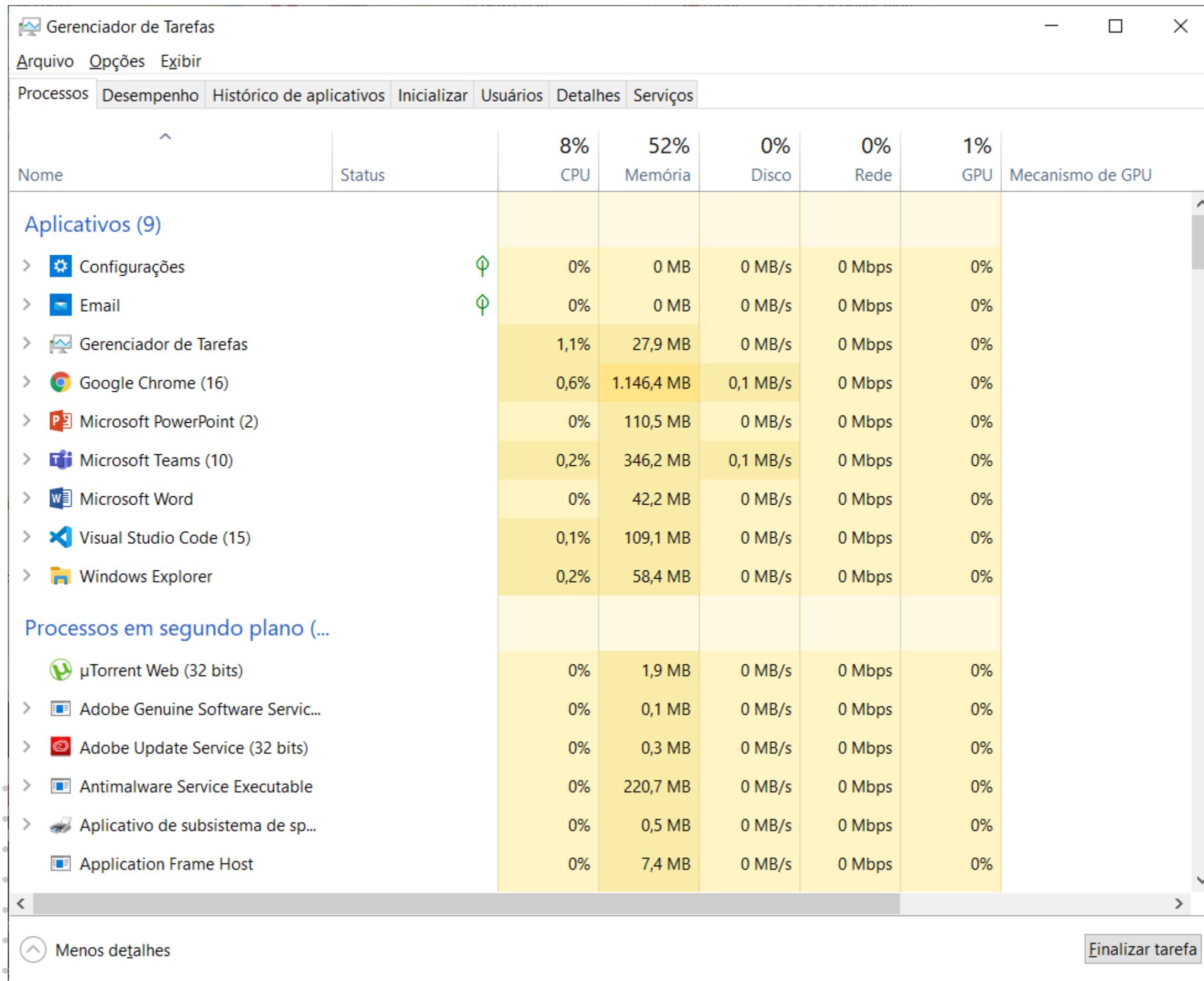
- 
- Criar Pastas
  - Deletar Pastas
  - Mover Pastas
  - Renomear Pastas

# Tipos de arquivos



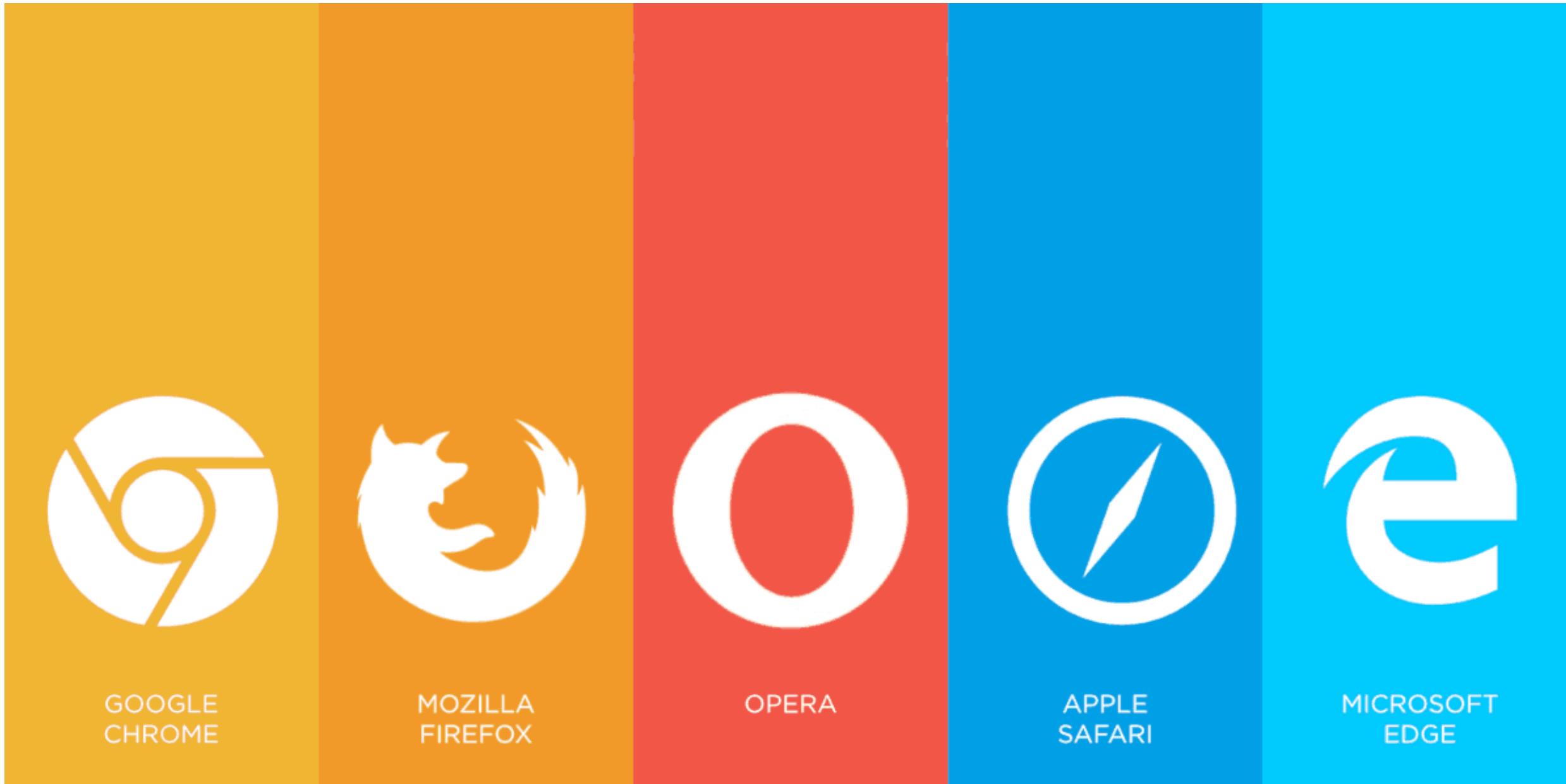
- 
- Tipos de Arquivo de Imagem
  - Tipos de Arquivo de Áudio
  - Tipos de Arquivo de Vídeo
  - Tipos de Arquivo de Texto

# Informações do Sistema Operacional



- Tipo do Sistema
- Hardware
- Processos
- Tarefas

# Navegadores



- 
- O que são Navegadores
  - Diferenças entre os navegadores

# Downloads



- 
- Como baixar
  - Onde Baixar
  - Taxa de Transferência
  - Pasta de Downloads

