O JavaScript é sensacional, pois permite entregar todas as aplicações que um negócio precisa com uma linguagem.

Roda nativamente em todos os browsers

O JavaScript é simplesmente a única linguagem interpretada pelos browsers (tanto desktop quanto mobile).

JavaScript é baseado na linguagem C e é fracamente tipada

Basicamente o que aprendemos com a linguagem é manipular textos, datas e fazer cálculos com funções matemáticas.

As variáveis são usadas para armazenar dados.

As funções são blocos de código que realizam uma tarefa. Com elas reaproveitamos um código, ou seja, escrevemos uma rotina e utilizamos mais de uma vez.

Variáveis são usadas para armazenar dados

Ao utilizar var conseguimos redeclarar uma mesma variável.

Já utilizando let isso não é possível e um erro vai ocorrer.

O tipo de uma variável é determinado pelo seu valor.

Para descobrir o tipo de uma variável sem ser pelo seu valor, basta utilizar 'typeof' antes do nome da variável para imprimir no terminal.

O array é uma coleção de dados e com esse recurso podemos colocar mais de um valor em apenas uma variável.

var estados = ["Rio de Janeiro", "São Paulo", "Bahia"];

console.log(estados[0]);

Todo array começa com o índice 0, portanto para acessarmos o seu primeiro valor utilizamos nome\_do\_array[0]

Apesar de agrupar valores de tipos diferentes utilizando arrays ser possível, conseguimos fazer isso melhor com objetos, o que aprenderemos no próximo curso.

Existem casos em que precisamos saber quantos elementos tem dentro do array. Isso é feito utilizando .length depois do nome do array. Veja um exemplo no Código 10:

console.log(linguagens.length);

Com o array conseguimos armazenar em uma variável mais de um valor em comum.

No JavaScript, undefined e null podem ser utilizados para representar a ausência de valor de uma variável.

NaN é o resultado de uma operação matemática que falhou.

Uma variável será undefined quando não for atribuído um valor a ela.

É possível iniciar uma variável com null, o que significa que queremos adiar intencionalmente ou não atribuir um valor a ela, como mostra o Código 1.

Caso você ainda não saiba qual deve ser o valor de uma variável, utilize null em sua inicialização para comunicar essa incerteza.

O valor null é considerado 0 pelo JavaScript, portanto o código acima vai imprimir 1. Lembrando que esse é só um exemplo. Nos próximos cursos ganharemos mais ferramentas para não fazer esse tipo de operação.

Sempre que precisar adiar a atribuição de valor de uma variável, você pode iniciá-la com null para que ela não seja undefined.

const é usado sempre que não precisamos fazer alterações a uma variável.

Operadores são usados para modificar ou obter um novo valor a partir de uma ou mais variáveis.

Operadores aritméticos são as operações básicas da matemática.

Lembre-se que ao utilizar o operador OU, caso o valor da primeira expressão seja true, o que vier a seguir não será avaliado.

Os operadores lógicos analisam expressões ou valores booleanos e retorna true ou false.

# Objeto

Criação de um objeto literal;

var produto = {

id: 9,

nome: "Cafeteira Elétrica",

valor: 99.00

};

O objeto literal é uma estrutura que nos ajuda a organizar melhor os dados. Utilizando essa estrutura deixamos nosso código com mais qualidade, o que o torna mais fácil de ser lido e de ser mantido

As propriedades de um objeto são representadas por pares de chave/valor. Cada propriedade tem uma chave (identificador da propriedade) e um valor (conteúdo da propriedade), como vemos na Figura 1.

Um objeto literal deve ser utilizado para guardar dados de um único domínio (contexto). Se mais domínios (contextos) estiverem envolvidos, devem ser criados objetos literais para cada um deles

Um objeto literal só pode conter dados que pertencem a um mesmo domínio (contexto). Caso existam dados que são relacionados a domínios (contextos) diferentes, será necessário criar um ou mais objetos literais para guardar esses dados

Propriedades do objeto literal;

Uma propriedade de um objeto é composta por um par de chave/valor. A chave serve para identificar a propriedade e o valor define o conteúdo que está contido nessa propriedade

Acessando propriedades do objeto literal;

Coleção de objetos literais.

Para acessar a propriedade de um objeto específico dentro de uma coleção de objetos são necessárias duas coisas: A 1ª é saber qual o índice do objeto dentro da coleção e a 2ª é a chave (identificador) da propriedade que precisa ser acessada

Estruturas condicionais

estruturas condicionais

Através das estruturas condicionais podemos construir sistemas capazes de tomar decisões sozinhos

if / else;

A estrutura condicional if / else é um recurso poderoso na criação de todo tipo de sistema

O operador == deve ser utilizado com cuidado quando for comparar certos tipos de dados

O uso de igualdade estrita em alguns casos é essencial para o bom funcionamento de um sistema

O if ternário permite ao desenvolvedor criar uma estrutura condicional de if / else de forma simplificada

Utilizar curto-circuito permite ao desenvolvedor implementar condicionais if com muito menos linhas de código

else if;

switch.

