JavaScript

tipos primitivos

JS

JavaScript

parte 4

Alan Alencar, 2023

aprendendo Javascript

agenda

iremos aprender quais tipos de variáveis é possível se trabalhar com javascript e como podemos fazer uso delas para os nossos programas.

quais tipos posso usar?

primitivos

via de regra, um tipo primitivo é todo e qualquer tipo de dados que uma linguagem de programação é capaz de reconhecer que não são objetos. logo, por consequência disto não possui métodos em suas especificações.

no Javascript são reconhecidos 6 tipos primitivos.

1. ***string.***
2. ***number.***
3. ***boolean.***
4. ***null.***
5. ***undefined.***
6. ***Symbol***

mas no Javascript, os tipos String, Number, Boolean e Symbol, todos eles possuem um wrapper que nos permite acessa-los em uma notação de objetos e isto é muito interessante.

regras

uma variável ou “identificador" em Javascript deve obedecer as seguintes bases:

1. ***iniciar obrigatoriamente com letra, _ (underline) ou $ (cifrão) e não pode conter espaços.***
2. ***posteriormente pode conter números e quaisquer caracteres unicode.***
3. ***não é preciso dizer qual é seu tipo na criação.***
4. ***não pode ter o mesmo nome das palavras reservadas da linguagem. Ex: if, for, switch, let.***

vale recordar que assim como a sintaxe da linguagem as variáveis também são case sensitive.

tipagem

linguagem usa inferência

em JavaScript é seguro afirmar que por ser uma linguagem interpretada, ela trás consigo uma característica de ser de tipagem fraca.

o termo é devido ao fato de as declarações de variáveis não exigir que seja informado qual é o tipo de dado que ela vai tratar, mas o motor do JS automaticamente escolhe para nós baseado na natureza do dado que vamos armazenar.

ao lado a diferença entre a declaração de variáveis na linguagem JS e C.

```javascript
// Javascript
let empresa = 'Terra Vista Ltd'; // string
let salario = 3546.87; // number
let gratuito = true; // bool
```

```c
// C
float salario = 3546.87; // ponto flutuante
char caractere = 'A'; // char (1 byte)
int numero = -61; // inteiro
unsigned int num_positivo = 61; // inteiro positivo
```

inferência

inferência de tipo

```
// Iniciando variáveis em JS para determinar seu tipo por inferência.

let nome = 'Alan'; // por inferência será do tipo string.
let alturaMts = 1.85; // por inferência será do tipo number.
let eAtleta = false; // por inferência será boleano.
let eCalvo = null; // por inferência não tem valor nem tipo ainda.
let esportePredileto = undefined; // por inferência não terá um tipo.
```

chamamos de **inferência** de tipo a forma com que o JavaScript interpreta o conteúdo de inicialização de uma variável para determinar qual será seu tipo. Desta forma, quando iniciamos uma variável, o JS vai por inferência analisar o conteúdo e determinar qual é o melhor tipo para tratar aquele dado específico. como não é uma linguagem tipada, este mecanismo é providencial para conseguir efetuar uma série de comandos como aprenderemos mais tarde.

string
primitivo string

```
// as variáveis do tipo string contém textos entre aspas simples ou duplas.
let primeiroNome = 'Alan';
let sobrenome = "Alencar";
let restaurant = "Joe's Crab"; // conteúdo da string contém aspas simples.
let et = '👽'; // pode conter um emoji e ainda assim é uma string.
let japaneseGirl = '逃げる' // tipos unicode.
let telefone = '\u260E'; // contém ☎ (emoji telefone)
```

um tipo primitivo `string` em JavaScript é usado para armazenar uma sequência de caracteres, sejam eles alfanuméricos ou numéricos, além de caracteres especiais e Unicodes. *atenção porque é possível guardar números só que como strings*.

o seu conteúdo deve estar obrigatoriamente expresso entre aspas simples o aspas duplas, e isto não mudará em nada seu comportamento. quando o conteúdo do tipo contiver aspas simples, você colocá-lo entre aspas duplas e vice-versa como mostrado no código acima.

por fim, é possível associar um emoji ou qualquer unicode válido.

number

primitivo number

```
// as variáveis do tipo number podem conter inteiros ou floats.
let qtdRodas = 4;
let pesoKG = 16.12; // separador de decimais é sempre ponto (.)
let distanciaMts = 145;
let temperaturaInverno = -5.89; // negativo.
let octalNum = 0601; // 385 em decimal.
let hexaNum = 0x3D; // 61 em decimal.
let binNum = 0b11110110001; // 1969 em decimal.
```

um tipo primitivo `number` (número) pode conter tanto valores inteiros quando decimais (float), positivos ou negativos. não é necessário definir se tem ou não decimais, basta apenas associar o valor e ambas sempre serão number. a precisão de um tipo number é de Double-precision floating-point format, ou um float 64 bits.

