Javascript

Características

- Linguagem interpretada
- Sintaxe utiliza chaves para controle de fluxo ({ }), parecida com Java/C#
- Multi-paradigma - Pode utilizar programação orientada a objetos ou funcional
- Tipagem dinâmica, fracamente tipada
- Funções são tratadas como membros de primeira-classe
- Suporte a programação orientada a objetos é implementada via prototypes ao invés de classes

Tipos de declaração de variável

- let
  - Método padrão para declaração de variável, só pode ser acessada dentro do bloco em que foi declarada.
- const
  - Precisa ser inicializada com um valor imediatamente, não se pode atribuir outro valor posteriormente. Também só pode ser acessada dentro do bloco em que foi declarada.
- var
  - Antigamente era o único método disponível para declaração, mas esse método não respeita o bloco em que foi declarada a variável e sim o escopo (global ou da função) pois ela tem o mesmo comportamento do *hoisting* de funções, tendo sua declaração movida para o começo do escopo, podendo gerar confusão. Evitar, utilizar *let* no lugar.

Operadores

- Númericos: +, -, /, *, %, +=, -=, *=, /=, ++, --
- Concatenação de strings: +
- Comparação: <, >, <=, >=, ==, ===
- Lógicos: &&, ||
- Ternário: ? (ex: x ? true : false)

Primitivas

Primitivas são dados básicos que não são objetos e não tem métodos.

São imutáveis, não podem ser alterados depois de criados.

- string
- number
- bigint
- boolean
- symbol
- undefined
- null

Wrappers Primitivos

São as classes que envelopam os valores primitivos para dar acesso aos métodos de cada tipo:

- string => String
- number => Number
- bigint => BigInt
- boolean => Boolean
- symbol => Symbol

Arrays

- Estrutura de dados padrão do javascript para representar vetores/arranjos
- Possui vários métodos auxiliares para realizar iterações no vetor.
- Métodos principais:
  - map = transforma os dados do vetor a partir de uma função de transformação
  - filter = filtra os dados a partir de uma função de predicado
  - Find = encontra um dado a partir de uma função de predicado
  - reduce = iterador genérico, pode implementar qualquer tipo de iteração
  - every = checa se todos os items passam na função de predicado
  - forEach = executa uma função para cada item

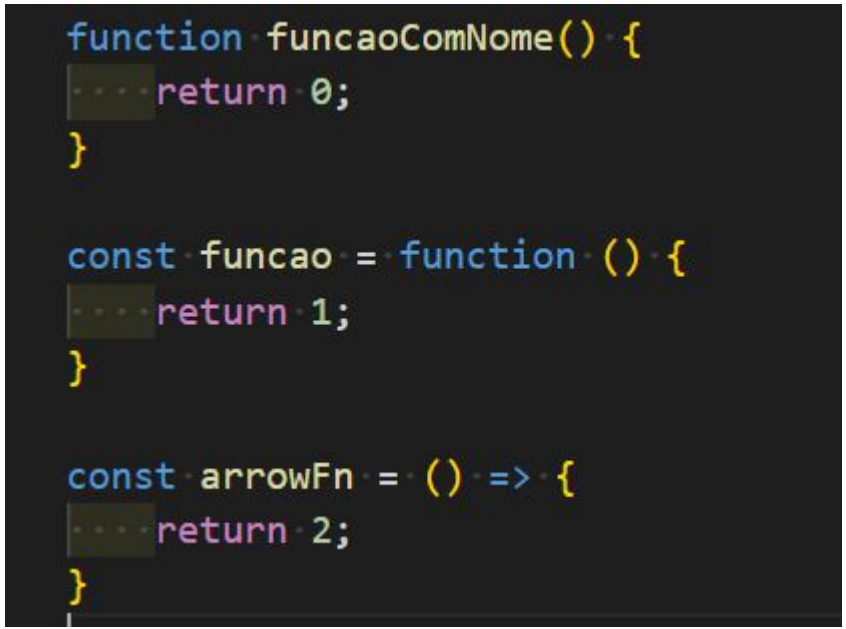
```
const arrayNumeros = [1,2,3,4,5];
```