Nota: Expressões booleanas também podem ser utilizadas como parâmetro do switch, entretanto, seu uso não é recomendado, sendo mais indicado o uso da estrutura condicional if, que funciona especificamente com expressões booleanas

A estrutura condicional switch é um recurso que, assim como o if / else, pode ser utilizado em qualquer tipo de sistema

switch(produto)

{

case "Smartphone":

case "Celular":

case "Telefone":

console.log("Produto: Smartphone");

break;

case "TV":

console.log("Produto: TV");

break;

default:

console.log("Produto inválido");

break;

}

Inputs

Durante o desenvolvimento de aplicações JavaScript é muito comum termos que criar aplicações que precisem de interação com o usuário e muitas das vezes essa interação precisa ser feita através de um ambiente externo. Nesta aula veremos o que são inputs e como eles nos ajudam a resolver essas situações.

A utilização do input nos permite construir aplicações de linha de comando dinâmicas autossuficientes em JS. Essas aplicações podem ser usadas em diferentes contextos, como teste de aplicações direto na linha de comando e até mesmo aplicações de linha de comando completas

Na aula anterior vimos como o uso de inputs pode ajudar na construção de aplicações de linha de comando com JS. Porém, esse não é um recurso nativo da linguagem, por isso, nesta aula vamos aprender a instalar e configurar um plugin que adiciona o recurso de input ao JS.

Instalando o readline-sync:

primeiro: inicializar o npm (npm init -y)

sera criado um arquivo .json dentro da pasta selecionada que precisara ser editado adicionando uma linha ao codigo ("type": "module")

segundo instalar o modulo do readline-sync (npm install readline-sync)

serão criados varios arquivos.

depois de tudo instalado será necessario chamar o plugin dentro do programa.

---

import entradaDados from 'readline-sync';

const nome = entradaDados.question('Digite seu nome: ');

console.log('Olá, '+nome);

---

Nota: Devido a uma limitação do Windows, caracteres acentuados podem não ser exibidos corretamente no prompt de comando (cmd). Esse não é um problema do readline-sync e sim do Windows. Para resolvê-lo, execute o comando chcp 65001 no prompt de comando e a aplicação passará a exibir a acentuação corretamente.

O readline-sync é o recurso que utilizamos para implementar inputs em nossos projetos JS no node. É através dele que poderemos construir nossas aplicações dinâmicas de linha de comando. Por exemplo:

Conhecer bem o readline-sync é importante no processo de desenvolvimento de aplicações de linha de comando, pois é através dele que implementamos os inputs que nos permitem a entrada de dados diretamente no console

a função Number() só é necessaria em operações matematicas de soma. as demais operações o JS faz a conversão de type antes de realiza-las.

O uso da função Number() deve ser utilizado sempre que formos utilizar valores numéricos do input em cálculos matemáticos. Por exemplo:

let x = entradaDados.question("Informe o valor de x: ")

let y = entradaDados.question("Informe o valor de y: ")

let soma = Number(x) + Number(y);

Através de exemplo práticos, vimos como podemos utilizar o input em aplicações do mundo real, deixando mais claro o uso do input com readline-sync

---

import readlineSync from 'readline-sync'

;

let

colecao\_empresas

= [

{id: 1, nome: "Microsoft"},

{id: 2, nome: "Apple"},

{id: 3, nome: "Alphabet"},

{id: 4, nome: "Facebook"}

];

console.log("\n1 - Microsoft")

console.log("2 - Apple")

console.log("3 - Alphabet")

console.log("4 - Facebook")

let empresa = readlineSync.

question

("\nDigite o id da empresa que deseja: ");

console

.log("\nVocê escolheu a empresa: "+colecao\_empresas[empresa-1].

nome

);

Arreys

Adicionando elementos no array

Vimos que o array é um tipo de dado muito importante, pois serve para reunir dados de um mesmo contexto dentro de uma variável. Arrays são utilizados em todo tipo de projeto, sendo essencial para o programador conhecer bem como trabalhar com esse tipo de dado.

