escola britânica de artes criativas & tecnologia

Desenvolvedor Full Stack Python

Fundamentos do JavaScript

escola britânica de artes criativas & tecnologia

Conteúdo do módulo

- Sintaxe
- Variáveis e constantes
- Tipos de dados
- Laços
- Condicionais
- Funções

Sintaxe

escola britânica de artes criativas & tecnologia

Conteúdo da aula

- Sintaxe do JavaScript
- Comentários

Sintaxe do JavaScript

Como linguagem de programação o JavaScript segue algumas diretrizes:

É case sensitive: duas palavras escritas com o mesmo texto mas com a caixa diferente, são diferentes, exemplo: nomeDoMeio e nomedomeio.

Existência de palavras reservadas: existem algumas palavras que não podemos usar na hora de nomear variáveis ou criar funções, pois são reservados pelo interpretador do JavaScript.

Sintaxe do JavaScript

Uso de ponto e vírgula ao final de uma sentença: em algumas linguagens de programação o uso de ponto em vírgula ao final de uma sentença de código é obrigatório, já foi assim no JavaScript, atualmente o uso do ponto e vírgula é opcional.

Comentários – Sintaxe

No JavaScript podemos escrever comentários por linha e comentários em bloco, que podem conter várias linhas.

Comentários em linha: para comentar uma única linha usamos o sinal //

```
// sou um comentário
```

Comentários em bloco: para comentários em bloco, começamos com: /* e determinamos em */

```
/*
Também sou
um comentário
*/
```

Variáveis e constantes

escola britânica de artes criativas & tecnologia

Conteúdo da aula

- Variáveis
- Constantes
- Nomeando variáveis e constantes

Variáveis

Variáveis são como caixas que armazenam um valor.

Para criar uma variável no JavaScript podemos usar as palavras-reservada var e let.

**let pode não funcionar em navegadores mais antigos, esse recurso foi introduzido em 2015.*

Nossa primeira variável:
Let nome = "Gian"

Composição de uma variável:
Let ou var NOME_DA_VARIAVEL = VALOR

Variáveis

Podemos alterar o valor da variável simplesmente escrevendo o seu nome e atribuindo (=) o novo valor:

```
nome = "Gian Souza"
```

O JavaScript não possui tipagem, isso quer dizer, se quisermos sobrescrever o exemplo anterior, com:

```
nome = 32
```

Constantes

Constantes assim como as variáveis são containers de valor.

A grande diferença além do uso da palavra reservada const, é que uma vez inicializada não poderá ter o seu valor trocado.

```
const cpf = 12345678910
cpf = 123 // irá gerar um erro
```

Nomeando variáveis e constantes

Podemos escrever o nome das variáveis seguindo alguns padrões de nomenclatura:

Underscore: var primeiro_nome= “Gian”
PascalCase: var PrimeiroNome= “Gian”
CamelCase: var primeiroNome= “Gian”

*Não podemos começar uma variável ou constante com um número, caracteres especiais ou usar traço hífen no meio das palavras.

Tipos de dados

escola britânica de artes criativas & tecnologia

Conteúdo da aula

- Sobre o tipo de dados no JavaScript
- Literais
- Lógicos
- Numéricos
- Objetos

Sobre o tipo de dados no JavaScript

O JavaScript não é uma linguagem fortemente tipagem, isso quer dizer que na hora de declaramos uma variável por exemplo, não preciso informar que será um texto ou número.

Mas, isso gera alguns problemas, como por exemplo, uma variável que originalmente era numérica em determinado momento do programa pode se tornar um número.

Os tipos de dados que temos disponíveis no JavaScript são:

String: tipo literal, usado para textos e caracteres;
Number: tipo numérico, usado tanto para números inteiros e reais;
Boolean: tipo lógico, que indica se determinada afirmação é verdadeira ou falsa;
Object: tipo mais básico do JavaScript, todos os tipos nasceram a partir do Object.

Literais - String

O tipo String, é usado quando trabalhamos com caracteres, até mesmo um espaço em branco é uma **String.**

Podem ser construídas ao envolver o valor atribuído a uma variável em aspas, simples ou duplas.

```
var nome = "gian"
Nome = 'gian'
```

Literais - String

Podemos usar alguns recursos da String para contar o seu tamanho, selecionar apenas um pedaço ou procurar por um pedaço de texto ou uma letra.

Consideramos:
var nome = "JavaScript"

Contando o tamanho de uma String
nome.length// 10

Tornando todo o texto em caixa alta
nome.toLocaleUpperCase() // JAVASCRIPT

Literais - String

Procurando por uma letra ou palavra:
nome.search('v') // 2
O 2 indica a posição da letra V em JavaScript, naturalmente seria 3, mas a posição inicial é 0 e não 1

nome.slice(0, 2) // Já
Retorna um pedaço da String, 0 sendo a posição inicial e 2 o número de caracteres que queremos selecionar.

Unindo Strings

Podemos unir Strings + Strings ou Strings + Números, com o operador **+**

"Olá " + nome // Olá JavaScript
nome + " é " + 10 + "!" // JavaScript é 10!