Loops de repetição

- for
- while
- do...while
- break
- continue
- labels de controle de loop
- for...in
- for...of

Funções

- São tratadas como membros de primeira-classe, podendo ser:
  - Atribuidas a uma variável
  - Recebidas como argumentos de outra função
  - Podem retornar outra função
  - Devido a tipagem dinâmica, não há suporte a funcionalidade de *overload* nativamente, mas pode ser implementado com código customizado.
- Vários jeitos de declarar
  - Função nomeada
  - Função anônima atribuída a uma variável/constante
  - “Arrow-function”, sintaxe abreviada que também tem comportamento diferenciado
- Gera um escopo próprio
  - Funções filhas podem acessar variáveis do escopo da função pai (*Closure*)



Classes

São sintax-sugar para se trabalhar mais facilmente no Javascript com o paradigma de Orientação a Objetos

- Por ser uma linguagem de tipagem fraca, não possui tanto poder quanto outras linguagens fortemente tipadas como Java ou C#
- Classes são implementadas via cadeia de prototypes, que são objetos base que definem os métodos do objeto em si
- A utilização de classes evita problemas gerais que o javascript possui com o *this*

Promises

- Estrutura de dados padrão para lidar com operações assíncronas
- Consegue ser encadeada com outras promises
- Consegue “borbulhar” os erros para a promise acima

```
const minhaPromise: Promise<string>
const minhaPromise = new Promise<string>((resolve, reject) => {
    setTimeout(() => resolve('resultado'), 1000);
})

minhaPromise.then(resultado => {
    console.log(resultado) // vai printar 'resultado' após 1 segundo
})
```

Modulos

Serve para compartilhar código entre arquivos javascript.

Há duas declarações que usamos para trabalharmos com módulos:

- export
  - Pode exportar variáveis, funções, tipos (no caso de typescript)
  - Cada arquivo pode somente um export *default*
- import
  - Precisa ser usada no nível mais alto do arquivo (não pode ser usada dentro de funções, por exemplo)
  - Imports precisam referenciar cada um dos exports por nome
  - Pode-se importar todo um modulo como uma namespace usando a sintaxe de “import * as NAMESPACE from “./caminho”
  - Pode-se importar um export default dando-se um nome ao export default

Destrutores

Permite declarar variáveis a partir da estrutura de um objeto.

- Ajuda a dar foco às propriedades do objeto que são relevantes
- Evita repetir o acesso constante ao objeto original para navegar nas propriedades
- Consegue destruturar arrays em índices e resto
- Pode-se definir valores padrão caso a propriedade não consiga ser desconstruida

Shorthands

- Sintaxes abreviadas para casos comuns de uso
- As vezes devem ser evitadas pois podem diminuir a clareza do código
- Shorthands recomendados:
  - Retorno de arrow-function
  - Arrow-function com um argumento
  - Inicializadores de objetos literais
- Shorthands não recomendados
  - Ifs/elses sem bloco { }
  - Omissão do uso do ponto e vírgula

