Front End 2

# Aula 01 – Assíncrona: Bem-vindo

Nós da equipe docente queremos lhe desejar boas vindas nessa etapa que iniciamos juntos. Durante todo percurso iremos aprender novos conceitos, diferentes metodologias de trabalho e nos conhecer como pessoas.

Serão apresentados recursos que irão adicionar interatividade à camada HTML e CSS, tornando possível capturar os dados e interações gerados pelo usuário com o intuito de posteriormente manipular os dados e a camada visual da aplicação.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* MÓDULO 1: Boas-vindas e Introdução ao Javascript Front I
  + Boas-vindas
  + Introdução ao Javascript Front
* MÓDULO 2: Manipulando DOM
  + Introdução ao DOM
  + Modificando elementos HTML com Javascript
  + Trabalhando com nós.
* MÓDULO 3: Web reativa
  + Recapitulando + Introdução a eventos
  + Eventos do usuário
* MÓDULO 4: Validação do lado do cliente
  + Primeira entrega
  + Formulários I
  + Formulários II: Como validar?
  + JSON e storage
  + Recapitulando To-do app
* Módulo 5: Assincronismo e APIs
  + Introdução a requisições assíncronas
  + APIs I
  + APIs II
  + To-do app: documentação
  + To-do app: fluxo de trabalho
  + To-do app: segurança
* Módulo 6: Implementação prática, animações e bibliotecas
  + To-do app: MVP
  + Avaliação final
  + Animações
  + To-do app: Melhorando experiência do usuário
  + Bibliotecas de terceiros
  + Encerramento da matéria

