

Aula 1 – Javascript para Backend

PROFESSOR: HARLEY MACÊDO DE MELLO

PROJETO DE SISTEMAS WEB 2

Códigos: github.com/harleymacedo/AulaBackendDeveloper

Materiais de apoio: w3schools.com/js

Contato: harley.mello@ifce.edu.br

Roteiro

- Conceitos
- Ambiente de desenvolvimento
- Tipos de dados
- Escopo de variável
- Estrutura de seleção
- Array
- Funções

Roteiro

- Funções para string
- Funções matemáticas
- Funções para números
- Funções para data e hora
- Funções de callback e assincronismo
- Promise e Async Await
- Classes e objetos
- Módulos e importações

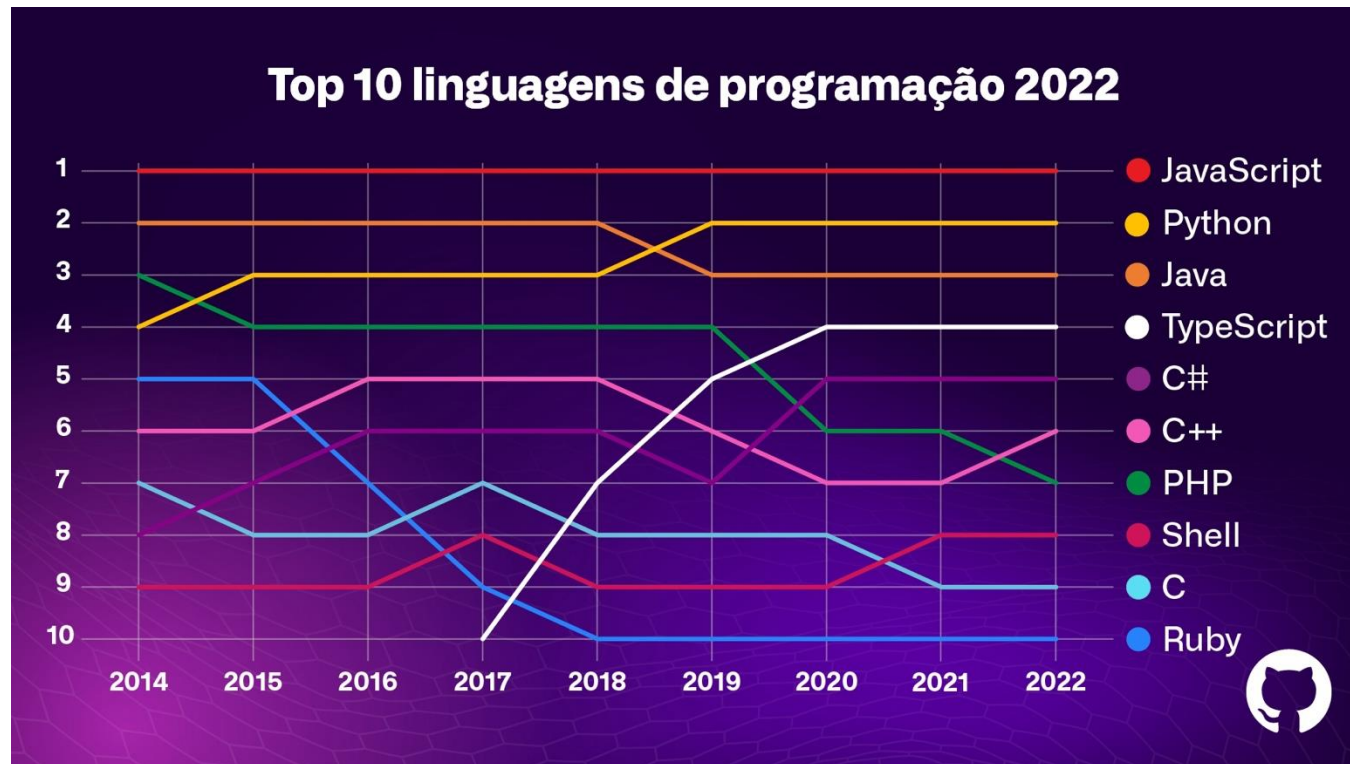
Conceitos

- As plataformas web e mobile são as principais escolhas para os softwares atualmente
- A linguagem Javascript se destaca para o desenvolvimento nas plataformas web e mobile
- Muitos produtos foram e são desenvolvidos baseados na linguagem Javascript
- Muitas vagas de trabalho exigem Javascript e alguns frameworks relacionados
- Conhecer a linguagem à fundo é o primeiro passo para a programação web e mobile
- Só então deve-se estudar os frameworks e bibliotecas relacionados