O array é uma estrutura onde podemos reunir diversos dados de um mesmo domínio. Utilizar essa estrutura deixa o código mais organizado e fácil de manter

Como vimos, o array é utilizado para agrupar elementos dentro de uma variável. E para acessar cada elemento é necessário utilizar um índice, que funciona como uma posição numérica dentro do array. Vamos ver como isso funciona no flow a seguir:

Ao trabalhar com arrays nunca se esqueça: o primeiro elemento de um array está na posição 0

Para acessar o último elemento de um array é necessário obter o tamanho do array e subtrair uma unidade. Por exemplo, se o array tem 10 unidades, o último elemento desse array será acessado através do índice 9

O índice para acessar o primeiro elemento de um array deve ser sempre 0. Já o índice utilizado para acessar o último elemento do array é igual à quantidade de elementos do array subtraída por uma unidade. Se o array tem 5 elementos, o último deles será acessado pelo índice 4.

Quando precisamos alterar um elemento de um array, será necessário utilizar o índice desse elemento para essa tarefa. É preciso usar o índice, pois com ele identificamos a posição que o elemento ocupa dentro da estrutura.

A propriedade length é muito importante quando trabalhamos com arrays, pois com ela não precisamos nos preocupar com o total de elementos que um array possa ter

Parabéns! Você aprendeu a trabalhar com arrays na linguagem JavaScript. Com arrays podemos representar diversos dados agrupados tais como meses do ano, notas bimestrais, um conjunto de funcionários etc. Isso mostra que essa estrutura de dados é muito utilizada para reunir dados em qualquer tipo de projeto.

estruturas de repetição

As estruturas de repetição nos permitem executar um mesmo bloco de código quantas vezes forem necessárias, mesmo que não saibamos quantas serão. Com elas, podemos obter diversos resultados diferentes executando um mesmo trecho de código

Na linguagem JavaScript temos uma estrutura de repetição com uma sintaxe simples que serve para repetir blocos de código enquanto uma condição for atendida. Essa estrutura de repetição é conhecida como while, como vemos no flow a seguir:

while

let capital\_emprestado = 3000;

let taxa\_juros = 0.02; // 2%

let contador = 1;

let parcelas\_totais = 5;

while ( contador <= parcelas\_totais ) {

let numero\_parcela = contador;

let juros\_emprestimo = capital\_emprestado \* taxa\_juros \* numero\_parcela;

console.log("A taxa de juros no período de " + numero\_parcela + " mes(es) é de R$" + juros\_emprestimo + ",00");

contador++;

}

Independentemente de usar incremento, decremento ou outro recurso no script, uma coisa fundamental nas estruturas de repetição é que elas sejam finalizadas em algum momento. Por isso normalmente usamos um contador para essa finalidade.

Para que o loop while funcione corretamente é necessário usar um contador que deve ser incrementado dentro do bloco. Esse contador é o mesmo presente na condição dessa estrutura de repetição, e sem ele o bloco não funciona da forma esperada

Para que o loop while funcione corretamente é necessário usar um contador que deve ser incrementado dentro do bloco. Esse contador é o mesmo presente na condição dessa estrutura de repetição. Sem esse contador, o bloco não funciona da forma esperada

Caso seja necessário iterar todos os elementos da coleção, use o comando continue. Agora se o objetivo for encontrar apenas um único elemento que atenda a uma determinada condição, o comando break é a melhor opção.

Para interromper todo o bloco de repetição utilize o comando break e para interromper apenas uma iteração utilize o comando continue

do/while

et contador = 0;

do {

console.log(contador);

contador++;

} while ( contador < 5 );

----

let funcionarios = [

{id: 1, nome: "César", habilitado: false},

{id: 2, nome: "Jéssica", habilitado: false},

{id: 3, nome: "Renan", habilitado: true},

{id: 4, nome: "Marlon", habilitado: false},

{id: 5, nome: "Ana", habilitado: false},

];

let contador = 0;

let encontrouHabilitado = false;

do {

let funcionario = funcionarios[contador];

if( funcionario.habilitado == true ){

console.log("Funcionário habilitado encontrado: " + funcionario.nome);

encontrouHabilitado = true;

break;

}

contador++;

} while ( contador < funcionarios.length );

if (!encontrouHabilitado) {

console.log("Nenhum funcionário habilitado encontrado");

}

// Funcionário habilitado encontrado: Renan

----

for

A estrutura de repetição for já tem um contador na sua sintaxe, não sendo necessário criar um, como é feito na estrutura while

pesquisar sobre

for .. in

pesquisar sobre

Quando for necessário percorrer as propriedades de um objeto, utilize a estrutura de repetição for .. in para resolver esse problema

for .. of

pesquisar sobre

let produtos\_array = [

{ id: 1, nome: "mouse", preco: 50 },

{ id: 2, nome: "teclado", preco: 90 },

{ id: 3, nome: "monitor", preco: 350 },

{ id: 4, nome: "nobreak", preco: 600 },

];

let percentual\_desconto = 10;

for ( let produto of produtos\_array ) {

let nome = produto.nome;

let preco = produto.preco;

let valor\_desconto = (preco \* percentual\_desconto) / 100;

let novo\_preco = preco - valor\_desconto;

console.log("Novo valor do produto " + nome + ": R$" + novo\_preco + ",00");

}

A estrutura de repetição for .. of é uma ótima solução para percorrer valores de uma coleção de dados

O tipo string é muito utilizado nos sistemas atuais e é essencial que um programador saiba como manipulá-lo.