Exercicios Javascript

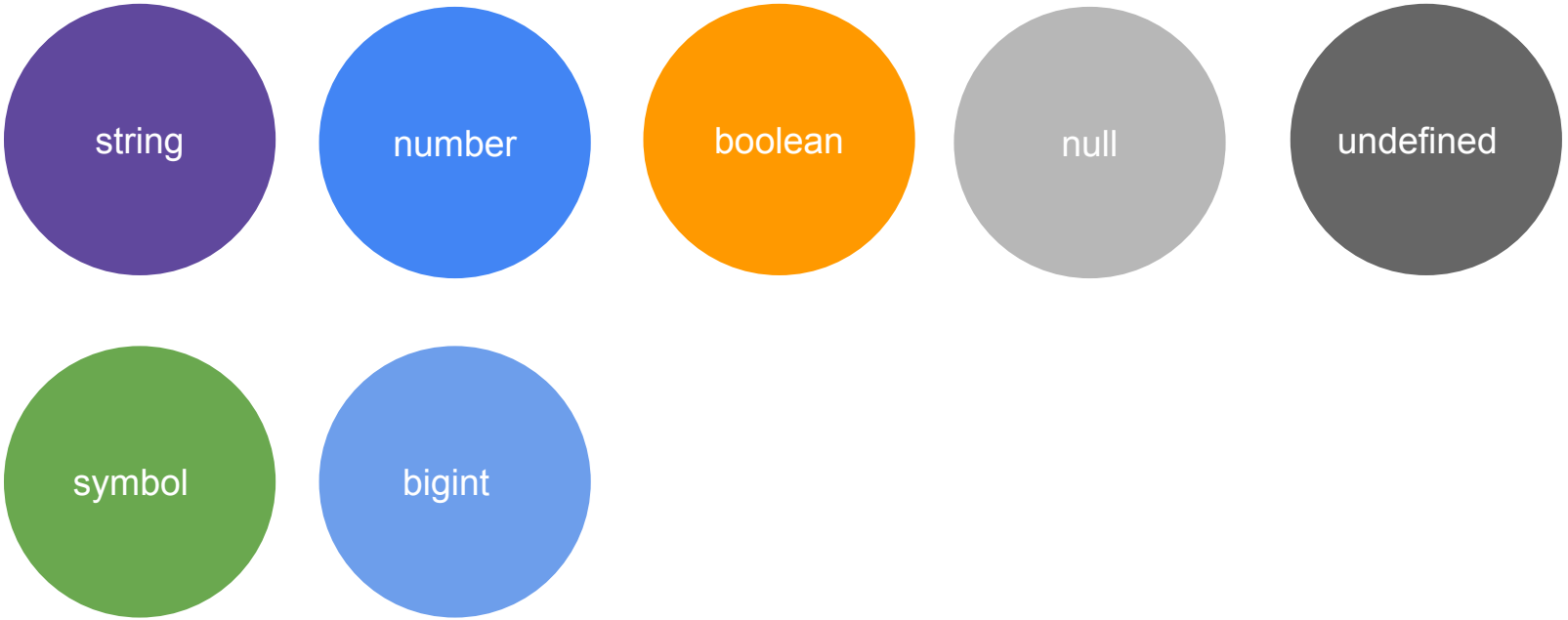
Typescript

Características

- Superset de Javascript - adiciona um sistema estático de tipagem estrutural
- Linguagem compilada para javascript
- Altamente configurável, com vários níveis de rigidez
- Compilador, que consegue compilar tanto Typescript quando Javascript moderno para versões mais antigas de Javascript.
- Servidor de linguagem que serve para prover funcionalidades de IDE para diversos editores de texto (ex: imports automáticos, refatorações, etc)
- Open-source e feito na própria linguagem de Typescript.