importante: o separador de decimais é sempre **ponto** (.) e não é necessário usar separadores de milhares.

números binários inicia com 0b (zero e bê), Octal inicia com 0 (zero) e hexadecimal com 0x (zero e xis).

boolean

primitivo bool

```
// as variáveis do tipo boolean podem conter true ou false.
let maiorIdade = true; // verdadeiro (em minúsculo).
let aprovado = false; // falso (em minúsculo).
```

um tipo primitivo `boolean` (bool) só pode conter dois valores possíveis. **true** ou **false** e sempre em minúscula. Não é possível associar 0 (zero) ou 1 (hum) para representar booleanos. Neste caso ele seria um number.

null
primitivo null

```
// tipos null (nulos) em JavaScript precisa ser declarado e sinalizado.
let decisao = null;
let nulo = null; // a variável recebe null por definição.
```

um tipo `null` para o JavaScript é uma situação quando uma variável ainda não reconhece bem sua característica e portanto aponta para um segmento de memória inexistente. neste caso para o JavaScript ele é um object (falaremos muito sobre object mais adiante.

salvo você tenha um controle bem feito no seu programa, usar nulls (nulos) não é uma boa prática, ainda que com JavaScript isto não represente um cenário tão catastrófico como seria em outras linguagens como o C por exemplo.

undefined

primitivo undefined

```
// tipo undefined é quando você não dá um valor inicial a sua variável por exemplo
let nome; // sem dar um valor inicial, o JS vai considerar a variável como indefinida.
let qualquerCoisa;
```

um tipo `undefined` no JavaScript é uma situação quando uma variável é declarada dentro do seu programa, porém ela ainda não recebeu nenhuma informação para saber suas características. logo, para todos os efeitos ela é tratada como indefinida, ou undefined.

o uso de undefined vai permitir que as decisões sejam bem avaliadas quando você precisar trabalhar com objetos ou tipos inexistentes.

qual é o tipo da variável?

operador typeof

um operador muito importante e útil no JavaScript é o `typeof`. com ele é possível efetuar uma checagem de qual é o tipo de uma determinada variável, garantindo mais segurança aos nossos algoritmos. é bem simples de usar e sempre que for necessário trazer segurança, use-o.

como o JavaScript trabalha com o conceito de inferência, ou seja, o valor atribuído é o tipo que ele associa a variável e por não ser uma linguagem de programação tipada (*weakly-typed*), que requer dizer qual é o tipo no momento em que a criamos, em várias situações uma variável pode nascer com um tipo e alterar para outro causando sérios bugs aos nossos programas.

```
// operador typeof em ação
let nome = 'Alan';
let sobrenome = "Alencar";
let idade = 53;
let cargo = null;
let salario;
let developer = true;
let cachorro = '🐶';

console.log('nome é do tipo', typeof nome); // string
console.log('idade é do tipo', typeof idade); // number
console.log('cargo é do tipo', typeof cargo); // null é um object
console.log('salário é do tipo', typeof salario); // undefined
console.log('developer é do tipo', typeof developer); // boolean
console.log('cachorro é do tipo', typeof cachorro); // string

salario = 100.61; // passou a receber r$ 100,61
cargo = 'Analista de Sistemas'; // agora a variável recebeu um valor.
console.log('agora salário é do tipo', typeof salario); // number
console.log('agora cargo é do tipo', typeof cargo); // string

if (typeof developer == 'boolean' && developer) {
    console.log('E é um desenvolvedor / programador de computadores');
}
```

fica a dica

cuidado com os operadores

em JavaScript **=** (*igual*) é usado para atribuição, logo toda vez que você quiser atribuir um valor a uma variável, vai usar = para comandar.

quando você usar **==** (*dois iguais*) em JavaScript, você está efetuando uma operação de “comparação” de valores, ou seja se algo do lado esquerdo é igual a algo do lado direito. esta operação SEMPRE retorna um valor booleano, true ou false.

quando você usar **===** (*três iguais*) em JavaScript, você está efetuando uma operação de comparação de valores e também de tipo. o valor do lado esquerdo será comparado com o do lado direito e posteriormente se os tipos também são iguais.