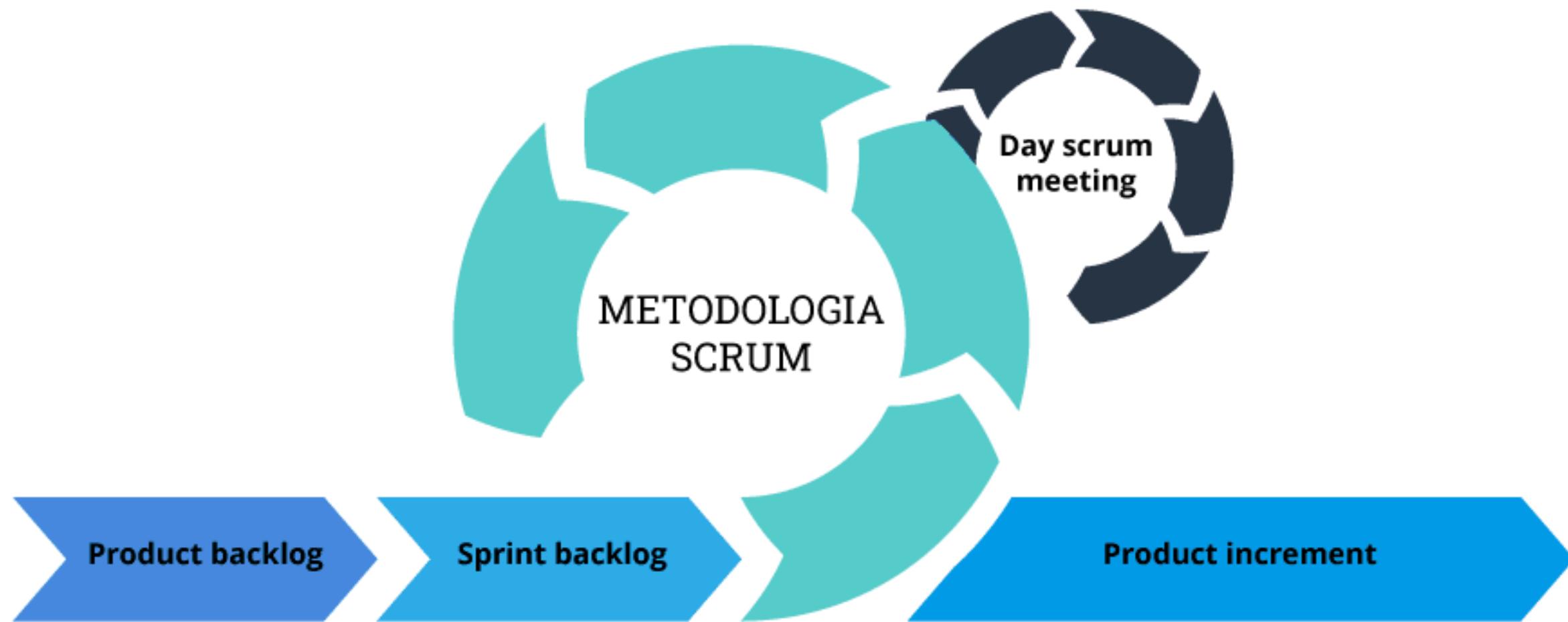


# Metodologias Ágeis

**SENAI**

# Scrum

É um framework de gerenciamento de projetos muito utilizado em equipes de desenvolvimento de software, porém pode ser aplicado a qualquer tipo de trabalho em equipe



- 
- Sprint
  - Product Backlog
  - Sprint Planning
  - Sprint BackLog
  - Daily Meeting
  - Sprint Review
  - Sprint Retrospective
  - Scrum Master
  - Product Owner

# Kanban

Criado pela Toyota na década de 60 e usado até hoje por empresas, o Kanban consiste em organizar as tarefas em colunas e movimenta-las de acordo com seu progresso.

*A executar*



*Em curso*



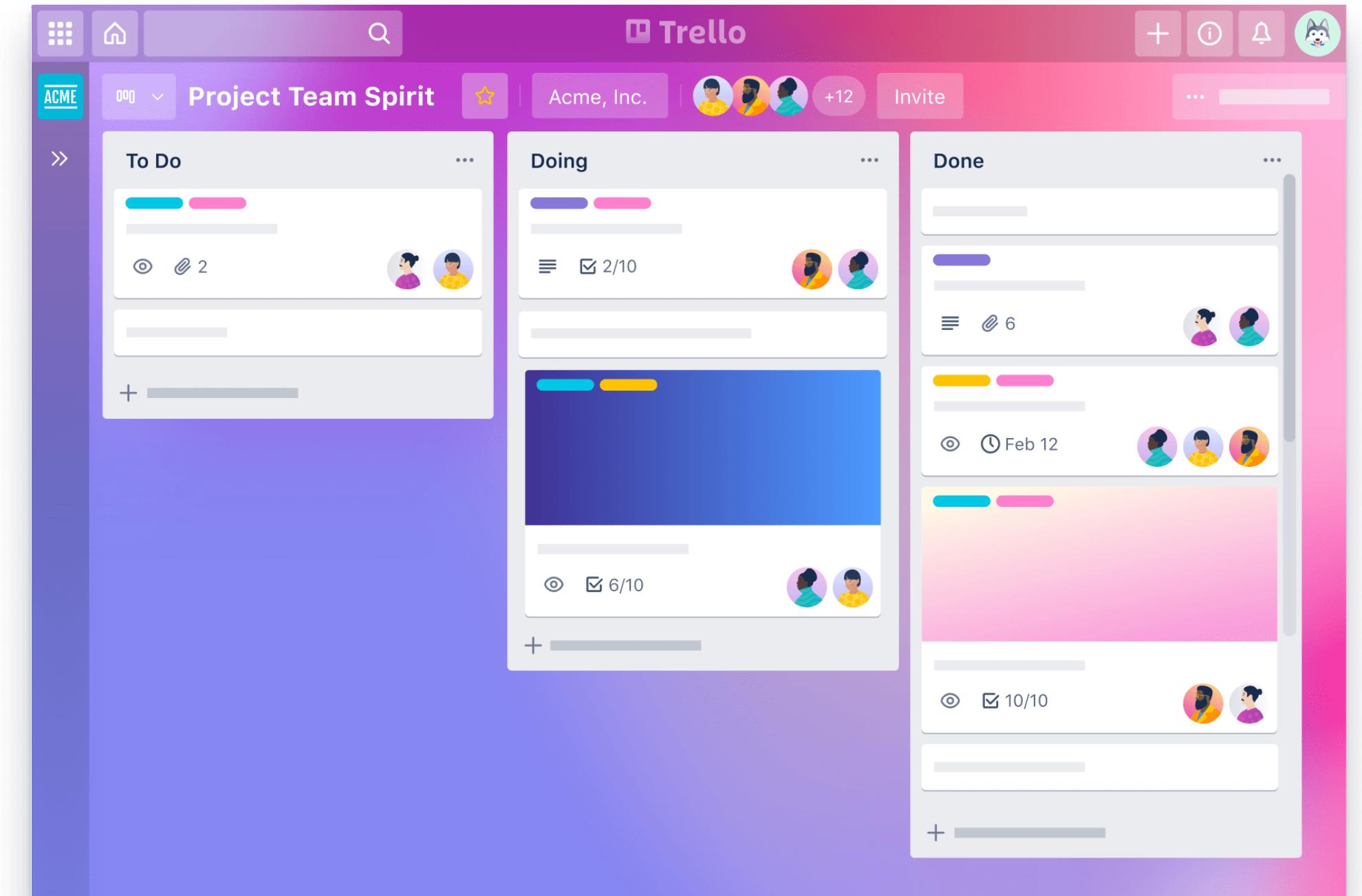
*Executados*



# Trello

Aplicativo de gerenciamento de projeto que utiliza como modelo o Kanban, nele é possível criar cartões e adicionar elementos como prazos, responsável, checklists, etiquetas, dentre outros e movimenta-los pelas colunas conforme seu status.

<https://trello.com/pt-BR>



# Trello + Scrum + Kanban



Screenshot of a Trello board titled "SENAI" showing a hybrid Scrum/Kanban workflow.

The board has the following sections:

- Links e Arquivos**: Contains a "Link Protótipo Figma" card with a diagram and a "Diagrama" section.
- Product Backlog**: Contains cards for:
  - Testes Sprint 3: Criar Teste Unitário Automatizado (Due 22 de abr, 0/5)
  - Testes Sprint 3: Criar Teste de Integração (Due 22 de abr, 0/5)
- Sprint Backlog (À Fazer)**: Contains cards for:
  - BackEnd Sprint 2: Criar Método de Login (Due 15 de abr, 0/3)
  - BackEnd Sprint 2: Criar Método Cadastro de Livros (Due 15 de abr, 0/3)
  - BackEnd Sprint 2: Criar Método Listar Livros (Due 15 de abr, 0/3)
- Fazendo**: Contains cards for:
  - BackEnd Sprint 2: Criar Método de Cadastrar Usuário (Due 15 de abr, 0/3)
- Testando**: Contains cards for:
  - BackEnd Sprint 2: Criar Estrutura de Arquivos (Due 15 de abr, 0/4)
- Feito**: Contains cards for:
  - FrontEnd Sprint 1: Fazer HTML e CSS Login (Due 6 de abr, 0/2)
  - FrontEnd Sprint 1: Fazer HTML e CSS Cadastro de Usuário (Due 6 de abr, 0/2)
  - FrontEnd Sprint 1: Criar View Cadastro de Usuário (Due 1 de abr, 0/3)
  - FrontEnd Sprint 1: Criar View Login (Due 1 de abr, 0/3)

General board controls include: Modelo, SENAI, Público, Compartilhar, Power-Ups, Automação, Filtro, and Mostrar Menu.