Conteúdos

* Módulo 1: Bem-vindo e introdução ao Javascript em interfaces
  + Aula 1: Bem-vindo
    - Fornecer aos alunos uma abordagem do que será apresentado no módulo e uma breve revisão dos principais tópicos do Javascript aplicado em interface web.
    - Introdução a matéria
    - Origem do Javascript
    - DevTools: Console
    - Integrando javascript em um documento HTML
  + Aula 2: Introdução ao Javascript em interfaces
    - Apresentar os recursos do Chrome Devtools a partir de uma implementação de funcionalidades com a linguagem Javascript.
    - Alert, prompt e confirm
    - For...in
    - For...of
  + Aula 3: Revisão
    - Revisar o conteúdo da semana.
* Módulo 2: Manipulando o DOM - O módulo é dedicado à manipulação de elementos HTML em um site.
  + Aula 4: Introdução ao DOM
    - Compreender o DOM (Document Object Mode) para começar a manipular seus elementos.
    - Objeto Window
    - Objeto Document
    - Selecionando elementos
  + Aula 5: Modificar elementos com Javascript
    - O conhecimento sobre manipulação do DOM é expandido, mas desta vez trabalhando com estilos dos elementos do documento.
    - Manipulando o DOM
    - Template literal
    - Propriedade style
    - Propriedade classList
  + Aula 6: Trabalhando com nós
    - Compreender o que são nós, quais recursos e como utilizá-los para manipular os nós.
    - Nós
      * Criar
      * Adicionar
      * Remover
    - Manipulando atributos
      * getAttribute
      * setAttribute
      * hasAttribute
      * removeAttribute
* Módulo 3: Web reativa - Com base na interação do usuário, acione diversos recursos para gerar dinamismo a uma aplicação web.
  + Aula 7: Revisão + Introdução a eventos
    - Compreender a parte dinâmica do Javascript, introdução a programação reativa.
  + Aula 8: Eventos
    - Estudar como podemos acionar diferentes procedimentos com base nas interações que o usuário tem com os elementos da aplicação web.
      * Quais são os eventos?
      * Eventos do mouse
      * Eventos do teclado
      * Eventos de tempo
  + Aula 9: Disparando eventos
    - Eventos que disparam funções.
* Módulo 4: Validação do lado do cliente - Tendo trabalhado com os elementos de formulário, chega hora de validar os dados com os recursos ensinados anteriormente.
  + Aula 10: Primeira entrega
    - Revisão dos tópicos vistos e apresentação dos recursos do produto entregável (avaliação)
    - Método de entrega
    - Forma de avaliação
    - Requisitos de entrega
  + Aula 11: Formulários I
    - Como capturar os elementos de um formulário e manipular seus valores.
    - Anteriormente em Programação Imperativa
      * Objetos literais
      * Métodos de strings
    - Obtendo dados de um formulário
      * Label
      * input (value)
      * Textarea
      * Select
      * Radio button
      * Check box
    - Outros métodos de strings
      * trim
      * toUpperCase
      * toLowerCase
      * concat
      * replace
  + Aula 12: Revisão - Compreender o papel do formulário na interação com o usuário.
  + Aula 13: Formulários II - Como validar?
    - Compreenda quais eventos podem ser utilizados na interação do usuário e enviar respostas visuais para validar os dados inseridos.
    - Eventos de formulário
      * focus
      * blur
      * change
      * submit
      * preventDefault()
    - Validação do cliente
    - Validação assíncrona
      * setTimeOut para imitar “promises”
    - Objeto Location
      * href
      * reload()
      * search
      * URLSearchParams
      * query.has()
      * query.get()
  + Aula 14: JSON e storage
    - Compreender a importância do JSON na comunicação web e como podemos armazenar informações no navegador.
    - Anteriormente em Programação Imperativa
      * JSON
    - Session Storage e Local Storage
    - Praticando em um formulário de login
  + Aula 15: Revisão + Aplicativo To-Do
    - Apresentação do projeto integrador.
* Módulo 5: Assincronismo e APIs - Entendendo as particularidades das requisições assíncronas e aproveitando as vantagens de consumir APIs (Application Programming Interface).
  + Aula 16: Introdução às requisições assíncronas
    - Revisão sobre o que são requisições assíncronas e compreensão de como funciona a comunicação web.
    - Anteriormente em Programação Imperativa
      * Promessas
      * async/await
      * Arquitetura cliente-servidor
    - Request e Response
    - Métodos HTTP
    - Status (200, 300, 400, 500)
  + Aula 17: APIs I
    - Compreender o que é uma API e conhecer os principais recursos que o Javascript oferece de forma nativa para o desenvolvimento de aplicações web.
    - O que é API?
      * REST
      * Documentação
    - Fetch (GET)
    - try/catch /finally
    - APIs públicas
  + Aula 18: APIs II
    - Explorar os recursos da linguagem Javascript responsáveis por enviar e receber dados de uma API.
    - Fetch
      * Método POST
      * Método PUT
      * Método DELETE
    - Postman
  + Aula 19: App To-Do - Documentação
    - Documentando a aplicação To-Do.
  + Aula 20: App To-Do - Fluxo de trabalho
    - Definindo as funcionalidades da aplicação To-Do.
  + Aula 21: App To-Do: Segurança
    - Operações de CRUD (Create, Read, Update e Delete); Importância e conceitos sobre o Token JWT.
* Módulo 6: Prática de implementação, animações e importação de bibliotecas - Colocar em prática os últimos tópicos aprendidos sobre o aplicativo de tarefas (To-Do). Serão explorados o uso de animações, importação e utilização de bibliotecas de terceiros para enriquecer a experiência do usuário ao navegar na aplicação web.
  + Aula 22: To-Do App: MVP
    - Apresentação do projeto integrador. Prática de apresentação de um projeto.
  + Aula 23: Avaliação final
    - Aula dedicada à segunda e última avaliação da disciplina.
  + Aula 24: Animações
    - Estimular formas criativas de uso dos recursos de animação anteriormente abordados, para melhorar a experiência do usuário na aplicação.
    - Animações CSS e eventos
    - Tela de pré-carregamento
    - ...
  + Aula 25: To-Do App: Melhorias na experiência do usuário (UX)
    - Register spinner. Login spinner. Tasks skeleton.
  + Aula 26: Bibliotecas
    - Apresentar exemplos de bibliotecas de terceiros mais populares e compreender o processo de instalação na aplicação. Apresentação sobre React, assim como recursos e vantagens do framework.
    - Bibliotecas
      * Chart.js
      * Sweet Alert 2
      * Anime.js
      * AOS
    - Introdução ao Framework React
  + Aula 27: Encerramento da disciplina
    - Encerramento e retrospectiva dos conteúdos abordados.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Diagrama

Descrição gerada automaticamente

Tabela, Calendário

Descrição gerada automaticamente

## Origem ao JS

Quando nasceu o JavaScript? Quem é Brendan Eich? Estas são algumas das questões que iremos responder para entendermos o que o JavaScript pretendia originalmente resolver e como chegou a ser uma das linguagens de programação mais utilizadas em todo o mundo.