coerção

como o JavaScript lida com comparações de tipos diferentes

coerção

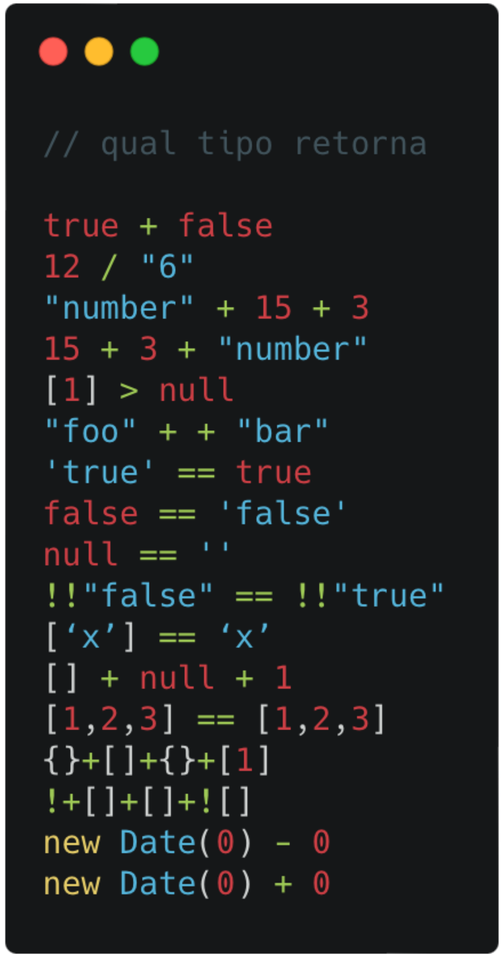
implícita e explicita

em Javascript a coerção pode acontecer de forma explícita, ou seja, você vai determinar que algo deve ser analisado como um determinado tipo, ou deixar que o JavaScript determina o resultado de forma implícita.

coerção de tipos (*type coercion*) é o processo de conversão de um valor de um tipo, para outro como a conversão de uma string para um número por exemplo.

sempre que possível evite usar coersão implícita pois é quase impossível lembrarmos de todas as combinações possíveis deste comportamento, como pode ser lido no link https://dorey.github.io/JavaScript-Equality-Table/

coerção
implícita e explicita



```
// qual tipo retorna

true + false
12 / "6"
"number" + 15 + 3
15 + 3 + "number"
[1] > null
"foo" + + "bar"
'true' == true
false == 'false'
null == ''
!!"false" == !!"true"
[‘x’] == ‘x’
[] + null + 1
[1,2,3] == [1,2,3]
{}+[]+{}+[1]
!+[]+[]+![]
new Date(0) - 0
new Date(0) + 0
```

alguns exemplos que podemos escrever em código JavaScript que vai retornar algo de um tipo. por coerção implícita, qual é o tipo que retorna cada uma das operações?

provavelmente algumas delas você jamais imaginaria encontrar ou que pudessem ser escritas, correto?

coerção
implícita e explicita

```
// qual tipo retorna?

console.log('true + false retorna um tipo:', typeof (true + false), 'e resultado:', true + false);
// "true + false retorna um tipo:", "number", "e resultado:", 1
console.log('12 / "6" retorna um tipo:', typeof (12 / "6"), 'e resultado:', 12 / "6");
 // "12 / "6" retorna um tipo:", "number", "e resultado:", 2
console.log('"number" + 15 + 3 retorna um tipo:', typeof ("number" + 15 + 3), 'e resultado:', "number" + 15
+ 3);
// ""number" + 15 + 3 retorna um tipo:", "string", "e resultado:", "number153"
console.log('"true" == true retorna um tipo:', typeof ("true" == true), 'e resultado:', "true" == true);
// ""true" == true retorna um tipo:", "boolean", "e resultado:", false
console.log('"foo" + + "bar" retorna um tipo:', typeof ("foo" + + "bar"), 'e resultado:', "foo" + + "bar");
// ""foo" + + "bar" retorna um tipo:", "string", "e resultado:", "fooNaN"
```


VAMOS PRATICAR
programming

revendo a prática

princípio básico

1. vimos que JavaScript possui tipos ***primitivos*** comuns de quaisquer outras linguagens.
2. que diferente de outras linguagens, JavaScript trata o tipo “**number**" para ***inteiros*** e ***floats*** da mesma forma.
3. que é possível testar o tipo da minha variável quando for necessário usando operador ***typeof***.
4. que javascript faz uso da ***coerção*** quando necessário.