**Tipos de dados**

Primitivos:

Números; String; Booleanos

O número Pi pode ser usado através do código Math.PI

Divisão por zero não é um erro em JS, ele retorna “*infinity”.* Mas 0/0 não tem um valor e o resultado é NaN

**Formatando número**

O método “Math.round() ” arredonda para o número inteiro mais próximo.

Já o “.toFixed()” controla o número de casas decimais, e arredonda usando a regra básica de arredondamento. O retorno é uma ***string.***

É possível formatar um valor para uma moeda usando o método “.*toLocaleString()”*o retorno é uma string também. Os argumentos desse método podem ser “style” que é possível usar *decimal, currency* ou *percent* outro argumento é o “currency”.

Ex.: num.toLocaleString(‘pt-BR’, { style: ‘currency’, currency: ‘BRL’})

Outros métodos de arredondamento:

- Math.ceil() -> 11.123 = 12

- Math.floor() -> 11.789 = 11

**Ordenando números**

O método sort usado puramente em uma lista de número não irá coloca-la em ordem crescente. Entretanto, o “sort” recebe opcionalmente uma função de comparação que, dados dois valores, deve devolver um número inteiro:

- Se for **0** indica que são iguais

- Se for **-1** indica que o primeiro valor é menor

- Se for **1** indica que o segundo valor é menor

Com isso usamos uma arrow function “.sort((a-b)=>a-b);

**Strings**

O padrão UTF (Unicode Transformation Format) é utilizado como padrão na web até hoje. Esse padrão cifra e decifra diversos idiomas.

Mais detalhes precisos e documentação sobre o Unicode na página da [Unicode Foundation](https://home.unicode.org/).

Método *“.toLowerCase()”*

Propriedade *“.length”*

**Convenções para nomes**

Cada linguagem tem a sua e algumas são

- camelCase **é a usada pelo *js* para variáveis e funções**

- snake\_case

- kebab-case **não pode ser usada no js para variáveis e funções**

- PascalCase

Não se utiliza espaço nem caracteres especiais, nem se inicia nomes de variáveis com números.

É importante seguir os chamados guias de estilo definidos para um produto.

**Null e undefined**

Teoricamente as duas indicam “ausência de valor”, entretanto, o *null* é um valor atribuído a uma variável que existe e foi iniciada, já o *undefined* é o valor de uma variável que foi inicializada e não foi atribuído um valor à ela. O tipo de uma variável *null* é *object*.

No cotidiano é comum considerar *undefined* como uma ausência de valor inesperada (causada por um bug ou erro no código) e *null* como um tipo de dado ‘ausente’, mas que não é inesperado. O tipo de de uma variável *undefined* é *undefined*.

**Tipos de variáveis (var, let, const)**

A *“var”* pode ser declarado mesmo depois de já termos usado a variável em si, isso porque o js lê todo o código e carrega as variáveis do tipo *var* primeiro. Isso pode ser um problema pois no mesmo código podemos acabar declarando duas variáveis *var* com o mesmo nome, dando conflito de informações. Uma variável declarada com “var” tem escopo global. *Não é usada basicamente*

A *“let”* já tem que ser declarado antes de ser usado, mas ainda assim ele permite que seja atribuído novos valores para a variável. “let” tem escopo local, não podendo ser acessada em outros escopos

A *“const”* tem que ser declarada antes e assim que for declarada tem que ser atribuído um valor para ela. Como é uma constante, o valor não pode ser alterado.

**Truthy e Falsy**

Não são de fato tipo booleanos, mas o js reconhece como tal.

0 -> false | 1 -> true

“” (string vazia) -> false

O js considerea alguns valores como *false* durante comparações, como com o *null*, *undefined*, 0, NaN e strings vazias

**Conversões**

Dois tipos, a implícita e a explicita

O js pode comparar com 3 ‘=’ que significa que leva o tipo em consideração também, se for com 2 ‘=’ o tipo não vai ser considerado e o js vai converter automaticamente um dos itens da comparação, para esse fim. Essa é a conversão implícita, que também ocorre quando somamos um número com uma string, o js converte o número para string e concatena os dois valores, ao invés de somar.

Na conversão explícita, usamos as função “*Number()”* ou *“String()”*

Outro método para converter número para string é o *“.toString”*

Um método rápido e interessante de converter string para número é colocar o operador + antes das varáveis:

let largura = “10”;

let altura = “5”;

console.log(+largura \* +altura);

Valores booleanos também podem ser convertidos para número (0=false e 1=true).

Para uma boa prática é bom sempre fazer as conversões explicitamente, sendo que o uso do operador + para converter não é muito comum, assim como converter booleanos.

**Sobreo o JavaScript**

É uma linguagem de tipagem dinâmica, ou seja, não precisamos atribuir um tipo à uma variável, na hora de declara-la, isto é, uma mesma variável pode receber um valor tipo número e depois ser reatribuido um valor tipo string. Também pode ser chamado de *untyped.*

É uma linguagem multiparadigma, o que quer dizer que podemos resolver o mesmo problema de várias maneiras diferentes, muitas linguagens são desse tipo.

ES2015 foi uma versão marcante do js que implementou diversas funcionalidades novas, e a partir desse ano começaram a sair atualizações anuais.