JS é uma das 3 principais linguagens para desenvolver sites. Enquanto o HTML cuida da estrutura e CSS trata do estilo, JS permite adicionar comportamentos e funcionalidades. Isso permite que os visitantes interajam com o conteúdo de forma dinâmica.

A década de 1990 foi importante para a internet. Empresas como Netscape e Microsoft estavam no meio da guerra dos navegadores. A NetScape foi fundada por Mike Andreessen e estava crescendo rapidamente para capturar quase 80% do mercado de navegadores. Mas os wbdesigners precisavam de uma linguagem mais interativa para desenvolver seus sites. No início, eles usavam a linguagem Java, mas perceberam que era necessário algo mais dinâmico e acessível. Nesse momento, em 1995, entra em cena Brendan Eich, que é recrutado pela Netscape para criar um protótipo de linguagem para o navegador, com a pressão de fazê-lo antes da Microsoft. Em apenas 10 dias, ele cria o Mocha, uma linguagem baseada na sintaxe Java, funções de primeira classe tipo Scheme, tipos dinâmicos como Lisp e propótipos como Self. Embora se chamasse Mocha, no Netscape decidiram lançar como LiveScript. Mas como todos pensavam que Java era a coisa mais incrível do mundo, decidiram renomear para JavaScript. Foi uma jogada de marketing para se aproveitar da fama do Java, embora essas linguagens não tenham nada a ver. Em 1997 JS já era um sucesso e foi decidido padronizar a linguagem para evitar competição. Assim, ela é enviada ao órgão ECMA e algumas pessoas começaram a chamar de ECMAScript. Um ano depois, em 1998, é criado o ECMAScript3, que continua adicionando funcionalidades. Em 2006 nasceu uma das bibliotecas JS mais conhecidas, o JQuery. Durante 2008, o Google lançou um motor V8 JavaScript junto com seu navegador Chrome. O backend não escapa do JS e em 2009 nasceu o Node.js, um ambiente que permite que o JS seja executado fora do navegador no lado do servidor. Desde sua criação, o JS passou por muitas modificações e implementações. Ele não para de crescer.

Uma imagem contendo Gráfico

Descrição gerada automaticamente

Para a revisão, este são alguns conceitos de JavaScript para iniciarmos em JavaScript Front end:

### Variáveis

#### var

Declara uma variável global ou no escopo de uma **função.**

var varGlobal= 'valor inicial';

function name() {

// escopo da função

console.log(varGlobal); // -> 'valor inicial'

if (condicao) {

// escopo do bloco

var varFuncao= 'Estou declarada em uma função';

console.log(varGlobal); // -> 'valor inicial'

varFuncao = 'Contínuo dentro da Função';

}

console.log(varFuncao); // -> 'Eu continuo dentro da Função' }

console.log(varGlobal); // -> 'outro valor'

console.log(varFuncao); // -> erro: não está declarada

#### let

Declara uma variável dentro do escopo de um bloco.

let varGlobal = 'valor inicial';

function name() {

// escopo da função

console.log(varGlobal); // -> 'valor inicial'

if (condicao) {

// escopo de um bloco

let varFuncao = 'Estou declarada em uma função?'; console.log(varGlobal); // -> 'valor inicial'

varGlobal = 'outro valor';

}

console.log(varFuncao); // -> error não está declarada

// let faz que a variável esteja disponível somente dentro do if }

console.log(varGlobal); // -> 'outro Valor'

console.log(varFuncao); // -> error não está declarada

#### const

Declara uma **constante** no escopo de um bloco.