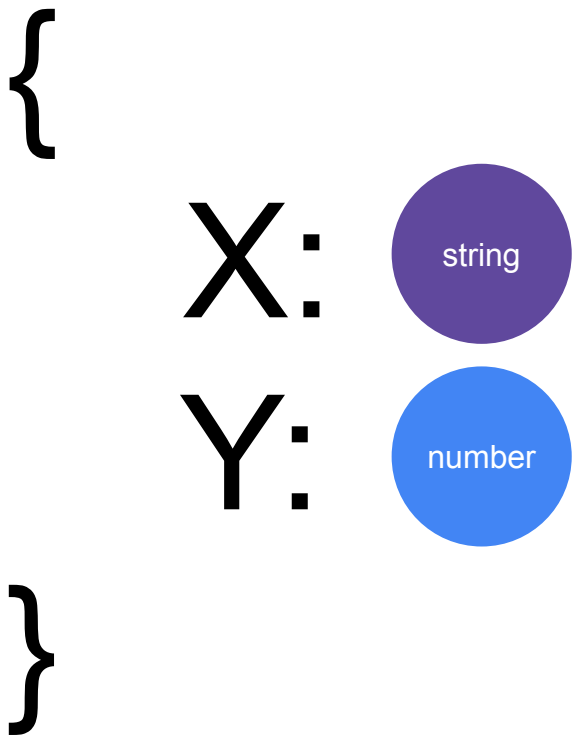
Sistema de tipos

Tipos primitivos - iguais os de Javascript

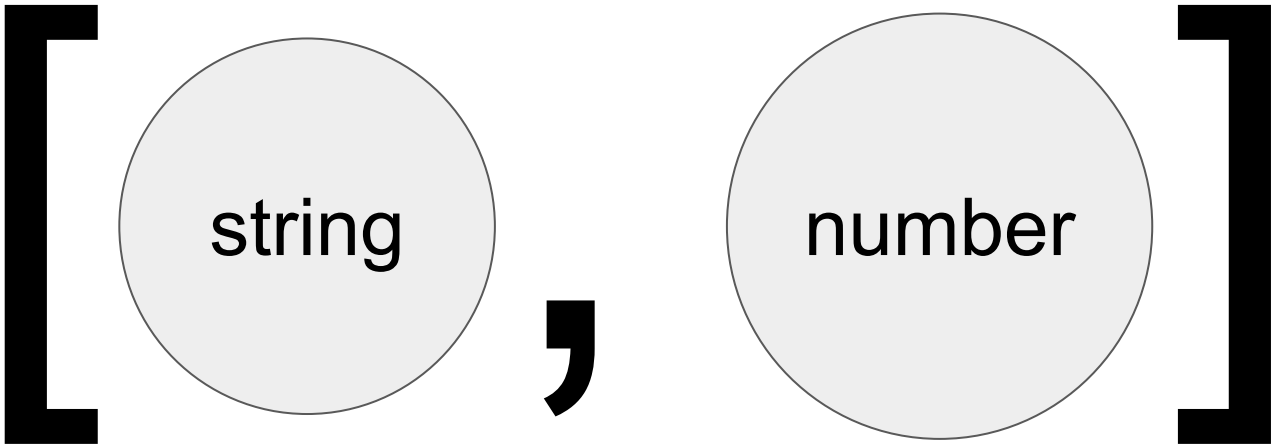


Arrays

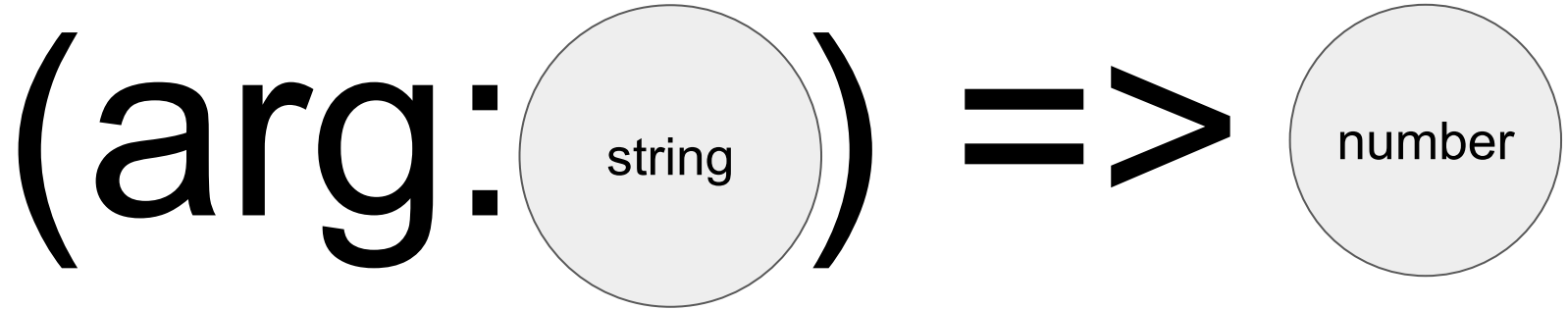
Objetos



Tuplas

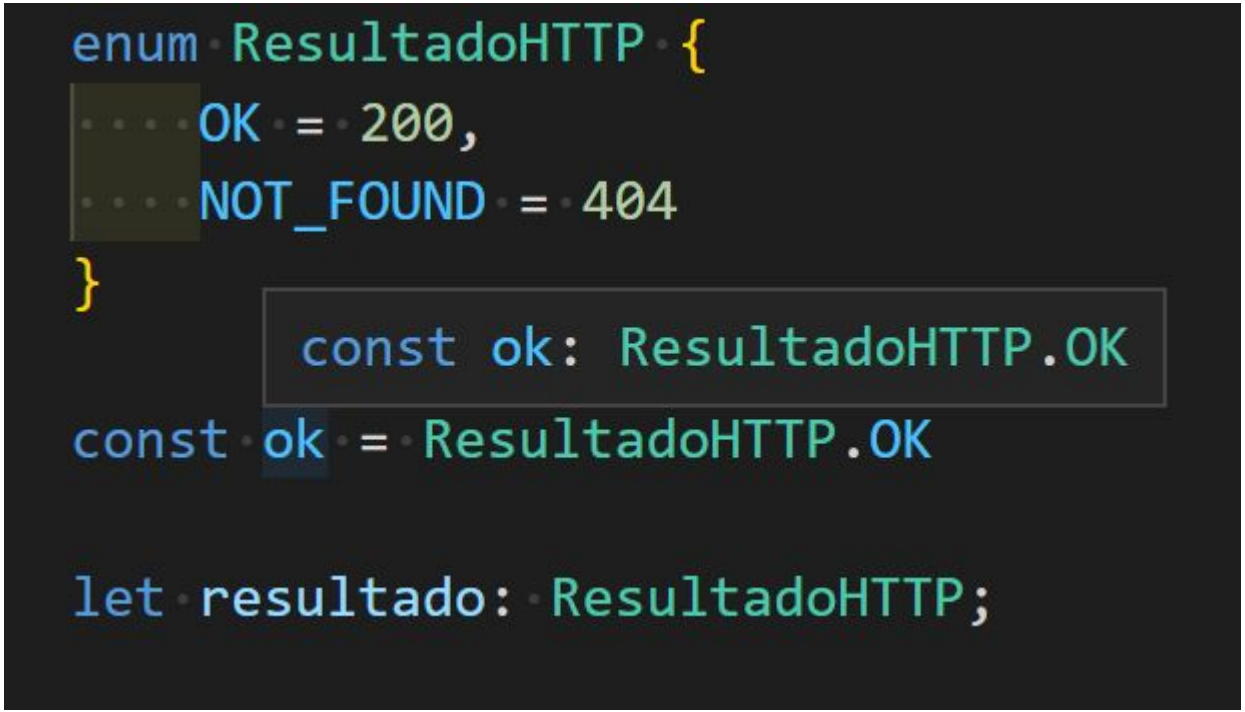


Funções



Enums

Representa tanto um valor e um tipo com chaves e valores padrões.



Interface e Type Alias

Servem para criar tipos customizados.

- Interfaces devem ser usadas para criação de tipos objetos estruturados
- Interfaces podem ser extendidas por outros módulos
- Interfaces tem melhor performance em relação aos types devido a simplicidade
- Types podem ser utilizados para criar tipos de objetos estruturados como Interfaces, mas são muito mais poderosos
- Types podem representar tipos literais primitivos
- Types podem modificar outros tipos para gerar um novo tipo

Generics

Tipos que aceitam outros tipos como parâmetro

- Podem restringir o parâmetro de tipo em um subconjunto
- Podem ter um valor default para o parâmetro de tipo
- É a base para que se faça a manipulação e transformações de tipos dinamicamente via Typescript

Exercicios Typescript