# Versionamento ( Git e Github )



**SENAI**



É um sistema de controle de versões usado principalmente no desenvolvimento de software, mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo local.



GitHub é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git.



- 
- git init
  - git add
  - git commit
  - git status
  - git log
  - git remote
  - git push
  - git pull



# Design De Interface Básico



**SENAI**



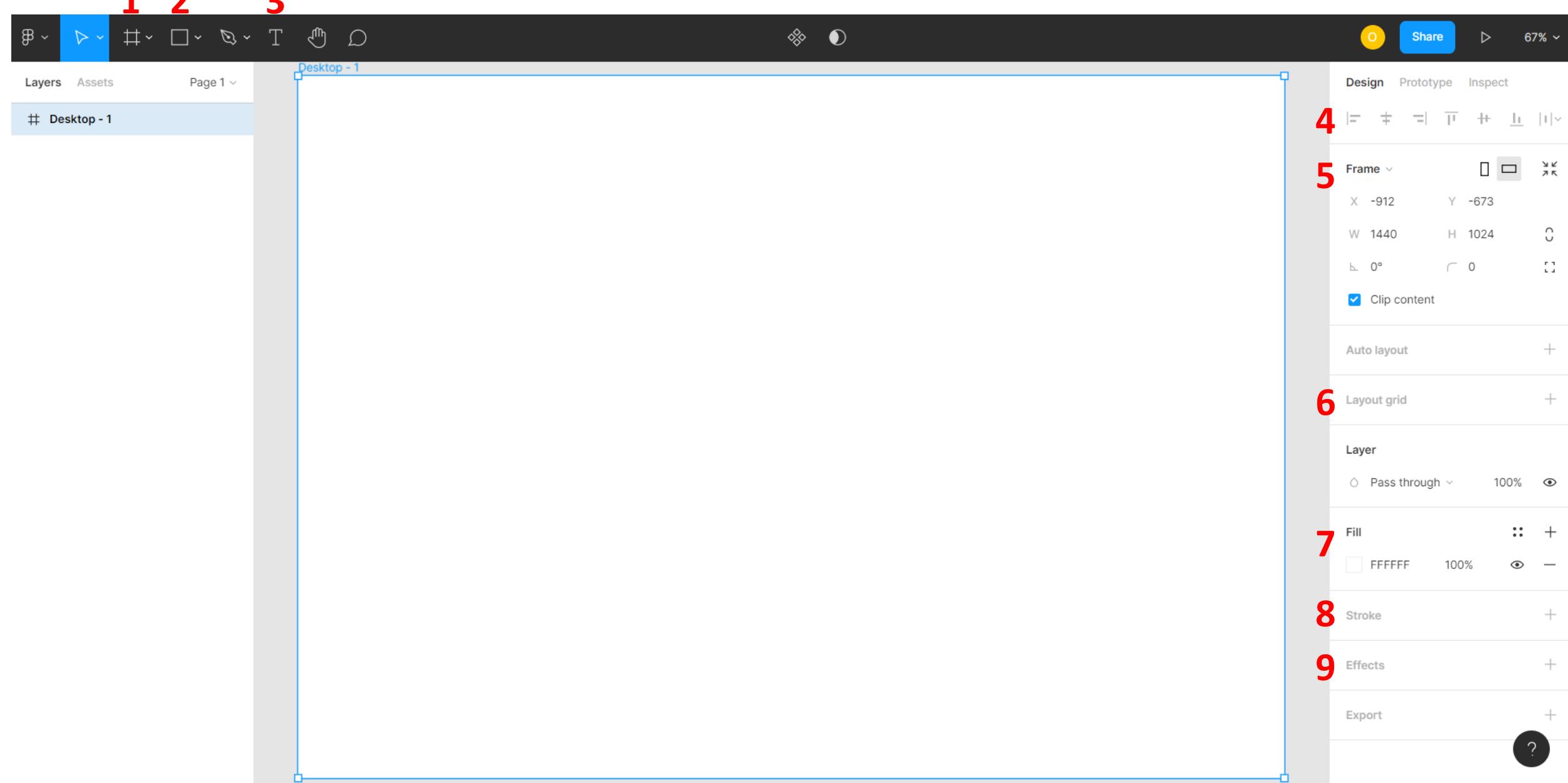
É um editor gráfico focado no desenvolvimento de sistemas de design gráfico, prototipagem de interface gráfica de usuário e desenvolvimento de UI/UX com opções online e offline.

<https://www.figma.com/>





# Figma



- 
- 1 – Frame
  - 2 – Formas
  - 3 – Texto
  - 4 – Alinhamento
  - 5 – Dimensões
  - 6 - Grid
  - 7 – Preenchimento
  - 8 – Contorno
  - 9 - Efeitos



# Wireframe

Um Wireframe é uma ilustração semelhante ao layout e tem como principal objetivo dispor o posicionamento dos elementos de uma tela.



# Tamanhos de Telas



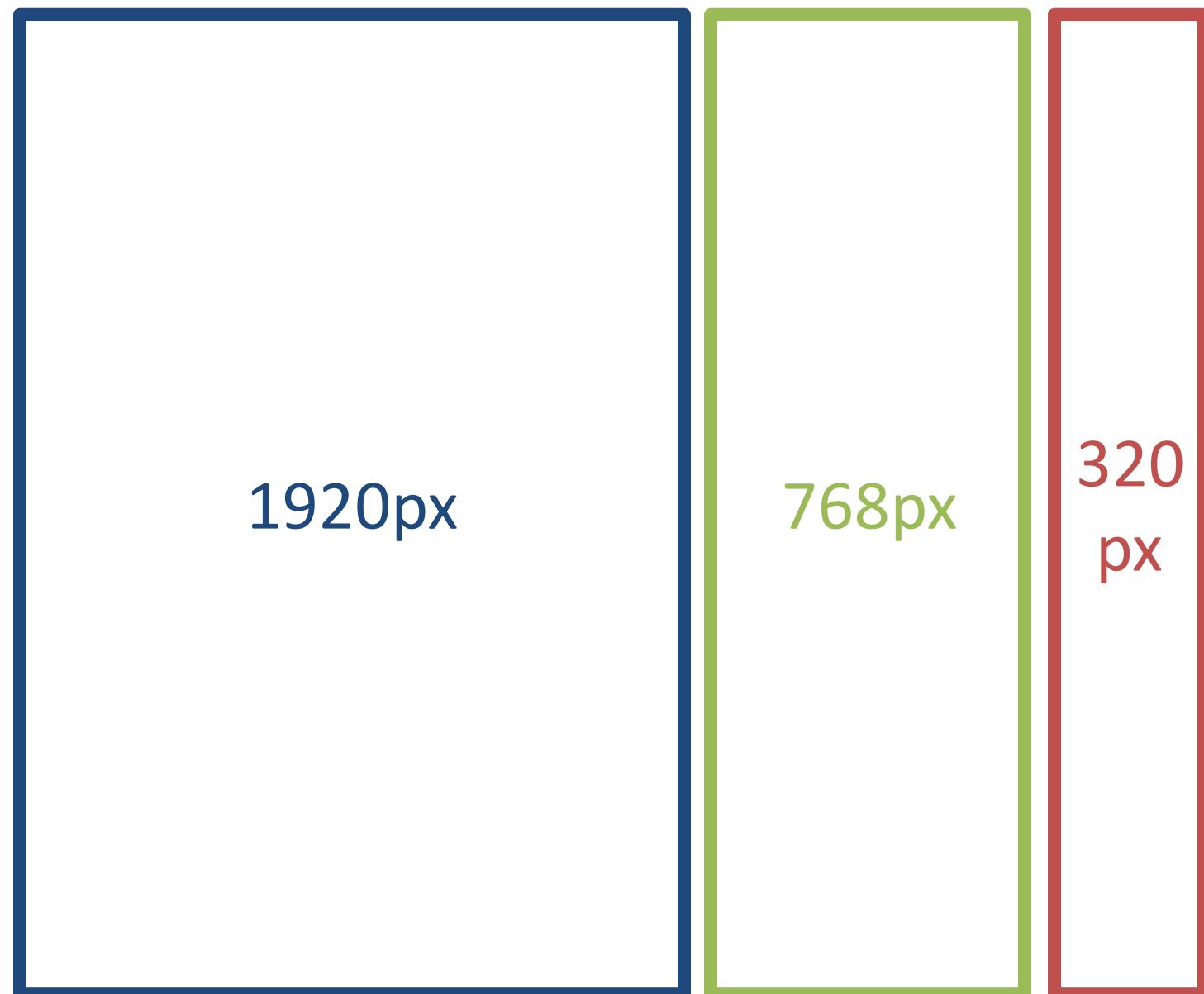
FULL HD - 1920 - 1080P

4K - 3840 - 2160px

Prancheta para layout – desktop / 1920px

Prancheta para layout – tablet / 768px

Prancheta para layout – mobile / 320px



# Layout de Alta fidelidade



Layout de Alta fidelidade é uma representação visual, com detalhes de como o produto final deve ficar, já o Protótipo de Alta fidelidade acrescenta também funcionalidades à interface.