const varGlobal = 'valor inicial';

function name() {

// escopo da função

console.log(varGlobal); // -> 'valor inicial'

if (condicao) {

// escopo do bloco

const varFuncao = 'Estou declarada em uma função?';

console.log(varGlobal); // -> 'valor inicial'

varGlobal = 'otro Valor'; // erro a const não pode ser modificada }

console.log(varFuncao); // -> erro no está declarada

// const se comporta igual ao let em termos de alcance

}

console.log(varGlobal); // -> 'valor inicial'

console.log(varFuncao); // -> error no está declarada

### Tipos

let myVariable = 'Hello wold'; // é uma string

let myVariable1 = 22; // é number

let myVariable2 = false; // é boolean

let myVariable3; // É undefined

let myVariable4 = { nome: 'meu nome’' } ; // É um objeto

let myVariable5 = null; // É um objeto (É um tipo de objeto especial)

let myVariable6 = function() { let doSomething; }; // é uma function

// Você pode verificar esses tipos usando typeOf typeOf myVariable // -> number

### Estruturas de controle

#### If

Permite executar um bloco somente se uma determinada condição for atendida.

if (condicao) {

// se verdadeiro, executa este bloco de código

}

#### If... else

Permite avaliar uma condição e executar um bloco de código ou outro.

if (condition) {

//se verdadeiro, executa este bloco de código

} else {

//se falso, executa este bloco de código

}

#### Switch

Permite executar diferentes ações dependendo do valor de uma variável.

switch (variavel) {

case 1:

// code if variavel == 1;

break;

case 2:

// code if variavel == 2;

break;

default:

// Executar se nenhuma condição anterior for encontrada

break;

}

### Laços

#### For

Permite que você execute repetidamente um bloco de código.

let n = 4;

for(var i = 0; i < n; i++) {

// código para executar n vezes (4)

}

**for(** inicial ; condição ; final **) { }**

**inicial**: Este código é executado no início único do loop, normalmente uma variável é declarada conforme mostrado no exemplo.

**condição:** Sempre que o bloco de código fechado termina, esta condição é verificada, e se for verdadeira, o loop termina e o código abaixo continua.

**final:** Uma ação a ser executada a cada vez que o bloco a ser repetido é finalizado, a variável usada na condição é comumente modificada.

#### While

Seu comportamento é semelhante a um loop 'for', mas o bloco continuará executando indefinidamente enquanto a condição for verdadeira.

let n = 1;

while (n < 3) {

// código que será executado

}

while( *condicao* ) { }

condição: condição a ser avaliada antes de cada execução do bloco, se nunca se tornar falsa, o loop permanecerá em execução indefinidamente.

#### Do... while

Permite que você execute um bloco de código, desde que uma condição seja verdadeira. Ao contrário de while, a condição é avaliada no final de cada execução, que resulta na execução do bloco fechado pelo menos uma vez.

do {

// código que será executado

} while (n < 3);

## DevTools: Console

O console é uma ferramenta que os desenvolvedores possuem no navegador para tomar decisões sobre nosso projeto ao mesmo tempo em que ele é interpretado pelo Chrome.

Existem atalhos de teclado que, dependendo do seu sistema operacional, permitirão que você acesse o console.

Tabela

Descrição gerada automaticamente

No console, encontramos várias guias e suas funcionalidades:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Quando algo não funciona da maneira que desejamos, podemos gastar muito tempo tentando descobrir qual é o problema e corrigi-lo.

É por isso que o console nos ajuda a agilizar esse processo, pois nos informa o que acontece quando executamos uma tarefa ou solicitação ao servidor.

Pode ser que ele retorne os dados que procurávamos, um erro ou um aviso (warning). Esse processo é chamado de **depuração** do código ou **debugging**.

### Console

Se executarmos esses exemplos no console, qual resultado ele retornaria para mim em cada caso?

Texto

Descrição gerada automaticamente

### Elementos e Estilos

Na guia Elementos, veremos o código HTML e em Estilos, nosso CSS. Se clicarmos na seta indicada na imagem e depois colocarmos o mouse sobre os elementos da tela, perceberemos facilmente onde eles estão no código para poder manipulá-los e apreciar a mudança sem ter que ir ao editor de texto e salve.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

### Tipos de erros

Existem vários tipos de erros que são exibidos em vermelho no console. Os mais comuns que você conhecerá por enquanto são:

* SyntaxError: Representa um erro de sintaxe no idioma representado em JavaScript.
* TypeError: Representa um erro que ocorre quando uma variável ou parâmetro não é de um tipo válido, ou seja, indefinido.