Toda string possui métodos e propriedades nativas

As vantagens do uso desses recursos nativos são:

Redução de código repetitivo;

Economia de tempo;

Aceleramento no processo de desenvolvimento.

Na manipulação de strings é fundamental usar propriedades e métodos nativos, pois isso aumenta a produtividade do programador

Uma string pode ser usada para formar valores em novas strings

Utilizar acentos graves para criar uma string (template string) possui vantagens com relação as aspas simples ou duplas. Uma delas é criar uma string que possui um texto com mais de uma linha. Veja um exemplo no flow abaixo:

console.log(`Estude Node.js para programação ${ stack } em ${ linguagem }`);

Utilizar template string melhora a legibilidade permitindo inserir variáveis junto com o texto, ou seja, tudo dentro de acentos graves

A propriedade length é bastante usada para validação de dados pois com ela é possível verificar o tamanho das strings, o que evita problemas como strings vazias ou com tamanho inadequado

.toUpperCase()

.toLowerCase()

Quando os dados tiverem sensibilidade à diferença entre caracteres maiúsculos e minúsculos será necessário usar os métodos nativos para resolver esse problema

É comum que algumas strings tenham espaços em branco no início e/ou no final. Como esses espaços normalmente são desnecessários, devem ser removidos deixando apenas o conteúdo da string. No JavaScript temos um recurso nativo que cuida dessa tarefa. Veja no flow a seguir:

Geralmente o método trim() é usado no cadastro e na validação de dados

Com os métodos padStart() e padEnd() é possível alinhar informações com a linguagem JavaScript de forma que a exibição dos dados não fique desordenada

O método replace() é a melhor forma de substituir um termo dentro de um texto na linguagem JavaScript

O método substr() foi descontinuado e os navegadores em breve não vão dar mais suporte a ele, por isso, o substr() não deve ser mais utilizado em nenhum projeto.

Ao invés disso, sempre que precisar retornar uma parte de uma string você deve utilizar o método substring().

Entretanto, é importante que você entenda o funcionamento do substr, pois muitos projetos antigos o utilizam, e apenas conhecendo o funcionamento dos dois métodos você poderá atualizar seus códigos antigos que utilizam substr() para usarem substring() corretamente.

Quando o método indexOf encontra a substring dentro de uma string, ele retorna a posição onde a substring foi encontrada. Caso não encontre ele retorna o valor -1.

O método indexOf é um ótimo recurso da linguagem JavaScript para encontrar termos simples dentro de textos grandes

let paises\_string = "Brasil Espanha Canadá Irlanda Suíça";

let paises\_array = paises\_string.split(" ");

console.log(paises\_array); // (5) ["Brasil", "Espanha", "Canadá", "Irlanda", "Suíça"]

O método split() aceita um parâmetro que tenha mais de 1 caractere para quebrar strings, porém é mais comum o uso de um único caractere para essa tarefa.

O método split() é muito útil para extrair dados de strings maiores. Assim é importante saber como é a composição da string para identificar o caractere que será usado como separador

Date nos permite trabalhar com data e hora na linguagem JavaScript

Agora que vimos que Date é um objeto nativo que possui informações sobre uma data, no Flow abaixo você vai aprender porque e quando Date é útil nas nossas aplicações

Através do objeto do tipo Date conseguimos comparar duas datas

const hoje = new Date( );

// Sem parâmetro: Data atual

const hoje= new Date();

// Com parâmetro: String com a data

const exemplo1 = new Date( "2020-02-28" );

const exemplo2 = new Date( "2020-02-27 10:35:00" );

// Com parâmetro: Um número para ano, mês, dia, hora, minuto, segundo e milissegundo

const exemplo3 = new Date( 2020, 2, 28, 13, 20, 2, 15 );

Podemos criar um objeto do tipo Date utilizando um parâmetro para definir uma data específica, ou então não passar um parâmetro e receber a data atual do sistema

A data UTC é representada no código a seguir:

1970-01-01T00:00Z

UTC é o tempo universal que serve de base para os fusos horários

Para saber a diferença de fuso horário do sistema para o UTC utilizamos o método .getTimezoneOffset()

Os métodos toLocaleString, toLocaleDateString e toLocaleTimeString permitem retornar uma string personalizada. Isso é feito através de dois parâmetros que podem ser vistos no Flow abaixo:

As opções de idiomas não são suportadas em todos os dispositivos. Por exemplo no Node o idioma pt-BR não está disponível