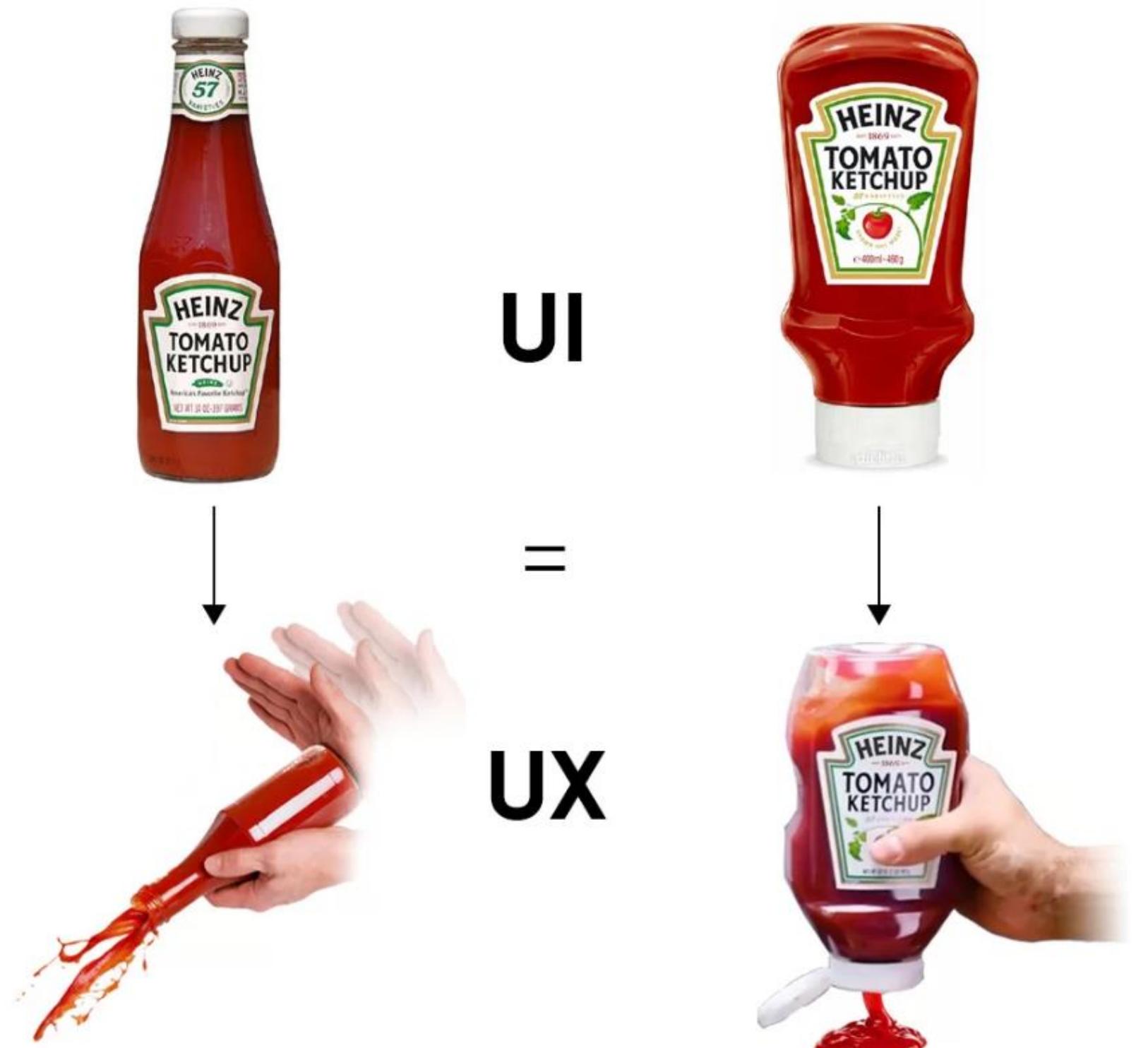
A high-fidelity layout for a website about a 2022 software development course. The header features the SENAI logo and navigation links for INÍCIO, SOBRE, BEM-VINDOS, and DIRETÓRIO. The main title "DESENVOLVIMENTO DE SISTEMAS 2022" is displayed in large white text over a background image of hands working on a computer keyboard. Below the title is the text "SENAI INFORMÁTICA" and a small computer mouse icon. A section titled "SOBRE OS PROJETOS" shows a thumbnail of two people in a workshop. To the right, there's a sidebar with the text "PARTICIPANTES" and "MAIS DE 80 ALUNOS ORGANIZADOS EM 16 GRUPOS". A small paragraph of placeholder text follows.

# Layout de Alta fidelidade



UI – ( User Interface ou Interface do Usuário ), o objetivo da UI é tornar a interação do usuário com o sistema o mais simples possível levando em consideração seu público alvo.

UX - ( User Experience ou Experiencia do Usuário ) é o conjunto de elementos relativos a experiencia do usuário em relação ao sistema, como o usuário utiliza o sistema.



# Teoria das Cores



Quando e quais cores utilizar?

## GUIA EMOCIONAL DAS CORES

OTIMISMO	<i>clareza calor</i>
AMIGÁVEL	<i>alegria confiança</i>
EXCITAÇÃO	<i>juventude coragem</i>
CRIATIVO	<i>imaginativo sabedoria</i>
CONFIANÇA	<i>seguro força</i>
PACÍFICO	<i>crescimento saúde</i>
EQUILÍBRIO	<i>neutro calmo</i>



# Tipografia

Quando e quais fontes utilizar?

- **Fonte Serifada**

Com traços alongados em suas extremidades podem ser facilmente encontradas em textos longos por facilitar sua leitura.

- **Fonte não Serifada**

Fonte mais simplificada, sem alongamentos nas extremidades mais utilizadas em textos curtos, simples e objetivos.

- **Fontes Caligráficas**

Fonte que se aproxima da escrita cursiva, partindo da ideia de liberdade e singularidade mais utilizadas em textos para causar impacto visual.

- **Fontes Decorativas**

Utilizadas comumente em logotipos e não empregadas em textos para leitura por terem uma característica mais visual.