### Responsivo

Muitas vezes precisamos adaptar nosso site para poder visualizá-los corretamente em diferentes dispositivos. Na parte superior do inspetor há um ícone de dispositivo, e clicando podemos determinar a resolução da tela ou escolher diretamente um dispositivo.

Texto

Descrição gerada automaticamente

### Aplicação

Com esta aba, podemos executar tarefas referentes aos dados que são salvos pelo cliente em nosso site por um determinado período de tempo. Por exemplo, excluir da memória os dados inseridos no login de um usuário.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

### Lighthouse

Lighthouse é uma ferramenta que gera relatórios para verificar recursos que a web deve ter para ser considerada ótima. Alguns deles são desempenho, acessibilidade e SEO.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

### Primeiros passos em Javascript

#### console.log( )

Agora que conhecemos o console como parte das ferramentas que devemos usar durante o desenvolvimento, vamos colocar algumas linhas de código em prática.

O clássico seria mostrar uma mensagem por console e para isso, devemos implementar o método **log( )** do objeto **Console**. Então, como seria?

console.log("Esta é uma mensagem por console");

Podemos tentar executar a mesma instrução com mensagens diferentes, mesmo com tipos de dados diferentes, e devemos ter o mesmo resultado. Por exemplo:

console.log("Olá, sou outro texto");

console.log(25);

console.log(true);

🚨 É sempre importante colocar seu código em prática, experimentar diferentes possibilidades e não se preocupar com bugs. Além disso, o aparecimento de erros é fundamental para o processo de aprendizagem, se não os encontrarmos dificilmente estaremos exigindo o suficiente de nós mesmos.

#### Variantes

Como alternativas ao **log( )**, mas raramente implementado, é bom saber que existem outros métodos, tais como:

**.error( )** → Escreve um erro no console.

**.warn( )** → Escreve uma advertência no console.

**.table( )** → Escreve uma tabela no console.

O último é outra maneira de exibir um objeto ou array no console de uma forma um pouco mais visual. Por exemplo:

let meuObjeto= {

mensagem: "Mensagem de texto",

utilidade: "Prova de JS"

};

let meuArray= [ “Primeira mensagem do array”, ”Segunda mensagem do array”, “Terceira mensagem do array”]

console.table(meuObjeto);

console.table(meuArray);

Continue tentando alternativas, familiarize-se com o console do navegador e implemente o código para obter resultados diferentes. **Você também pode realizar operações aritméticas e relacionais.**

let numero = 5;

let numeroEmLetras = "5";

console.log(numero == numeroEmLetras);

console.log(numero === numeroEmLetras);

#### alert( )

Assim como exibimos mensagens por console, muito úteis para depurar nosso código, também temos **mensagens de alerta que o usuário pode ver**.

O método **alert()** pertence ao objeto **window**, mas para usá-lo podemos implementá-lo diretamente no console.

alert("Este é um alerta.");

O método mostra uma caixa de alerta, com a mensagem que passamos como parâmetro, e um botão OK. Ele é implementado precisamente para mostrar ao usuário certas informações que consideramos importantes. Desta forma simples e rápida já podemos nos comunicar com o usuário.

Não é a interface mais bonita do mundo, mas pelo menos assim podemos mostrar mensagens mais práticas.

🎈Estes são os métodos mais simples, talvez isolados não nos pareçam muito úteis, mas depois os complementaremos com outros que darão funcionalidade ao nosso código.

# Aula 01 – Síncrona: 21.02.2022

## Vinculando HTML e JavaScript

### Vinculação interna

Nos permite escrever códigos JavaScript diretamente em nosso arquivo HTML.

No entanto, esta não é a forma mais adequada de se trabalhar.

Texto

Descrição gerada automaticamente

### Vinculação externa

Permite ligar nosso arquivo HTML com um arquivo JavaScript externo.

Texto

Descrição gerada automaticamente com confiança baixa

Lembre-se de que, com o uso de links (vinculação) externos, não é necessário escrever as tags <script> dentro do nosso arquivo .js.