Conceitos

- A linguagem Javascript surgiu em 1995, criada pela Netscape
- Tinha a finalidade de dar dinamismo aos sites
- No início, apenas os navegadores de internet interpretavam a linguagem
- Em 2009 houve a iniciativa de usá-la como linguagem de propósito geral
- A plataforma NodeJS foi desenvolvido para essa finalidade

Conceitos

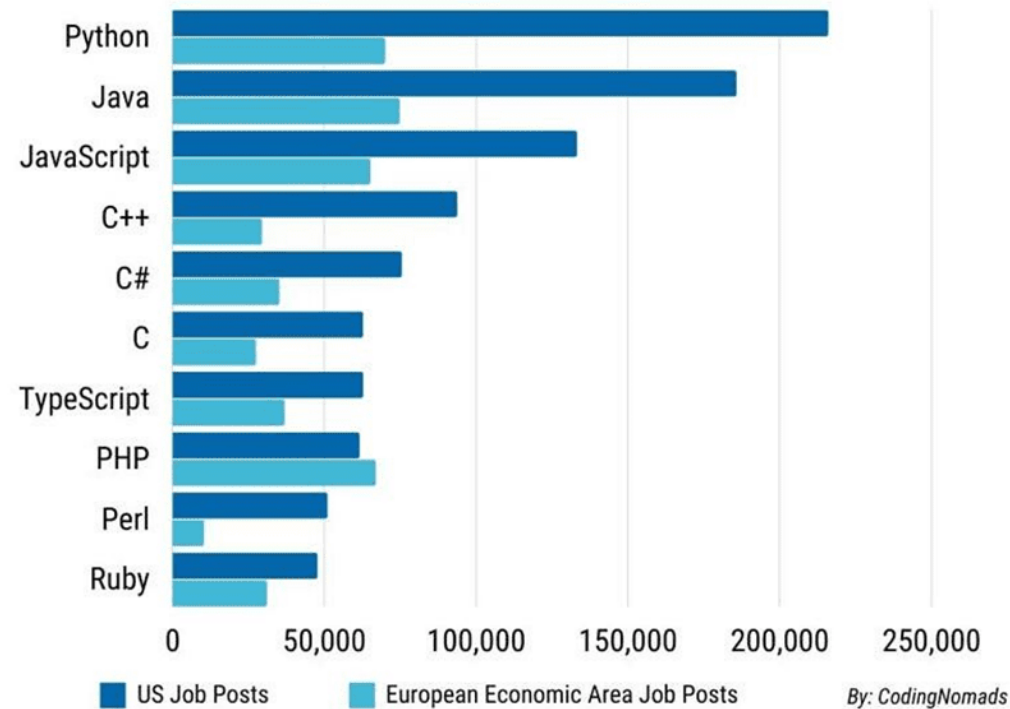


Linguagens mais usadas em 2022, segundo o Github.

Conceitos

Most in-demand programming languages of 2022

Based on LinkedIn job postings in the USA & Europe



Linguagens mais usadas conforme a demanda, em 2022, segundo Statista.

Ambiente de desenvolvimento

- VS Code
 - Instalar no NodeJS (nodejs.org)
 - Usar o console do VS Code
- Replit
 - Editor de código online
 - Projeto salvo em nuvem
- Console do navegador
 - Abrir console de algum navegador
 - Limitação na criação de alguns recursos