Contemporâneo

ADELLE

Tradicional

BEMBO

Sofisticado

MONTAGUE SCRIPT

Simples

AKAGI PRO

Corporativo

DIN NEXT

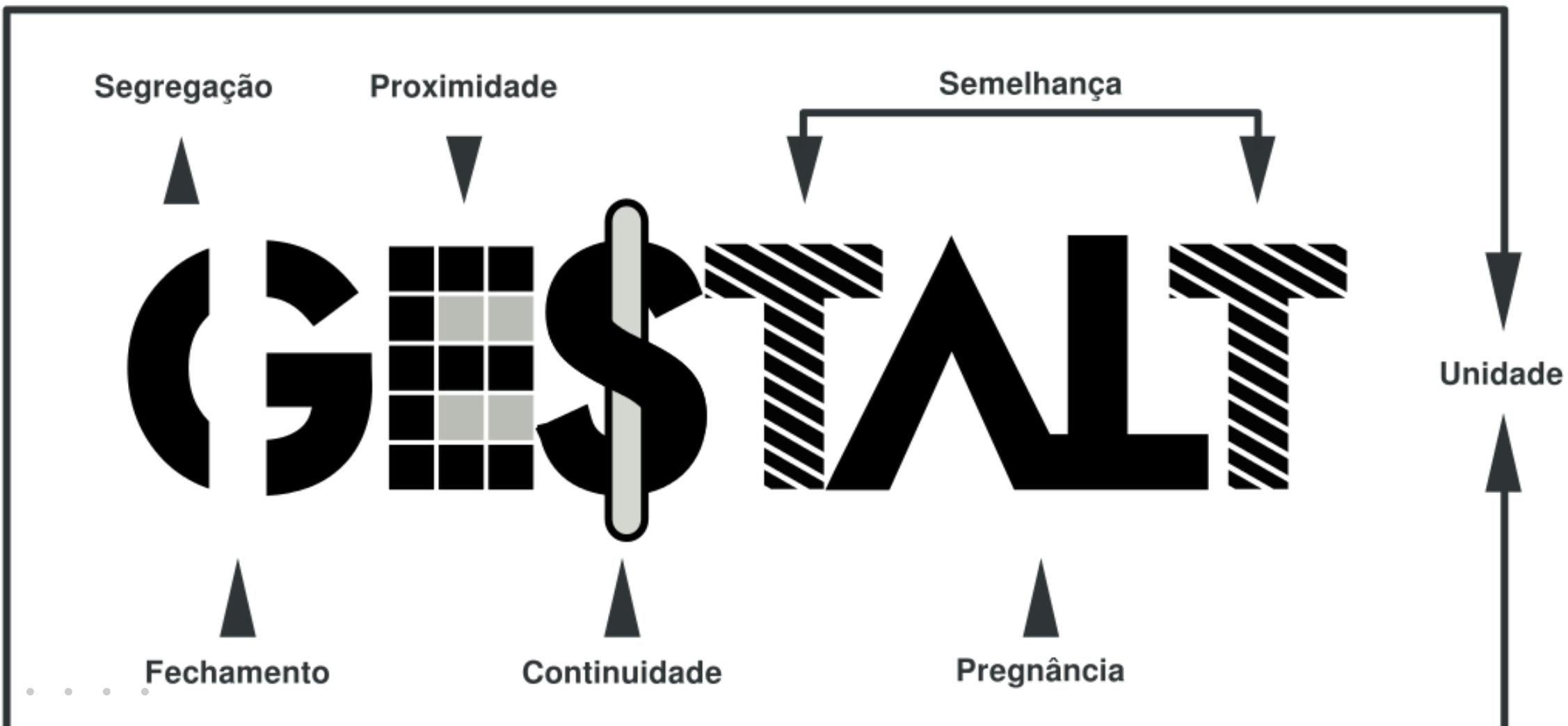
Dinâmico

WONDER BRUSH



# Teoria da Forma - Gestalt

A teoria de Gestalt se baseia na ideia de que para se compreender as partes devemos antes compreender o todo.



- 
- Segregação
  - Proximidade
  - Semelhança
  - Fechamento
  - Continuidade
  - Pregnância
  - Unidade

# Estrutura de Páginas Web



**SENAI**

# Introdução às Tecnologias



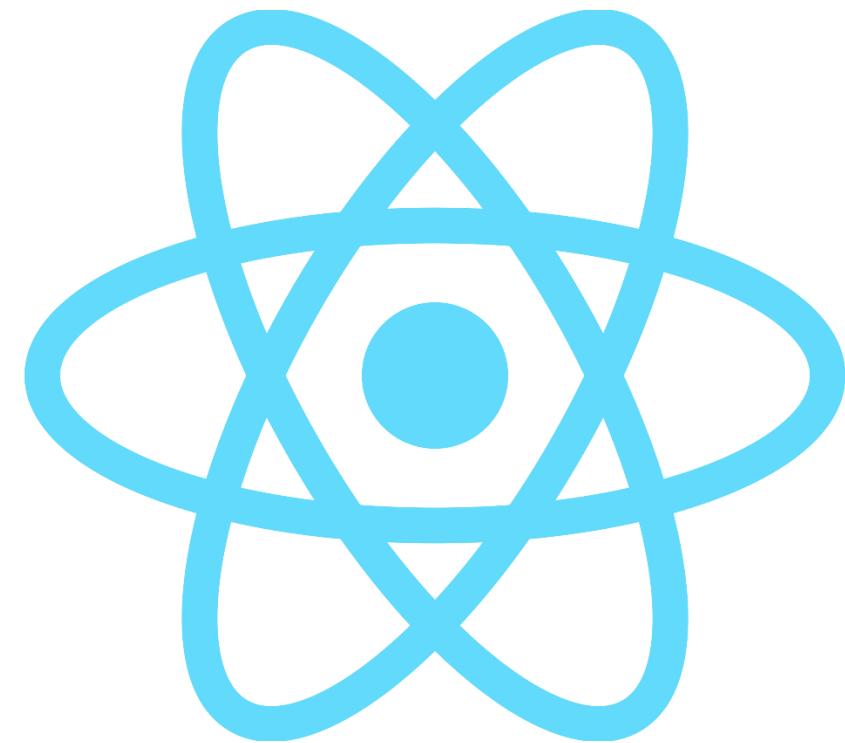
**HTML**



**CSS**



**JS**



# Introdução às Tecnologias



**HTML** – Linguagem de estruturação de páginas web, define como será a estrutura e o conteúdo da página.

**CSS** – Linguagem de estilização de páginas web, define as configurações de aparência dos elementos da página.

**JavaScript** – Linguagem de programação responsável por implementar funcionalidades em uma página e torna-la mais dinâmica.

# Introdução às Tecnologias



**ReactJS** – Biblioteca para construção de interfaces de usuário em páginas Web.

**React Native** – Biblioteca para construção de aplicativos Híbridos (IOS e Android).



# Introdução às Tecnologias



## Ferramentas



# Introdução às Tecnologias



## Ferramentas

**Visual Studio Code** – Editor de código capaz de trabalhar com várias linguagens.

**NodeJS** – Interpretador de JavaScript, necessário para executar os códigos escritos em JS fora do navegador

# Introdução às Tecnologias



## Links

### Visual Studio Code

<https://code.visualstudio.com/download>

### NodeJS

<https://nodejs.org/en/>



# Estrutura de Páginas Web



**SENAI**

# Estrutura de Páginas Web

**HTML** – O HTML é responsável por definir a estrutura de uma página, para isso utiliza **tags** para definir os **elementos**.



# Estrutura de Páginas Web



Uma página HTML possui dois elementos essenciais, o **head**, que é o elemento que reúne informações e configurações da página, e o **body**, que marca todo o conteúdo de uma página.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8
9
10 </body>
11 </html>
```

# Estrutura de Páginas Web



## Tags mais comuns:

`<h1>` - Tag de títulos e subtítulos, vai do `<h1>` até o `<h6>`

`<p>` - Tag de parágrafos

`<a src="https://www.google.com">Texto do link</a>` - Tag de link

``  
Tag de imagem



# Estrutura de Páginas Web



Tabelas no HTML:

```
<table> <!-- Tag de Tabela -->
  <thead> <!-- Tag do cabeçalho da tabela -->
    <th></th> <!-- Tags dos itens do cabeçalho -->
    <th></th>
    <th></th>
  </thead>
  <tbody> <!-- Tag do corpo da tabela -->
    <td></td> <!-- Tag dos itens do corpo da tabela -->
    <td></td>
    <td></td>
  </tbody>
</table>
```



# Estrutura de Páginas Web



Listas não ordenadas no HTML:

```
<ul> <!-- Tag de Lista não ordenada -->
    <li>Item 1</li> <!-- Tag de item da lista -->
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
</ul>
```



## Semântica no HTML5

A principal evolução que o HTML5 trouxe em relação à suas versões anteriores, foi o maior peso semântico dos elementos, ou seja, existem mais tags com responsabilidades específicas.

**SEMÂNTICA == SIGNIFICADO**



# Estrutura de Páginas Web



Isso melhorou a estrutura das páginas web, que eram construídas com mais tags genéricas, agora temos significado em cada seção da página.

## Exemplos:

`<header>` - Tag para cabeçalhos

`<main>` - Tag para o conteúdo principal da página

`<footer>` - Tag para o rodapé

`<section>` - Marca uma seção de uma página

# Estrutura de Páginas Web



Existem centenas de outras tags que podemos utilizar nas nossas páginas, como podemos verificar em algumas referências:

W3Schools

<https://www.w3schools.com/tags/default.asp>

Mozilla Developer Network - MDN

<https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element> -



# Estilização de Páginas Web



**SENAI**

# Estilização de Páginas Web



**CSS** – O CSS é a linguagem responsável por definir as configurações de **aparência** dos elementos html.

Como por exemplo, definir tamanho dos elementos, cores das fontes, cores de fundo e também o posicionamento dos itens de uma página.



# Estilização de Páginas Web



Para aplicar as configurações de estilização, o CSS precisa indicar quais elementos serão alterados, para isso utilizamos os **seletores**, para indicar o que será alterado, e como será alterado usamos os pares de **propriedade** e **valor**.



# Estilização de Páginas Web



Temos várias formas de selecionar os elementos, vários tipos de seletores, dentre os principais estão:

**Por nome de tag:** Seleciona todos os elementos que possuem esse nome de tag.

A screenshot of a code editor showing a snippet of CSS. The code is: 'a { color: red; }'. The 'a' selector is in orange, the opening brace '{' is in green, the 'color' property is in blue, the colon ':' is in light blue, the red square color box is in red, and the closing brace '}' is in green. The code is displayed on a dark background.

Ex.: Todos os elementos <a> terão como cor de fonte o **vermelho**

# Estilização de Páginas Web



**Por nome de classe(class):** Seleciona todos os elementos que possuem o esse valor no atributo class.

