Curso de JavaScript

Tipos de dados em JS

- Numérico (Em JS, todos os dados numéricos são considerados do tipo "number", não possuindo distinção entre float, int, unsgined int, etc...).
- String.
- Booleano.
- Valores vazios (null, undefined).

Operadores aritméticos em JS

- + → Soma | - → Subtração | / → Divisão | * → Multiplicação | % → Modulo (resto da divisão).

Special numbers

- São considerados números. mas tecnicamente não são.
- Infinito.
- Infinito negativo.
- NaN (not a number).

Sobre concatenar com JS

- Pode-se concatenar strings em JS usando o operador de soma, com a função str1.concat(str2), ou com template literals.
- Template literals exemplo: console.log(`4 + 6 = ${4+6}`)

Operadores de comparação:

- > → Maior que | < → Menor que| >= → Maior igual | <= → Menor igual | == → Igual (compara os valores) | === → Compara os valores e o tipo | != → Diferente

Operadores logicos:

- && → And | || → Or | ! → Not

Operador ternário:

- Basicamente, faz comparações em 1 linha e "troca" valores com base nisso.
- Exemplo: console.log(5 > 2 ? "Sim" : "Não");

Sobre declaração de variáveis em JS:

- Let nome_variavel (maneira mais atual e que separa os escopos)
- Var nome_variavel (maneira mais antiga e que não separa os escopos de maneira adequada, por exemplo, se declarado uma variável "x" dentro de um laço de repetição for, em uma função por exemplo, tal variavel pode ser acessada dentro de todo

escopo da função, não só do for)

- Const nome_constante → Valores em constantes não podem ser alteradas.
- A saber melhor sobre as diferenças entre let e var, teste este exemplo:

```
function teste(){
  for(var x = 0; x < 10; x++){
    console.log(x);
  }
  console.log(`Ultimo valor de x: ${x}`);
}

function teste2(){
    for(let x = 0; x < 10; x++){
                console.log(x);
        }
    console.log(`Ultimo valor de x: ${x}`);
} // Ira retornar um erro, dado o escopo da variavel x
teste();
teste2();
```

Algumas funções "pre-prontas" (built-in):

- Prompt() → Basicamente, abre um bloco no navegador do usuário, no qual pode ser inserido um input, e retorna um valor (podendo ser armazenado em uma variável, por exemplo). [Má pratica]
- Alert() → Exibe a mensagem passada como parâmetro na tela do usuário. [Má pratica]
- Math.x() → Classe com uma serie de funções relacionadas a matemática.
- Console.log() → Função, derivada da classe Console, que exibe no terminal o parâmetro passado.

Sintaxe de funções:

- function nomeFuncao(parametro1, parametro 2){
  codigo;
  codigo;
  }
- Não é possível especificar o tipo de dado em uma função. (Ao meu ver, prejudicial e torna a linguagem passível de alguns comportamentos indesejáveis).
- É possível declarar funções em variáveis/constantes, exemplo:

```
let a = function(){
    console.log("Teste");
}
```

- É possível definir valores padrões em parâmetros, como: function teste(a = 2){};

Arrow functions:

- Funções "sem nome", tendo de ser diretamente ligadas a uma variável/constante.
- Sintaxe:

```
const ola = (nome) =>{
  console.log(`Olá ${nome}`);
}
ola("Jorge");
// Output: "Olá Jorge"

// Ou, caso a função so tenha um parametro e so vá executar 1 ação:
```

```
const ola = nome => console.log(`Olá ${nome}`);
```

Conceito de closure:

- Com recorrência é usado para simular métodos privados.
- Consiste, basicamente, em uma função dentro de uma função.
- Exemplo:

```
function teste(texto){
    return function(){
    console.log(`Olá ${texto}`);
  }
}
let a = teste("Jorge");
a()
// Output = Olá Jorge
```