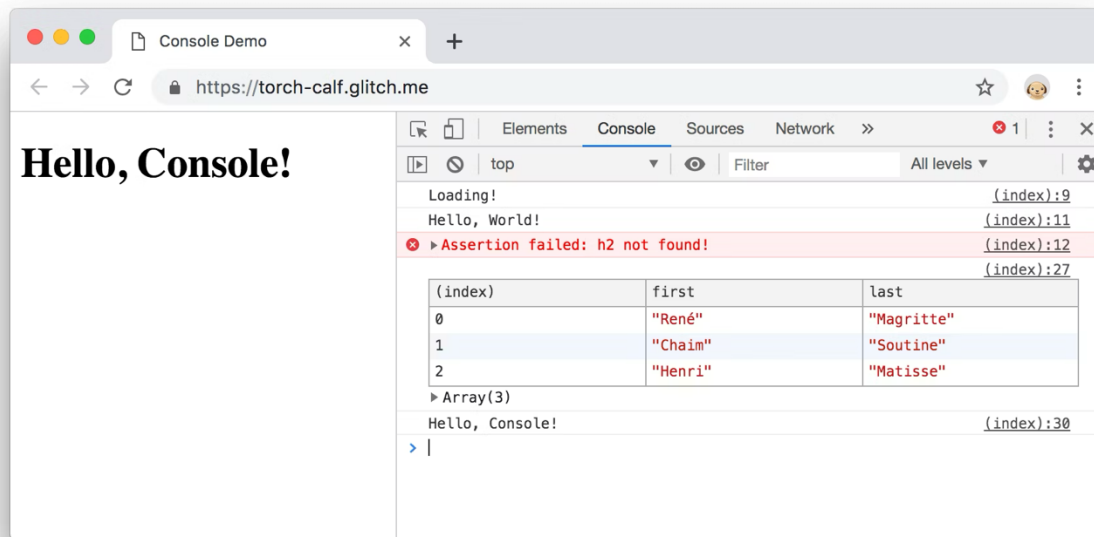
Ambiente de desenvolvimento



Ferramentas VS Code e Replit,
disponíveis nos respectivos sites:
code.visualstudio.com e replit.com.



Ambiente de desenvolvimento



Uso do console no navegador para execução de código Javascript. No Google Chrome basta acessar o menu Ver > Desenvolvedor > Inspecionar elementos.

Tipos de dados

- String
 - Conjunto de caracteres
 - Delimitados por aspas simples ou duplas
- Number
 - Qualquer valor numérico
 - Com ou sem casas decimais

Tipos de dados

- Boolean
 - Representam os valores lógicos true ou false
 - Usados em testes condicionais
- Array
 - Armazenam múltiplos valores sob um único nome
 - Criados com colchetes
 - Possuem índices, iniciando do zero

Tipos de dados

- Null
 - Valor inexistente
 - Tipo é null
- Undefined
 - Valor inexistente
 - Tipo é undefined

Tipos de dados

- Objeto
 - Conjunto de dados e comportamentos
 - Formados por nomes e valores
 - Delimitados por chaves
 - Podem ser acessados usando ponto

Tipos de dados

```
1 //Tipo string
2 var cidade = 'Crato'
3 console.log(typeof cidade)
4
5 //Tipo number
6 var km = 12
7 console.log(typeof km)
8
9 //Tipo boolean
10 var ligado = true
11 console.log(typeof ligado)
12
13 //Tipo objeto literal
14 var conta = {agencia: '1234-1', numero: '1234567-1'}
15 console.log(typeof conta)
16
17 //Tipo objeto array
18 var idades = [20, 45, 60]
19 console.log(typeof idades)
20
21 //Tipo undefined
22 var email = undefined
23 console.log(typeof email)
```

Tipos de dados do Javascript e a visualização dos mesmos usando o comando `typeof`.

Escopo de variável

- Var
 - Escopo global
- Let
 - Escopo global, função ou bloco
- Const
 - Escopo global, função ou bloco
 - Não pode ser alterado

Escopo de variável

```
1  //var e escopo de função
2  {
3  |   var x = 2
4  | }
5  console.log(x)
6
7  //let e escopo de bloco
8  {
9  |   let y = 2
10 | }
11 console.log(y)
12
13 //const e escopo de bloco
14 {
15 |   const z = 5
16 | }
17 console.log(z)
18
19 //const com mudança de valores do objeto
20 const notebook = {marca: 'Apple', modelo: 'Macbook Air'}
21 notebook.modelo = 'Macbook Air M1'
```

Escopo de variáveis, acesso e mudança das mesmas.