Chamamos essa união de **concatenação.**

Lógicos – Boolean

Os valores do tipo lógico são compostos por **true** ou **false**, através desses valores podemos mudar o fluxo de execução do software.

```
var foiConvidado = true
var podeEntrar = foi Convidado
```

No exemplo acima temos uma situação onde se pode entrar num local apenas se for convidado, a primeira variável diz se a pessoa foi convidada, e como o valor da segunda é condicionada a ela, podemos simples repassar seu valor, logo:

```
foiConvidado = true ENTÃO podeEntrar = true
foiConvidado = false ENTÃO podeEntrar = false
```

Numéricos - Number

Para criar uma variável do tipo numérico em JavaScript, basta atribuir o número ao valor.

Apesar de tanto o número inteiro como o fracionado serem considerados ambos do tipo number, existe uma separação interna entre o tipo **integer (inteiro)** e **float (fracionado)**:

```
var idade = 32 // Numbermas Integer
var altura = 1.72 // Numbermas Float
```

Numéricos - Number

Com os números podemos fazer cálculos aritméticos:

```
var ano = 2022
var idade = 32
idade * 2 = 64
idade / 2 = 16
```

```
var anoDeNascimento = ano –idade
```

Podemos usar operações para atribuir o valor às variáveis.

Numéricos - Number

Assim como nas Strings os elementos do tipo **Number** possuem alguns recursos especiais, como:

Verificar se um número é **Integer.**

```
Number.isInteger(10) // true
Number.isInteger(10.5) // false
```

Converter uma String para **Float:**

```
parseFloat("10.5632") // 10.5632
parseFloat("10,5632") // 10 – não devemos usar vírgula
```

Converter um Float para **Integer:**

```
parseInt(10.43) // 10 –o restante do número é descartado
```

Objetos – Object

O tipo **Object** é o mais primitivo do JavaScript, através dele outros tipos nasceram.

Um ponto importante é que o JavaScript trata os **vetores como objetos.**

Vetores são conjuntos de dados, o conjunto é representado pelos símbolos de colchetes ([.....]), e seus itens são separados por vírgula.

Também são chamados de arrays.

Objetos – Array

Exemplo de um **array** de **Strings**
var nomes = [“gian”, “carlos”, “maria”]

Por possuir uma fraca tipagem o JavaScript não nos obriga a adicionar itens do mesmo tipo em um conjunto, por exemplo:
nomes = [“gian”, “carlos”, “maria”, “joão”, 15, 32]

Para acessar um item do **array** usamos seu índice, onde o primeiro item possui o índice 0.
nomes[0] // gian
nomes[4] // 15

Laços

escola britânica de artes criativas & tecnologia

Conteúdo da aula

- Sobre laços
- Laço for
- Laço for in
- Laço While
- Laço do while

Sobre laços

Laço é o nome que damos quando queremos percorrer um conjunto de itens, também chamamos de loop ou iteração.

Podemos ter um conjunto com os nomes dos ganhadores de uma promoção:

```
var ganhadores = [“Gabriela”, “João”, “Luana”]
```

Sem o uso de um laço, teríamos que acessar o índice de item no conjunto:

```
ganhadores[0] // Gabriela
ganhadores[1] // João
ganhadores[2] // Luana
```

Laço for

O tipo de laço mais comum é o for, sua estrutura é:

```
for (var i = 0; i < qtdde itens; i++) {
// ...
}
```

Onde:

var i = criamos uma variável que irá contar as vezes em que “entramos” no laço

Qtd de itens = quantas vezes queremos entrar no laço, geralmente é o tamanho do conjunto, que pode ser obtido através de ganhadores.length, por exemplo.

i++ = Informamos que a cada vez que o laço for acionado o valor de i deve ser incrementado + 1.

< = Será executada apenas quando i for menor que a qtd de itens

Laço for

Exemplo prático:

```
var ganhadores = [“Gabriela”, “João”, “Luana”]

for (var i = 0; i < ganhadores.length; i++) {
    console.log(“Parabéns: “ + ganhadores[i]);
}
```

```
Parabéns: Gabriela
Parabéns: João
Parabéns: Luana
```

Desta forma não precisamos nos preocupar em informar o índice do elemento, basta informar o i entre colchetes que ele terá o valor atualizado do item atual.

Laço for in

O laço for in é um meio mais simples de executarmos o laço for, nessa abordagem recebemos o índice do item em uma variável.

Exemplo considerando o conjunto:

```
ganhadores = [“Gabriela”, “João”, “Luana”]

var saudacoes= 0;
for (ganhador in ganhadores) {
console.log(“Parabéns: ” + ganhadores[ganhador]);
}
```

Laço while

Outra forma de iterarmos um conjunto de dados é através do laço while, sua estrutura é:

```
while(condição verdadeira) {
// ...
}
```

Enquanto (while) uma condição for verdadeira o código dentro do bloco será executado.