```
<a href="#" class="link">Login</a>
```

```
.link {  
    color: darkblue;  
    text-decoration: none;  
}
```

Ex.: Todos os elementos com a classe **link** terão como cor de fonte o **azul escuro**

# Estilização de Páginas Web



**Pelo id do elemento:** Seleciona o elemento que possui esse valor no atributo **id**.

```
<input type="email" name="email" id="campo-email">
```

```
#campo-email {  
    height: 30px;  
    background-color: #grey;  
}
```

Ex.: O elemento com o id **campo-email** terá 30px de altura e **cinza** como cor de fundo

# Estilização de Páginas Web



Outro seletor muito utilizado é o seletor **universal**, é usado principalmente quando queremos remover alguma estilização padrão (CSS Reset)

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

Ex.: CSS Reset, zerando os valores de **margin** e **padding**, e definindo o **box-sizing** para border-box

# Estilização de Páginas Web



Referências para seletores CSS:

W3Schools

[https://www.w3schools.com/cssref/css\\_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp)

Mozilla Developer Network - MDN

<https://developer.mozilla.org/en-US/docs/Web/CSS>



# Estilização de Páginas Web



Tratando de posicionamento e disposição dos elementos, o CSS3 tem uma técnica de chamada **flexbox**, com essa técnica, podemos facilmente criar layouts mais flexíveis e responsivos, sem utilizar técnicas mais antigas e mais rígidas como o **float** e o **position**.



# Estilização de Páginas Web



Principais propriedades:

**display: flex;** -> Indica que o elemento é um container flex, e habilita as demais propriedades do flexbox

**flex-direction:** -> Propriedade que indica o sentido de disposição dos seus elementos filhos

**justify-content:** -> Determina o alinhamento dos itens no mesmo sentido de disposição dos elementos do container



# Estilização de Páginas Web



Referências:

Guia do Flexbox

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Flexbox Froggy – Exercício para fixação

<https://flexboxfroggy.com/>



# Estilização de Páginas Web



Para algumas ocasiões específicas, pode ser interessante utilizar a técnica de **position**.

Como por exemplo, **fixar** o cabeçalho da página para torná-lo disponível ao clique ao longo de toda a rolagem



# Estilização de Páginas Web



```
header {  
    height: 80px;  
    display: flex;  
    justify-content: space-around;  
    align-items: center;  
  
    position: fixed;  
    z-index: 1;  
    top: 0; }
```

Fixa o elemento na tela

Faz com que o elemento  
fique por cima dos outros

Posiciona o elemento à  
uma distância de 0px da  
extremidade superior da  
página, ou seja, no topo

# Estilização de Páginas Web



Referências:

Referencia de Position, W3Schools

[https://www.w3schools.com/cssref/pr\\_class\\_position.asp](https://www.w3schools.com/cssref/pr_class_position.asp)



# Estilização de Páginas Web



É muito comum em várias aplicações, utilizarmos dependências, que são arquivos externos que adicionam alguma funcionalidade ou característica ao projeto.

Podemos, por exemplo, adicionar fontes e ícones prontos de provedores CDN, com o intuito de implementar designs mais diversos.



# Estilização de Páginas Web



Duas opções de CDN que oferecem essas possibilidades, são o FontAwesome e o Google Fonts. Ambos possuem guias de utilização bem simples e fáceis.

FontAwesome

<https://fontawesome.com/>

Google Fonts

<https://fonts.google.com/>



# **Estilização de Páginas Web**



## **Responsividade**

Usando todos esses conceitos, ainda é necessário garantir que a página desenvolvida seja utilizada pelo público alvo do negócio, e a maioria das pessoas atualmente acessam sites da internet utilizando dispositivos móveis, como smartphones e tablets.



# Estilização de Páginas Web



Para garantir que nossas aplicações estejam acessíveis e com boa usabilidade nos dispositivos móveis, temos que entender e implementar os conceitos de **responsividade**.

Uma aplicação **responsiva** é uma aplicação que se adequa perfeitamente aos diferentes tipos diferentes de dispositivos.



# Estilização de Páginas Web



Para isso temos que sempre considerar a largura de tela dos dispositivos de destino no desenvolvimento das aplicações, pois esse é o principal fator que a aplicação terá de se adequar.

Essa adequação é feita pelo CSS pelas **media queries**, que com base em características do dispositivo, como largura da tela, consegue mudar alguma configuração de estilo de uma página.



# Estilização de Páginas Web



Nas media queries é possível modificar qualquer configuração de estilo feita por CSS. Por exemplo, podemos modificar a largura de uma imagem, em dispositivos com largura de tela inferior à 768px

```
1  img {  
2      width: 600px;  
3  }  
4  
5
```

Configuração padrão da tag <img>

```
1  @media screen and (max-width: 768px) {  
2      img {  
3          width: 200px;  
4      }  
5  }  
6
```

Configuração da tag <img>, para dispositivos de telas menores



# Linguagem de Programação para Páginas Web

**SENAI**

# Histórico da Linguagem

JavaScript, nomeada inicialmente por LiveScript, é uma linguagem de programação criada em por Brendan Eich a pedido da Netscape no ano de 1995.

Na época a empresa SUN Microsystem (Java), hoje sob domínio da Oracle, entrou de cabeça na colaboração do desenvolvimento da linguagem, pois viu vantagem em sua ideia inicial, a validação de formulários web.

Foi criada para ser Server Side, porém acabou como Client Side.

# Histórico da Linguagem

A Microsoft percebeu o poder do JavaScript e ao invés de ajudarem no desenvolvimento acabou desenvolvendo sua própria linguagem, a Jscript para rodar apenas em seus navegadores o Internet Explorer.

Foi uma dor de cabeça para os desenvolvedores da época, pois deveriam dominar as duas linguagens para os seus sites rodarem em todos os navegadores.



# Histórico da Linguagem

A ECMA International, empresa dedicada a padronização de sistemas de informação, criou a especificação ECMA-262, utilizada em tecnologias web em geral e nomeada de ECMAScript.

A Netscape submeteu o JavaScript para as especificações ECMA-262 no final de 1996.

Tanto JavaScript como JScript são baseadas em ECMAScript porém com recursos adicionais.

Atualmente a linguagem ECMAScript adotou o nome JavaScript por motivos comerciais e hoje podemos considerar a linguagem padrão da Web.

# Programa

## Conceito

Um programa, independente de linguagem, é composto por um conjunto de instruções que são os passos para a resolução de um determinado problema.

A solução se divide em 3 etapas principais:



# Exemplo

```
//entrada
let n1 = prompt("Digite o primeiro número: ");
let n2 = prompt("Digite o segundo número: ");

//processamento
let resultado = n1 + n2;

//saída
console.log("Resultado da operação " + resultado);
```

- Tivemos os dados imputados pelo usuário;
- Processamento do cálculo previamente armazenados;
- E a exibição do resultado processado, para o usuário;
- ENTRADA – PROCESSAMENTO – SAÍDA;



# Variável

## Conceito

Variável é um recurso computacional que serve para armazenamento de dados temporários na memória.

A grosso modo é composta de um nome e um valor, como no exemplo anterior onde tivemos os números digitados pelo usuário armazenados em duas variáveis para utilização posterior, em um cálculo aritmético.

O valor de uma variável, claramente, varia com o tempo.

# Exemplo 1

```
let n1 = parseFloat(prompt("Digite o primeiro número: "));  
let n2 = parseFloat(prompt("Digite o segundo número: "));
```

## Exemplo 2

Uma variável possui este nome por armazenar valores que podem variar com o tempo de execução do seu programa. Veja:

```
11 let n1 = 5;  
12 n1 = n1 * 2;  
13 alert(n1);
```

- Na linha 11 definimos uma variável com o nome n1 e valor 5 (não é necessário utilizar a instrução "var", devido ao fato de JavaScript ser de tipagem fraca);
- Na linha 12 o valor da variável n1 foi substituído pelo valor que ela tinha multiplicado por 2
- Na linha 13 foi exibido o valor da variável para o usuário. Repare que o valor foi referente à última atualização da variável;

# Regras básicas para definição

- Pode começar apenas com letras, \_ ("underlines") ou \$ ("cifrão" apenas no início);
- Pode conter letras, underlines e números (caracteres alfanuméricos), a partir do segundo caracter;
- NÃO PODE conter espaços;
- NÃO PODE começar com números;

```
//Nomes válidos
let n1 = 5;
let nome_completo;
let _n1;
let _;
let $nome;
let $;
```

```
//Nomes Inválidos
let 1a;
let nome completo;
let valore$;
let nome _completo;
```



# Tipos de Dados

## Conceito

Tipos de dados são os possíveis valores que podem ser atribuídos às variáveis. São eles:

### Number

Qualquer tipo numérico como real (float) ou inteiro (Integer);

### String

cadeia de caracteres imutáveis onde cada caracter está contido em uma posição, da esquerda para a direita, a partir do zero e pode ser utilizado aspas duplas ou simples:

```
// posições 0123456
let nome = "Eduardo";
let sobrenome = 'Costa';
```



# Tipos de Dados

## Bool

Valores que representam o estado verdadeiro (true) ou falso (false)

## Undefined

Inexistência ou variável não criada.

## Null

Ausência de valores. Neste caso a variável pode receber um valor null ou nunca ter recebido algum valor, porém a variável existe;

## NaN

É um tipo especial, na realidade uma propriedade global do JavaScript cujo seu valor especifica que o valor não é um número. É uma propriedade somente leitura. Você não pode configurá-la.

Exemplo: a expressão ("Eduardo" \* 10) retorna NaN.