Estruturas de Seleção

- If, else, else if
 - O 'if' executa um código se uma condição for verdade
 - O 'else if' executa um código se uma condição for verdade, e a anterior for falsa
 - O 'else' executa um código se as condições anteriores não forem verdade
- Switch
 - O 'case' testa um caso para verificar se o mesmo é verdadeiro
 - O 'default' executa um bloco quando nenhum dos casos anteriores é verdadeiro

Estruturas de Seleção

```
1  //uso do if
2  var i = 10
3  if (i === 10) {
4      console.log('É 10')
5  } else if (i === 5) {
6      console.log('É 5')
7  } else {
8      console.log('Nenhuma das opções')
9  }
10
11 //uso do switch
12 var j = 15
13 switch (j) {
14     case 5:
15         console.log('É 5')
16         break
17     case 10:
18         console.log('É 10')
19         break
20     default:
21         console.log('Nenhuma das opções')
22 }
```

Uso das estruturas de seleção IF e Switch.

Estruturas de Repetição

- For
 - Executa bloco um número definido de vezes
 - Uso de variável de controle
 - Uso de incremento
 - O 'for in' itera pelas propriedades de um objeto
 - O 'forEach' itera pelos itens de um array

Estruturas de Repetição

```
1 //Estrutura for
2 for (let i = 1; i <= 10; i++) {
3     console.log(i)
4 }
5
6 //Estrutura for in
7 let usuario = {
8     nome: 'Ana',
9     email: 'ana@servidor.com',
10    cidade: 'Crato'
11 }
12 for (const atributo in usuario) {
13     console.log(atributo)
14 }
15
16 //Estrutura forEach
17 let opcoes = ['a', 'b', 'c']
18 opcoes.forEach(elemento => {
19     console.log(elemento)
20 })
```

Uso das estruturas de repetição for, for in, e forEach.

Estruturas de Repetição

- While
 - Executa bloco um número indefinido de vezes
 - Testa condição antes de executar
 - Precisa haver condição de parada atingível
 - Usar 'do while' para testar condição depois

Estruturas de Repetição

```
22  //Estrutura while
23  let j = 1
24  let resultado = 1
25  while (j <= 5) {
26      resultado = resultado * j
27      j++
28  }
29  console.log(resultado)
30
31  //Estrutura do while
32  let z = 1
33  do {
34      console.log(z)
35  } while (z < 1)
```

Uso das estruturas de repetição while e do while.

Array

- Armazena múltiplos valores em uma variável única
- Facilita a navegação por seus itens
- Utiliza números como índices automaticamente
- Facilita a busca por itens
- Atributo 'length' armazena o tamanho do Array
- Muitas funções utilitárias embutidas

Array

```
1 //Atributo length
2 var estados = ['Ceará', 'São Paulo', 'Minas Gerais']
3 console.log(estados.length);
4
5 //Uso do índice
6 console.log(estados[0])
```

Declarando um Array, usando o length e índice.

Array

- Podem armazenar dados mistos
- Podem armazenar objetos
- Função 'push' adiciona novo elemento
- Função 'Array.isArray(x)' testa se x é um array

Array

```
19 //Array armazenando vários tipos de dados
20 const variados = []
21 variados[0] = 'Apple'
22 variados[1] = Date.now()
23 variados[2] = function funcao1 () {}
24 variados.push(10)
25 console.log(variados)
26
27 //Teste de array
28 console.log( Array.isArray(variados) )
29 console.log(typeof variados)
```

Array com vários tipo de dados, uso do push e uso do isArray.

Array

- Principais funções utilitárias
 - 'concat()' junta 2 ou mais arrays
 - 'toString()' converte array para String
 - 'pop()' remove o último elemento
 - 'shift()' remove o primeiro elemento

Array

- Principais funções utilitárias
 - 'unshift()' insere no início
 - 'slice()' obtém um subconjunto do array
 - 'reverse()' inverte os elementos
 - 'sort()' ordena os elementos

Array

- 'map()'
 - Aplica uma função usando os valores do array
 - Não altera o array original
 - Retorna um novo array
- 'reduce()'
 - Reduz o array para um valor único
 - Executa uma função fornecida, reaplicando cada valor

Array

```
1 //Uso da função map
2 var studios = ['Warner Bros', '21 Century', 'Sony']
3 studios.map( function (s) { console.log(s.length) } )
4
5 //Uso da função reduce
6 var numeros = [8, 9, 4, 5, 9, 9]
7 var soma = numeros.reduce(function (total, numero) {
8   |   return total + numero
9   } )
10 console.log(soma)
```

Uso das funções map e reduce.