O js é uma linguagem interpretada, outro tipo de linguagem é a compilada. Sendo uma linguagem interpretada o código vai ser lido por um programa por trás e que vai executar o código. Em linguagens compiladas, o código primeiro é compilado e transformado em linguagem de máquina e aí sim será executado. Por ser mais fácil para o computador ler uma linguagem compilada elas são mais rápidas que as interpretadas, mas precisam antes passar pelo compilador, mas por esse quesito de rapidez elas podem ser mais eficientes para algumas aplicações.

Também é conhecido com ECMAScript, que é o nome original da linguagem.

**NodeJS**

O node por si só não é uma linguagem é uma ferramenta que interpreta o js e costuma ser usado no backend, assim como o navegador é também um interpretador de js porém, usado mais no front. O conjunto de ferramentas do node e do navegador diferem, já que eles diferem de funções. Para utilizar o node e suas funcionalidades ele precisa ser instalado.

**Erros**

*RangeError* -> o dado está no tipo certo, porém não no formato aceitável. Por exemplo, um processamento para receber números inteiros e maiores que zero e recebe -1.

*ReferenceError* -> normalmente ocorre quando o código tenta acessar algo que não existe.

*SyntaxError* -> costuma ocorrer quando há erros no programa e o js não consegue executá-lo.

*TypeError* -> o código esperava receber um dado deum tipo, como uma string, mas recebeu outro, como um número.

O NodeJS trabalha com outros tipos específicos de erro, dá para consultar na [documentação oficial](https://nodejs.org/api/errors.html#errors_errors).

*stacktrace*, também chamado de pilha, que nos auxilia a saber o que está acontecendo quando temos um erro, e onde ele se localiza.

**Console**

O *log* do *“console.log()”*  é um registro no console.

Existe também o *“console.error()”*  e no parênteses é onde colocamos a mensagem de erro.

Outros métodos do console são:

c*onsole.table()*-> usado para visualizar de forma mais organizada informações tabulares

*console.time()* e *console.timeEnd()*-> para temporizar período que uma operação de código leva para ser iniciada e concluída

*console.trace()->* exibe a *stacktrace* de todos os pontos (arquivos) por onde o código executado passou durante a execução

A [documentação oficial do NodeJS](https://nodejs.org/api/console.html) tem mais exemplos de uso.

O método *“console.error()”* apresenta uma formação no terminal igual ao “console.log”, porém se usarmos o comando *“console.error(new Error(mensagem)”*  ai sim a mensagem de erro no terminal se parece mais com um erro de verdade

**Operadores**

|| -> operador “ou”, retorna *true* caso uma condição seja válida;

&& -> operador “e”, também retorna *true*

!= e !== -> operadores “não igual” e “estritamente não igual”, utilizados para comparação da mesma forma que “==” e “===”.

operador ternário-> é basicamente um if só que feito em uma linha só e de maneira bem reduzida. Ex.: condição ? se for verdadeiro : se for falso

Se for colocar várias condições é melhor usar o if normal, o ternário por si só já é muito difícil de ler.

**Template literal**

É uma terceira forma de escrever strings, além das “”, ‘’, essa começa com `` (acento grave). No que está dentro desse acento podemos escrever normalmente e se for adicionar alguma variável, basta colocar ${} e a variável entre chaves.

**Função Math()**

- Math.trunc() -> desconsidera os decimais

- Math.pow(n1,n2) -> faz a exponenciação de 2 números. (número,expoente)

- Math.sqrt() -> retorna a raiz do número

- Math.min() -> retorna o menor valor entre os argumentos. (0, 10, -5) retorna -5.

- Math.max() -> retorna o maior valor

- Math.random() -> retorna um valor aleatório entre 0 e 1

**Parâmetro de função**

É possível passar um valor padrão para um parâmetro para caso não seja passado nenhum, na chamada da função, esse padrão seja o que vale. Para isso basta colocar = *valor padrão* no parâmetro na hora de definir a função.

Ex.: *function multiplica (n1* ***= 1****, n2* ***=1****){} //nesse caso o valor padrão para os dois parâmetros é 1*

Os nomes escolhidos para os parâmetros das funções têm escopo local, ou seja, cada um vale para a função em si.

É uma boa prática passar poucos parâmetros para as funções, se estivermos passando muitos é porque provavelmente é possível dividir a função em partes.

**Parâmetros x argumentos:**

**Na prática se referem ao mesmo tipo de dado; algumas documentações se referem a**parâmetros**no momento em que a função é definida (no caso, *numero1*, *numero2*, etc.) e**argumentos**como os dados que utilizamos para executar a função (ou seja, *30*, *45*, etc.).**

**Formas de escrever funções**

**Em js tem mais de uma forma de escrever uma função. Tem a forma mais clássica que é declarando *funcition*, tem a expressão de função e tem a *arrow function.***

**A expressão de função a gente declara uma *const* antes, para receber a função e a função em si não recebe um nome (são anônimas), elas podem receber um nome, mas normalmente não usamos. Esse tipo de função é mais curta, normalmente ocupa uma linha só. A principal diferença é que como se inicializa com *const* a função só consegue ser chamada depois de declarada, o que não acontece na declaração normal, sendo assim a declaração da vaiável com a função deve estar no topo, esse comportamento recebe o nome de *hosting.***

Outro modo de escrever função é a ***arrow function****.* Também é uma função anônima e de fato não aceita nome, é declarada com *const*, costuma ocupar uma linha só e não usa chaves ‘ {} ’. Se for usado apenas um parâmetro também não usa parênteses, porém se for mais de um parâmetro aí usa. Se a função tiver mais de uma linha aí sim usa chaves, e o *return* também. Com relação ao *hoisting* a arrow function se comportar como expressão