```
let teste = "Eduardo" * 10;
console.log(teste === NaN);
console.log(isNaN(teste));
```



# Tipos de Dados

## infinity

É um tipo especial que especifica que o valor é infinito. Quando se faz uma divisão por zero em algumas linguagens, é retornado o valor "não existe divisão por 0", porém matematicamente esse valor é um valor infinito tendendo a zero. Dessa forma, no JavaScript teremos o mesmo resultado.

```
let x = 5 / 0;  
console.log(x); //infinity
```



# Tipos de Dados



## Object e Arrays

Estruturas de dados formadas por pares de chave e valor

### Array

Tipo de dado composto por índices e valores onde esses valores podem ser qualquer tipo de dado suportado pela linguagem JavaScript, inclusive outro array.

```
// índices      0      1      2
let foo = ["eduardo", 36, 1.67];
```



# Tipos de Dados



## Object

Tipo de dado composto por pares de chave e valores onde esses valores podem ser qualquer tipo de dado suportado pela linguagem JavaScript, inclusive outro objeto.

```
//objeto pedido
let pedido = {
  codigo : 5563,
  total : 131.98,
  itens : [ //array de objetos
    { //item 0 - primeiro item
      "descricao" : "Camiseta Polo",
      valor : 60.98
    },
    { //item 1 - segundo item
      descricao : "Calça Jeans",
      valor : 60.98
    }
  ]
};
```



# Introdução ao DOM



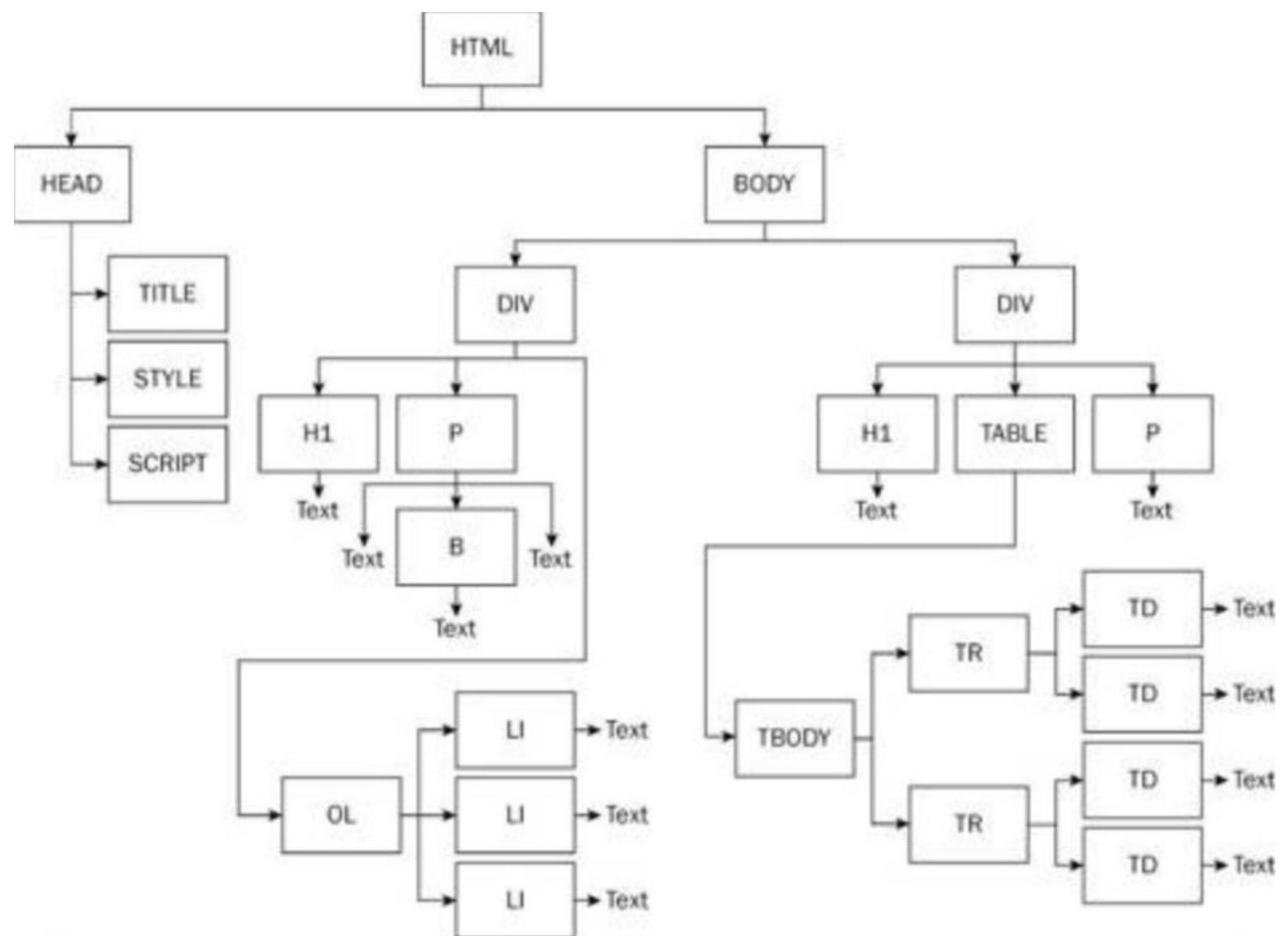
DOM – Document Object Model ou Modelo de Objeto de Documento, é uma interface que representa o documento HTML da página web.

E uma interface de programação para manipulação de documentos HTML, XML e SVG e está disponível para várias linguagens.

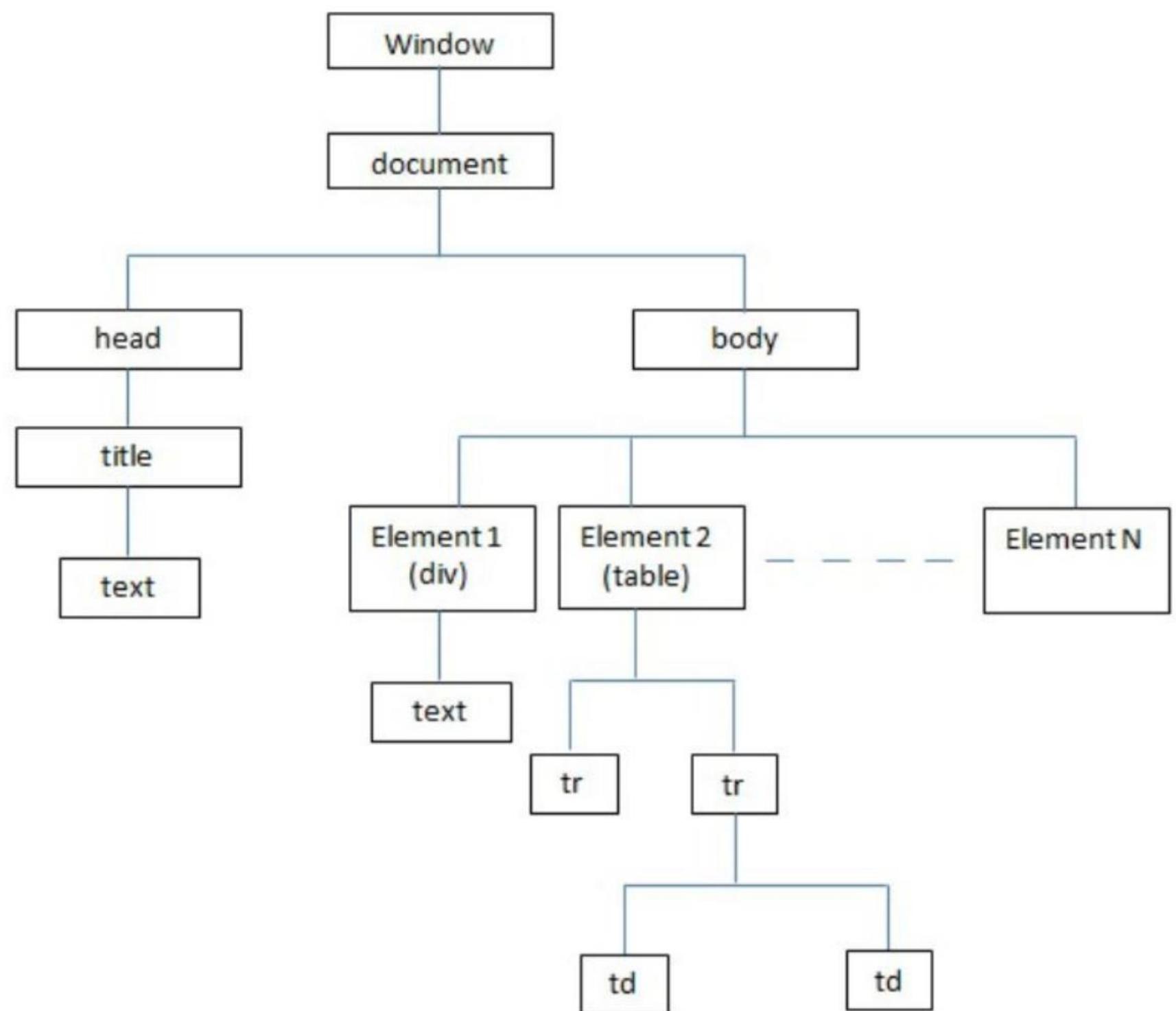
O DOM fornece essa interface através de nós e objetos para que possamos manipular toda nossa página através de métodos e propriedades, em nosso caso, JavaScript, onde veremos mais adiante



A ideia lógica do DOM é como uma árvore do HTML conforme a imagem



A hierarquia começa pelo objeto window:  
window.document.body ou apenas  
document.body



# Métodos DOM

Nós veremos os principais métodos para o entendimento na criação e manipulação de elementos HTML através do DOM.

Lembrando que JavaScript é case sensitive isso quer dizer que os métodos do DOM são escritos dessa forma: **getElementById** e não getelementbyid.

Os nomes desses métodos foram assinados com o padrão *Camel Case* a partir da segunda palavra.



# getElementById()

Pegando um elemento através do id e retorna o seu objeto.