Laço while

Exemplo considerando o conjunto:

```
ganhadores = [“Gabriela”, “João”, “Luana”]
var saudacoes= 0;
while(saudacoes< ganhadores.length) {
    console.log(“Parabéns :” + ganhadores[saudacoes]);
     saudacoes++;
}
```

Enquanto o número de saudações for menor que a quantidade de itens o bloco será executado, ao final de cada o número de saudações será incrementado para que no final da terceira saudação saudações = 3 e isso já não satisfaz mais a condição do laço **while.**

Laço *do while*

O laço do **while** é bem parecido com o laço **while**, apenas inverte o bloco de execução com a condição:

```
do {
// ...
} while(condição verdadeira);
```

Resumidamente, execute esse bloco de código enquanto a condição for verdadeira.

Condicionais

escola britânica de artes criativas & tecnologia

Conteúdo da aula

- Sobre condicionais
- If
- Else
- Ifelse
- Ternário

Sobre condicionais

Com o uso das condicionais podemos executar diferentes fluxos de instruções.

Em conjunto com os verificadores temos os operadores lógicos e os comparadores.

Comparadores

< Menor que
5 < 4 // false

<= Menor ou igual que
2 <= 3 // true

> Maior que
5 > 5 // false

>= Maior ou igual que
5 >= 5 // true

== Igual a
5 == "5" // true

=== Igual a // compara além do conteúdo o tipo do dado
5 === "5" // false

!= Diferente
5 != "5" // false

de !== Diferente de
5 !== "5" // true

! Negação
!true// false, estamos negando o true, logo false

Operadores lógicos

Com os operadores lógicos conseguimos construir uma cadeia de condicionais.

No Java Script possuímos os operadores && (E), || (Ou) e ! (Negação).

&& E

Com esse operador todos os termos devem ser verdadeiros para a condição ser verdadeira.
true&& true// true
5 > 10 && true// false, cinco não é maior que 10

|| OU

Com esse operador apenas um dos termos precisam ser verdadeiros para a condição ser verdadeira.
true|| false // true
5 > 10 || true// true, o termo da direita é verdadeiro

! Negação

Com esse operador negamos um termo.
!false // true, a negação de um termo falso é ele ser verdadeiro
!true// false

IF

Com o uso do IF (se) verificamos uma condição e se ela for verdadeira o código dentro do IF será executado:

```
If (condição) {
// ...
}
```

Else

Com o uso do Else (senão) podemos criar um fluxo alternativo para quando a condição do IF não for atendida:

```
If (condição) {
// ...
} else{
// ...
}
```

Logo, se a condição for false, o código executado será o dentro do bloco else.

Else if

Com o **else if** podemos criar outras condições para tratar nosso fluxo de execução, quando a condição inicial não é satisfeita.

Para ilustrar, vamos ver uma instrução onde só se pode entrar se tiver mais ou 18 anos ou menor com acompanhante.

```
If (idade >= 18) {
   podeEntrar = true
} else if (estaAcompanhado) {
   podeEntrar = true
} else {
   podeEntrar = false
}
```

Operador ternário

Com o operador ternário podemos executar o IF e o ELSE em apenas uma linha, isso é útil quando o valor de uma variável depende de alguma condição, por exemplo:

```
var eFeriado= true
var saudacao= eFeriado ? "Estamos fechados hoje" : "Seja bem-vindo";
```

Com o ternário o IF é executado após o sinal **?** E o else após o :

Funções

escola britânica de artes criativas & tecnologia

Conteúdo da aula

- Sobre funções
- Criando uma função
- Parâmetros
- Retornando dados

Sobre funções

Uma função, também pode ser chamada de método ou procedimento, **é uma forma de separamos partes específicas do código e que podemos reutiliza-las.**

Podem receber parâmetros, também chamados de argumentos e podem retornar algum valor ou não.

Criando uma função

Temos dois caminhos para criar uma função no JavaScript: **através de uma variável ou usando a palavra reservada function.**

```
var nome = "JavaScript"
function dizOi( ) {
   console.log("Oi, " + nome);
}

var dizOi = function( ) {
   console.log("Oi, " + nome);
}
```

Para executar a função escrevemos o nome dela seguido de **( ), dizOi()**.

Podemos dizer que estamos chamando ou invocando um função.

Parâmetros

No exemplo anterior na mensagem de saudação estamos concatenando com o nome que é uma variável declarada antes da função, estamos usando um recurso externo.

Com o uso de parâmetros podemos tornar nossa função independente de recursos externos.

```
function dizOi(nome, cidadeNatal) {
    console.log("Oi, " + nome + " de " + cidadeNatal);
}
```

Retornando dados

Até aqui vimos construir uma função e consumir dados de entrada (inputs), mas com as funções também podemos retornar dados (outputs). Fazemos isso com a palavra reservada return.

Um ponto importante é que todo o código escrito após o return não será executado.

Exemplo com return:

```
function numeroAoQuadrado(numero) {
   return numero * numero;
}
```

Podemos usar o retorno para atribuir o valor a uma variável, por exemplo:

```
var num1 = 4;
var num2 = numeroAoQuadrado(num1); // 16
```