Arrays

- Assim como em python, e consequentemente diferente de C/C++, arrays em Javascript são basicamente listas que permitem o armazenamento de vários tipos de dados.
- Sintaxe: let a = [1,2,"ABC"]
- Método pop: remove o ultimo elemento do array
- Método push: adiciona um elemento na ultima posição do array
- Método unshift: adiciona um valor no primeiro elemento do array
- Método shift: Remove um valor no primeiro elemento do array
- Método indexOf(elemento): Retorna o primeiro índice do elemento passado como parâmetro
- Método lastIndexOf(elemento): Retorna o ultimo índice do elemento passado como parâmetro
- Método slice(indice_1,indice_2): Retorna um array partindo do indice_1 até o indice_2 - 1
- Método forEach(x =>{codigo} : Basicamente, faz uma ação para cada elemento do array
- Método includes(elemento): Verifica se um determinado elemento esta dentro do array
- Método reverse(): Retorna um array invertido.
- Destructuring: Basicamente, transforma os itens de um array em variaveis, exemplo:

```
let a = [1,2,3];
let [primeiro, segundo, terceiro] = a;
console.log(primeiro);
console.log(segundo);
console.log(terceiro);
```

Objetos

- Basicamente, objetos em javascript tem sintaxe semelhante a dicionários em python. Tem-se um nome (chave) com um valor atribuído (propriedade), ou um método.
- Exemplo:

```
let cidade = {
  nome : "Cidade de São João",
```

```
  prefeito : "João",
  endereco : "São Paulo",
  funcao_exemplo : () => console.log(`${cidade.nome} - ${cidade.prefeito} - ${cidade.endereco}`)
} // Observação, a palavra "this" poderia ter sido utilizada ao inves do nome "cidade", contudo, neste caso, não poderia ser utilizado uma 
console.log(cidade.nome) // Ou cidade["nome"]
console.log(cidade.prefeito) // cidade["prefeito"]
console.log(cidade.endereco) // cidade["endereco"]
cidade.funcao_exemplo()
```

- Para deletar propriedades: delete objeto.propriedade
- Para adicionar propriedades: objeto.novaPropriedade = valor;
- Para copiar itens do objeto A para o objeto B: Object.assign(objetoA, objetoB)
- Para ver as chaves de um objeto: Object.keys(objeto)
- Obs: Se eu criar um objeto A e uma variável B, e atribuir o valor de A para B, não é criada uma copia dos valores, mas sim uma referencia (quase como se fosse uma espécie de ponteiro) do objeto A.
- Destructuring: Basicamente consiste em transformar as propriedades de um objeto em variáveis, exemplo:

```
let cidade = {
  nome : "Cidade de São João",
  prefeito : "João",
  endereco : "São Paulo",
  funcao_exemplo : () => console.log(`${cidade.nome} - ${cidade.prefeito} - ${cidade.endereco}`)
}
let {nome: var_nome, prefeito: var_prefeito, endereco: var_endereco} = cidade;
console.log(var_nome)
console.log(var_prefeito)
console.log(var_endereco)
```

- Para criar um objeto a partir de outra, utiliza-se algo como: let cidade2 = Object.create(cidade); Assim, o objeto "cidade2" herda todos os métodos e atributos de cidade.
- Formas de se instanciar classes/objetos em JS:

```
// Por meio de funções:
function criarCidade(nome_cidade){
  const cidade = Object.create({});
  cidade.nome = nome_cidade;
  return cidade;
}
const cidadeA = criarCidade("São Paulo");
const cidadeB = criarCidade("Rio de Janeiro");
console.log(`${cidadeA.nome}\n${cidadeB.nome}`);

// Por meio de new:
function criarCidade(nome_cidade){
  this.nome = nome_cidade;
}
const cidadeA = new criarCidade("São Paulo");
const cidadeB = new criarCidade("Rio de Janeiro");
console.log(`${cidadeA.nome}\n${cidadeB.nome}`);
```