Tela de celular

Descrição gerada automaticamente com confiança média

## Prática Integradora

Objetivo

Desenvolver um script que permita somar um array de números consecutivos, de forma que se some o primeiro número com o segundo e o imprima através do console.

Depois, temos que pegar este resultado e somar o terceiro número, e assim por diante, até termos terminado de percorrer todo o array.

Diagrama

Descrição gerada automaticamente

# Aula 02 – Assíncrona: Introdução ao JS em interfaces

Capturando dados do cliente

Como vimos anteriormente, JavaScript no navegador fornece-nos certos objetos e métodos que podemos aplicar com uma simples chamada a eles. Para continuar usando os métodos Window, podemos adicionar outros que permitem ao usuário inserir informações, que podemos capturar e usar em nossos programas.

Há apenas 3 funções que podemos fazer para interagir com o usuário. A primeira é alert(), que interromperá o uso para informar um alerta ao usuário. Em segundo lugar temos a função confirm() que interromperá a navegação em uma caixa de texto com as opções aceitar ou cancelar. Podemos então armazenar true se o usuário aceitar ou false se ele clicar em cancelar. Em terceiro lugar temos a função de prompt(), que novamente nos permitirá comunicar uma mensagem ao usuário, só que desta vez o usuário poderá escrever qualquer coisa em resposta. Por isso, armazenaremos como resultado o que o usuário escreve.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Nas capturas de vídeo, podemos ver no navegador uma arquitetura Node JS, e além disso, abaixo você pode ver um terminal. Ambas as caixas não devem importar para nós, devemos nos concentrar apenas no código implementado.

### prompt()

Um deles é o prompt(), que mostra uma caixa de diálogo com uma mensagem opcional, que pede ao usuário para inserir um texto. Você também tem duas opções: "Aceitar" ou "Cancelar".



No console do navegador, podemos ver o resultado que esse método retorna, independente do que o usuário inseriu em uma string de texto.

Isso significa que não importa o quanto o usuário insira apenas números, o que obteremos serão esses números, mas como se fossem texto.

No momento em que os dados são perdidos, se os armazenamos, no entanto, em uma variável, podemos usá-los para o resto do nosso programa. Vamos fazer esse teste.



Ao usar um espaço de memória para salvar esses dados, podemos usá-los como informações. Caso o usuário selecione "Cancelar", o resultado retornado é nulo. Por outro lado, se o usuário pressionar "Aceitar" sem completar nada, ele retornará um texto vazio. Vamos também tentar ver neste caso e checar o resultado.

### confirm()

O método confirm() exibe uma caixa de diálogo com uma mensagem opcional e dois botões, “OK” e “Cancelar”. Neste caso, o que nos permite é colocar uma questão ou indicação ao utilizador para que apenas responda sim ou não.

O valor que será retornado é um booleano indicando verdadeiro se clicarmos em OK e falso se escolhermos Cancelar.



Lembremos que por si só o método não armazena nenhum dado, mas se armazenarmos o valor retornado em uma variável, podemos utilizá-lo como informação. Vamos tentar combinar os dois métodos em algo mais funcional.

*let* tratamentoPersonalizado = confirm("Deseja um tratamento personalizado?")

if(tratamentoPersonalizado) {

*let* nome = prompt("Por favor, digite seu nome");

   alert("Bem-vindo ao nosso site"+ nome +". Muito obrigado pela sua visita, estamos à sua disposição! ?");

} else {

   alert("Obrigado por conectar.")

}

Agora podemos ver como faz mais sentido implementar os métodos, mas juntos para alcançar um resultado mais complexo. Ou, pelo menos, salve esses dados e use-os quando acharmos adequado.

Interface gráfica do usuário

Descrição gerada automaticamente

## Manipulando dados

Como já vimos, os resultados dos valores obtidos com o método prompt() nem sempre correspondem com o tipo de dado de que necessitamos. Ou seja, se pedirmos ao usuário que introduza a sua idade e a armazene numa variável para utilizar e adicioná-la ao ano corrente, veremos que não obtemos o resultado pretendido. 😐

*let* idade = prompt("introduza sua idade");

console.log(idade+2021);

É evidente que se acrescentarmos um número a um texto, não obteremos a soma dos dois, mas teremos o encadeamento dos dois como um só texto.