```
<form class="form">
  <label class="form__label" for="nick-name">Nome</label>
  <input class="form__input" type="text" id="nome" value="Eduardo" >
  <label class="form__label" for="nick-name">Nome</label>
  <input class="form__input" type="text" id="email" value="eduardo@email.com">
</form>

<script>
  var elemento = document.getElementById('email');
  console.log(elemento);
  console.log(elemento.value);
</script>

<input class="form__input" type="text" id="email" value="eduardo@email.com">
```

eduardo@email.com

Opcionalmente você pode retornar apenas uma propriedade do objeto selecionado:

```
var elemento = document.getElementById('email').value;
console.log(elemento);
```

eduardo@email.com

# getElementsByClassName()

Seleciona uma coleção de elementos através do nome da classe. Perceba o plural "Elements"

```
<ul>
  <li class="corsim">item 1</li>
  <li class="cornao">item 2</li>
  <li class="corsim">item 3</li>
  <li class="cornao">item 4</li>
</ul>

<script>
  var itens = document.getElementsByClassName('cornao');
  console.log(itens);
  console.log(itens[1]);
</script>
```

▼ HTMLCollection(2) [li.cornao, li.cornao]
 ▶ 0: li.cornao
 ▶ 1: li.cornao
 length: 2
 ▶ \_\_proto\_\_: HTMLCollection
 <li class="cornao">item 4</li>

# getElementsByName()

retorna os elementos, ou seja, uma coleção, através do nome de uma tag

através ao nome de uma tag:

```
<form class="form">
  <label class="form__label" for="nick-name">Nome</label>
  <input class="form__input" type="text" id="nick-name" value="Eduardo" >
  <label class="form__label" for="email">Nome</label>
  <input class="form__input" type="text" id="email" value="eduardo@email.com">
</form>

var elementos = document.getElementsByName('input');

console.log(elementos);

r HTMLCollection(2) [input#nick-name.form__input, input#email.form__input, nick-name: input#nick-name.form__input, email: input#email.form__input] ⓘ
▶ 0: input#nick-name.form__input
▶ 1: input#email.form__input
  length: 2
▶ email: input#email.form__input
▶ nick-name: input#nick-name.form__input
▶ __proto__: HTMLCollection
```

# getElementsByName()

retorna os elementos, ou seja, uma coleção, através do nome de uma tag

atraves do nome da tag:

```
<form class="form">
  <label class="form_label" for="nick-name">Nome</label>
  <input class="form_input" type="text" id="nick-name" value="Eduardo" >
  <label class="form_label" for="email">Nome</label>
  <input class="form_input" type="text" id="email" value="eduardo@email.com">
</form>

var elementos = document.getElementsByTagName('input');

console.log(elementos);

// HTMLCollection(2) [input#nick-name.form_input, input#email.form_input, nick-name: input#nick-name.form_input, email: input#email.form_input]
  ▷ 0: input#nick-name.form_input
  ▷ 1: input#email.form_input
  length: 2
  email: input#email.form_input
  nick-name: input#nick-name.form_input
  __proto__: HTMLCollection
```

```
var elementos = document.getElementsByTagName('input');

for (var posicao = 0; posicao < elementos.length; posicao++) {
  console.log("valor do input " + elementos[posicao].id.toUpperCase());
  console.log("    " + elementos[posicao].value);
}
```

valor do input NOME

Eduardo

valor do input EMAIL

eduardo@email.com



# querySelector()

Retorna um elemento, da mesma forma que o getElementById() porém você utiliza um seletor CSS ao invés do nome do ID.

```
<form class="form">
  <label class="form_label" for="nick-name">Nome</label>
  <input class="form_input" type="text" id="nick-name" value="Eduardo" >
  <label class="form_label" for="email">Nome</label>
  <input class="form_input" type="text" id="email" value="eduardo@email.com">
</form>
```

```
var elemento = document.querySelector('input');
console.log(elemento);
```

```
<input class="form_input" type="text" id="nome" value="Eduardo">
```



DICA: Em resumo, você pode pegar um elemento utilizando os seletores CSS disponíveis. querySelector.

```
var elemento = document.querySelector('[id*="name"]');
console.log(elemento);

<input class="form__input" type="text" id="nick-name" value="Eduardo">

var elemento = document.querySelector('input:last-child');
console.log(elemento);

<input class="form__input" type="text" id="email" value="eduardo@email.com">
var elemento = document.querySelector('.form__input');
console.log(elemento);

<input class="form__input" type="text" id="nick-name" value="Eduardo">
var elemento = document.querySelector('.form__input:last-child');
console.log(elemento);

<input class="form__input" type="text" id="email" value="eduardo@email.com">
```

# querySelectorAll()

Funciona da mesma forma que o querySelector porém retorna um Array.

```
<form class="form">
  <label class="form__label" for="nick-name">Nome</label>
  <input class="form__input" type="text" id="nick-name" value="Eduardo" >
  <label class="form__label" for="email">Nome</label>
  <input class="form__input" type="text" id="email" value="eduardo@email.com">
</form>
```

```
var elementos = document.querySelectorAll('.form__input');

//imprime os values de cada input no console
for (var i = 0; i < elementos.length; i++) {
  console.log(elementos[i].value);
}
```

Eduardo  
eduardo@email.com



## createElement() and appendChild()

Os dois próximos métodos da interface DOM servem para criar elementos e adicioná-los ao DOM.

É como se fossemos utilizar o innerHTML, porém aqui a ideia é criar cada elemento como um objeto separadamente.

Imagine que você precisa criar um elemento div qualquer e adicioná-lo em sua página. Com innerHTML fizemos algo como na imagem abaixo:

```
var div = '<div></div>';  
document.body.innerHTML = div;
```



Com `createElement` você cria um fragmento do elemento em memória e logo em seguida utiliza o método `appendChild` para inserir o elemento de fato no HTML. Veja:

```
var div = document.createElement('div');
document.body.appendChild(div);
```



# removeChild()

Remove um nó filho do DOM. Devolve o nó removido.

## HTML

```
<div id="topo" align="center">
  <div id="interno"></div>
</div>
```

## CSS

```
var d = document.getElementById("topo");
var d_interno = document.getElementById("interno");
var noRemovido = d.removeChild(d_interno);
```



# parentNode

**Node.parentNode** é uma **propriedade** DOM somente leitura que retorna o nó (node) parente de um Node referenciado na árvore DOM. É extremamente aconselhável que você conheça a estrutura DOM para um melhor estudo e aprendizado.

```
// Obtém o primeiro <li> de uma lista
var li = document.getElementById('li-first');
// A partir do <li> obtido, obtém o elemento <ul>
var ul = li.parentNode;
```



The SENA logo is displayed in white on a red rectangular background. The letters 'SENAI' are bold and italicized, with horizontal lines extending from the top and bottom of the letters 'E' and 'I'.

DEPARTAMENTO REGIONAL  
**DE SÃO PAULO**

[www.sp.senai.br](http://www.sp.senai.br)