- Outra forma de se criar métodos dentro de objetos:

```
function cidade() {
  this.nome = "Cidade de São João",
  this.prefeito = "João",
  this.endereco = "São Paulo"
}
cidade.prototype.funcao_exemplo = function(){
  console.log(`${this.nome} - ${this.prefeito} - ${this.endereco}`);
}
let cidade_teste = new cidade();
cidade_teste.funcao_exemplo();
```

- Como criar classes em JS ES6:

```
class Cidade{
  constructor(nome_cidade){
    this.nome_cidade = nome_cidade;
  }
}
let cidadeA = new Cidade("Cidade A");
let cidadeB = new Cidade("Cidade B");
let cidadeC = new Cidade("Cidade C");
console.log(`${cidadeA.nome_cidade}\n${cidadeB.nome_cidade}\n${cidadeC.nome_cidade}`);
```

Herança em JS:

- Sintaxe exemplo:

```
class Cidade{
  constructor(nome_cidade){
    this.nome_cidade = nome_cidade;
  }
}
class SaoPaulo extends Cidade{
  constructor(){
    super("São Paulo") // Basicamente, "seta" as propriedades da classe extendida
    this.prefeito = "João Doria";
  }
}
let teste = new SaoPaulo();
console.log(teste.nome_cidade);
console.log(teste.prefeito);
```

- Pode-se verificar se um objeto é instancia de outro por meio do operador "instanceof".

Alguns métodos de strings:

- Método trim(): Remove tudo que não é string
- Método padStart(quantidade_de_vezes, elemento): Adiciona no inicio da string, um número de vezes, um determinado elemento.
- Método split(separador): Separa uma string em um array com base um separador.
- Método join(separador): Transfora um array em uma string com base em um separador.
- Método repeat(quantidade_de_vezes): Repete a string um determinado número de vezes

Operador rest (funções):

- Permite uma função receber um número indefinido de parametros.
- Sintaxe: function teste(...args){ };

Json (JavaScript Object Notation):

- Utilizado para comunicação entre back-end e front-end
- Sintaxe parecida com os objetos JavaScript
- Exemplo:

```
let teste ={
    "objeto1" : "1",
}
console.log(teste["objeto1"])
```

- Para converter Json para string usar JSON.stringify(json), e para converter uma string em Json usar JSON.parse(string)

Expressões regulares (regex):

- É utilizado para encontrar padrões em uma string.
- Para criar uma expressão regular em JS: new RegExp("expressão") ou /expressão/
- Para verificar os padrões utilizar: /expressão/.test("string_a_ser_validade")
- Para definir intervalos de caracteres: [x-y]
- Caracteres especiais: \d → Qualquer dígito (0-9) | \w → Qualquer caractere alfanumérico (letras e números) | \s → Qualquer espaço em branco | \D → Qualquer caractere diferente de número | \W → Qualquer caractere não alfanumérico | \S → Qualquer caractere que não seja espaço em branco
- Operador not (^): Negação de um determinado padrão
- Operador plus (+): Aceita indefinidamente os caracteres em uma expressão
- Operador question (?): Deixe com que um determinado padrão seja opcional
- Operador precisão ({x}): Inseri a quantidade de vezes que um determinado padrão tem de se repitir na expressão
- Choice pattern (|) : Funciona como o operador OR, permitindo assim, por exemplo, verificar entre 3 strings se uma delas é valida dentro de uma expressão
- Exemplo: const validarDominio = /www.\w+\.com|com.br/

Programação assíncrona:

- Permite a execução de ações "não sequenciais" (isto é, é possível múltiplas ações/funções ocorrendo em simultâneo)
- Callback com setTimeout: Função que executa outra função com base em um tempo X (em milissegundos). Exemplo: setTimeout(function(){codigo}, tempo).
- Promises: Ações assíncronas que podem produzir algum valor em algum momento do codígo. Exemplo:

```
let promesa = Promise.resolve(4 + 8);
console.log("Ação não sincrona");
promessa.then((x) => console.log{`A soma dos valores é ${x}`}) // Obs: é possível também retornar valores e, portanto, manipula-los
```