### parseInt()

Para evitar erros como o anterior ou outros erros que possam surgir da não verificação do tipo de dados que estamos manipulando, temos a função parseInt(). Esta função analisa uma sequência de texto e devolve em um número.

parseInt("22");

parseInt(prompt("Introduza idade"));

Mais uma vez, se não armazenarmos esses dados em algum lugar, dificilmente podemos fazer algo com eles. Para isso, implementamos variáveis que armazenam o resultado das funções. Vejamos o resultado.

*let* a = parseInt("22");

*let* b = parseInt(prompt("Introduza idade"));

*let* c = parseInt("22"+"150");

*let* d = parseInt(22+150)

*let* e = parseInt(22+parseInt("150"));

*let* f = parseInt(22.55);

console.log(a);

console.log(b);

console.log(c);

console.log(d);

console.log(e);

console.log(f);

Combinando e testando diferentes possibilidades, obteremos resultados diferentes. O fundamental é entender o funcionamento de cada método e função para aplicá-lo conforme nossas necessidades. Como podemos ver, num caso de pontualidade, observamos que a função parseInt() só devolve a parte inteira do número que introduzimos, por isso, se tivermos decimais, serão truncados.

### parseFloat()

Aqui entra em jogo esta outra função, que tem o mesmo objetivo que a anterior, mas neste caso devolve os números decimais que existem.

console.log(parseFloat(22.34));

console.log(parseFloat(22.3456284));

Se testamos estas funções e também por curiosidade, ou engano —ambos são extremamente úteis🙂—, tentamos analisar um texto, vimos que o resultado obtido não é um número.

### NaN

A propriedade NaN nos indica que o valor não é um número (Not A Number), portanto, isto causaria um erro se quiséssemos realizar qualquer operação aritmética com este valor.

Tomemos este exemplo de uma situação que conduziria a um erro. Suponhamos que no código seguinte, quando executado, na caixa de diálogo do prompt, o usuário, por engano ou de propósito — algo que precisamos ter em mente como programadores🧐 —, introduza um texto “sua idade”.

*let* edad = parseInt(prompt("Introduza sua idade"));

if(edad>18){

console.log("É maior de idade");

}else{

console.log("É menor de idade");

}

🚨 Claramente não estamos isentos do fato do utilizador ser um troll, portanto, temos sempre de procurar formas de validar os dados que o utilizador pode manipular.

### Desafio:

Te convidamos a resolver o seguinte desafio a fim de continuar a praticar. Para isso, abra o VS Code e cole o último bloco de código implementado.

● Qual é o resultado deste código?

● É correto o que retorna com base no que o utilizador introduziu?

● Onde poderia existir um problema?

● Como poderíamos resolvê-lo e alcançar um resultado melhor utilizando os métodos que já conhecemos? → Tip

🎈 Seja encorajado a refazer o código, pense em possíveis bugs e em como corrigi-los.

→ Dica: Temos a função isNaN( ), a qual nos retorna se o valor dado como parâmetro é NaN. Para saber mais, clique aqui.

EXTRA

### Math()

Como uma separação desta seção, trazemos para rever informações sobre um objeto que nos possa ser útil em algum momento do nosso desenvolvimento. Estamos falando de Math, que tem muitas propriedades e métodos que podem ser úteis.

Por que criar algo que já existe e resolve o problema para nós?

Quando falamos de objetos embutidos, queremos dizer "coisas já inventadas".

Em referência à frase anterior, a resposta pode variar, mas se houver uma função ou método que resolva um problema típico, podemos recorrer a eles. Nem sempre é necessário criar uma nova solução, podemos implementar as disponíveis.

O objeto Math é precisamente um dos casos, porque possui propriedades e métodos para constantes e funções matemáticas.

#### Propriedades

Raramente implementaremos qualquer uma das propriedades, mas todas elas estão disponíveis.

Mais informações sobre isso, clique [**aqui**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math#propiedades)**.**

Por exemplo, podemos usar a propriedade que PI retorna para nós.



\*No total, são 8 constantes matemáticas que podem ser acessadas.