- Funções assíncronas (async functions): Funções parecidas com as promises(retornam as promises)
- Await: espera uma função ser executada

Interações com o WEB

- Protocolo: Forma de transferir dados dentro de uma rede
- HTTP: Protocolo de transferência de hipertexto
- SMPT: Protocolo utilizado para enviar emails
- URL: Cada arquivo carregado no navegador é nomeado por uma URL, esta, no caso, pode ser dividia em protocolo, servidor (geralmente DNS) e arquivo.
- DOM: DOM (document object model) basicamente "transforma/copia" o html da pagina em um objeto, podendo assim modificar seus atributos. Ficando, no caso, os objetos/"atributos" em formato de arvore. Tais elementos podem ser acessados através do "document", assim podendo ver os filhos através do document.childNodes, podendo especificar o indices, etc...
- Para encontrar elementos em JS: Para facilitar o acesso a elementos específicos dentro do document, é possível utilizar métodos como o getElementByTagName, getElementById, getElementByClassName e o querySelector (este no caso permite

encontrar através de uma "regra" de css, ou seja, "praticamente sem limites"). Também é possível acessar vários elementos através do querySelectorAll.

- Para alterar o html: Para fazer alterações no html (ou seja, no document em si, não em seus elementos) utilizar os métodos insertBefore, appendChild, replaceChild e createElement. Detalhe, é possível inserir texto nos elementos através de uma "mistura" entre o document.CreateTextNode("texto") e tag.appendChild(variavel_com_o_text_node).
- Para modificar e ler atributos utilizar dentro de uma tag o getAttribute("atributo") e setAttribute("atributo", "nova_propriedade")
- Para modificar o css de um elemento utilizar tag.style.propriedade = "propriedade"

Eventos com JS

- Eventos são ações acionadas a partir de alguma ação do usuário (como um clique, scroll do mouse, etc)
- Para acionar um evento é necessario atrelar um evento ao elemento (por meio do elemento.addEventListener("evento", função))
- Para remover um evento utilizar elemento.removeEventListner("evento", função)
- Propagação: Basicamente, consiste em um elemento acionar um evento destinado a outro elemento. No caso, o JS tem o metodo stopPropagation() para impedir a propagação do evento.
- Eventos de tecla: Para adicionar elementos relacionado ao clique de teclas utilizar o window.addEventListener("keydown", function(x)), podendo detectar quais teclas foram apertadas por meio de x.key | É possivel utilizar o keyup para detectar se a pessoa deixou de apertar a tecla.
- A saber as possibilidades de parâmetros dentro do eventListner: https://www.w3schools.com/jsref/dom_obj_event.asp

NodeJS

- Basicamente, é uma ferramenta que permite a execução de códigos js fora do navegador. Podendo assim, por exemplo, utilizar js em servidores.
- O node, de modo geral, surgiu para resolver o problema de outras linguagens server-side, onde cada conexão alocava um espaço na memoria. No caso, no node, cada conexão é um evento executado na engine, podendo portanto, suportar mais conexões.
- npm: Ferramenta utilizada para gerenciar pacotes (módulos) em js. No caso, para instalar pacotes utilizar npm install <pacote>
- package.json: Arquivo de configuração do projeto (onde são armazenados coisas como dependências, ou licenças por exemplo). No caso, utilizar o npm init para inicializar projetos com o Node.
- Modulo de filesystem: Modulo utilizado, basicamente, para manipulação de arquivos do sistema (como ler e editar arquivos por exemplo).
- Modulo HTTP: Modulo utilizado para fazer comunicação via HTTP.
- API (application programming interface): Uma meio de comunicação entre frontend e backend, por meio do protocolo HTTP, para fazer operações, como visualizar, deletar, criar e atualizar dados.
- Verbos HTTP: GET (solicitar dados), POST (inserir dados), DELETE (deletar dados), PUT (atualizar dados).
- Express: Framework web muito utilizado para criar coisas como API, por exemplo, com facilidade.
- Postman: Software utilizado para testar rodas de API.