#### Métodos

São funções matemáticas que já conhecemos, utilizadas diariamente para efetuar cálculos. Algumas delas são:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Para ver a lista completa de métodos, clique [**aqui**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math#propiedades)

#### Exemplos práticos

##### Math.random()

Obtém um número aleatório com base em um algoritmo.

* Não recebe parâmetros
* Retorna um ponto flutuante, um número pseudoaleatório dentro do intervalo [0, 1). De 0 (incluído) a 1 (excluído).

Texto

Descrição gerada automaticamente

##### Math.round()

Retorna o valor do número fornecido arredondado para o inteiro mais próximo.

Recebe um número, de preferência com um ponto flutuante (float).

Retorna o valor do número fornecido arredondado para o inteiro mais próximo.

Texto

Descrição gerada automaticamente

##### Math.max()

Recebe um conjunto de números.

Retorna o maior de zero ou mais números.

\*Se pelo menos um dos argumentos não puder ser convertido em um número, o resultado será NaN.

Texto

Descrição gerada automaticamente

## Ciclos específicos

Os ciclos ou loops são uma ótima ferramenta para poder passar por elementos iteráveis. Vamos adicionar mais algumas opções às que já conhecemos.

Essas frases em Javascript nos permitirão iterar os elementos usando uma sintaxe simples e clara.

Tabela

Descrição gerada automaticamente com confiança baixa

### For...in

Só itera sobre objetos literais.

O ciclo for…in nos permite iterar sobre cada uma das propriedades de um objeto.

Diagrama

Descrição gerada automaticamente

Forma, Retângulo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança média

Uma imagem contendo Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança média

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

### For....of

Posso usar com matrizes e Strings.

O ciclo for...of nos permite iterar sobre cada um dos valores de um elemento iterável, por exemplo, um array.

Diagrama

Descrição gerada automaticamente com confiança baixa

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança baixa

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança média

# Aula 02 – Síncrona: 22.02.2022

## Aprender para codificar, codificar para aprender

Para poder fixar os conhecimentos corretamente, é necessário pôr repetidamente em prática os conceitos vistos tanto nas aulas assíncronas como nas aulas ao vivo. Por esta razão, realizará constantemente atividades individuais e em grupo nas mesas de trabalho, com o objetivo de combinar diferentes casos para adquirir as competências necessárias para gerir os conceitos vistos.

### Jogo de Pedra, Papel ou Tesoura

Prática Integradora

Desafio - Com o professor

Programaremos um pequeno jogo de Pedra, Papel ou Tesoura. Neste jogo o usuário poderá escolher entre uma das três opções e o computador escolherá aleatoriamente uma das três opções, essas opções serão então comparadas e no final será mostrado quem ganhou e quem perdeu a disputa. Para fazer essa atividade é importante lembrar das regras básicas deste jogo:

* Pedra ganha de Tesoura.
* Tesoura ganha de Papel.
* Papel ganha de Pedra.

Desafio - Mesas de Trabalho

Após termos as regras e o sistema com uma partida do jogo, ficará ao encargo das mesas de trabalho:

* Refatorar o código, utilizando funções e a estrutura de decisão switch;
* Implementar uma regra de que só é declarado como ganhador, quem ganhou 2 rodadas, para auxiliar use uma variável para armazenar o vencedor de cada rodada;
* Empate no jogo, não dá pontuação para nenhum dos jogadores.
* Comentar o código e utilizar de qualquer outro recurso para deixá-lo melhor formatado.

# Aula 03 – Assíncrona: Revisão 1

Fixação dos conteúdos vistos

Chegamos ao dia da semana em que não veremos novos conteúdos. É tempo de rever o que vimos até agora e de esclarecer quaisquer dúvidas que possam ter surgido, tanto nas leituras como na prática. Recomendamos que revise o que foi apresentado anteriormente e reveja os exercícios práticos para tirar o máximo proveito da próxima aula ao vivo.

# Aula 03 – Síncrona: 23.02.2022

## Encerramento da semana

Calculadora

Prática Integradora

Objetivo

Vamos programar uma calculadora simples com o conhecimento que adquirimos até agora.

A calculadora terá 4 funções básicas:

Somar

Subtrair

Multiplicar

Dividir