Funções

- Conjunto de comandos ou expressões
- Tratadas como objeto
- Podem ser declaradas, atribuídas e passadas como parâmetro de outra função
- Podem ser anônimas
- Arrow function permite uma sintaxe mais enxuta

Funções

- Função estática
 - Não precisa instanciar o objeto que contém a função
 - Mais prático
 - Math é um exemplo
 - Usar a palavra 'static' para criar esse tipo de função

Funções

```
1  //Formas de declarar função
2  function adicionar (num1, num2) {
3      |   return num1 + num2
4  }
5
6  const adicionar2 = function (num1, num2) {
7      |   return num1 + num2
8  }
9
10 const adicionar3 = (num1, num2) => {
11     |   return num1 + num2
12 }
13
14 console.log(adicionar(1, 2))
15 console.log(adicionar2(1, 2))
16 console.log(adicionar3(1, 2))
```

Formas de declarar uma função.

Funções para string

- `'slice()'` extrai uma parte de uma string
- `'replace()'` substitui uma string por outra
- `'toUpperCase()'` retorna string maiúscula
- `'toLowerCase()'` retorna string minúscula

Funções para string

```
1 //Função slice
2 let texto = 'O país do futebol'
3 let pedaco = texto.slice(2, 6)
4 console.log(pedaco)
5
6 //Função replace
7 let texto2 = 'O IFCE tem o curso de Sistemas de Informação'
8 let novoTexto2 = texto2.replace('IFCE', 'IFCE Campus Crato')
9 console.log(novoTexto2)
10
11 //Funções toLowerCase e toUpperCase
12 let texto3 = 'Lab Info 03'
13 let novoTexto3A = texto3.toLowerCase()
14 let novoTexto3B = texto3.toLocaleUpperCase()
15 console.log(novoTexto3A, novoTexto3B)
```

Uso das funções slice, replace, toLowerCase e toUpperCase.

Funções para string

- `'trim()'` remove espaços em branco no início e no final
- `'charAt()'` retorna um caractere na posição passada
- `'split()'` retorna um array com partes da string divididas com um breakpoint escolhido
- `'indexOf()'` retorna a posição do texto passado

Funções para string

```
17 //Função trim
18 let texto4 = ' Turma com 25 alunos '
19 let texto4Novo = texto4.trim()
20 console.log(texto4Novo)
21
22 //Função charAt
23 let texto5 = 'Brasil'
24 let letra = texto5.charAt(0)
25 console.log(letra)
26
27 //Função split
28 var texto6 = 'O curso de Sistemas de Informação do IFCE Campus Crato'
29 var palavras1 = texto6.split(' ')
30 console.log(palavras1)
31
32 //Função indexOf
33 var texto7 = 'O IFCE fica com Bairro Gisélia Pinheiro'
34 var encontrado = texto7.indexOf('IFCE')
35 console.log('Encontrado: ', encontrado)
```

Uso das funções trim, charAt, split e indexOf.

Funções matemáticas

- `'pow()'` elevar número à uma potência
- `'sqrt()'` raiz quadrada de um número
- `'ceil()'` arredonda para cima
- `'floor()'` arredonda para baixo
- `'random'` número aleatório entre 0 e 1

Funções matemáticas

```
1 //Função pow
2 let resultado1 = Math.pow(20, 2)
3 console.log(resultado1)
4
5 //Função ceil
6 let resultado2 = Math.ceil(4.48)
7 console.log(resultado2)
8
9 //Função floor
10 let resultado3 = Math.floor(4.48)
11 console.log(resultado3)
```

Uso das funções pow, ceil e floor.

Funções matemáticas

```
13 //Função sqrt
14 let resultado4 = Math.sqrt(81)
15 console.log(resultado4)
16
17 //Função random, * máximo + mínimo
18 let aleatorio1 = Math.floor(Math.random() * 10) + 1)
19 console.log(aleatorio1)
20
21 //Função random com array
22 let times = ['Time1', 'Time2', 'Time3', 'Time4', 'Time5', 'Time6']
23 let aleatorio2 = Math.floor(Math.random() * 5 )
24 console.log(aleatorio2, times[aleatorio2])
```

Uso das funções sqrt e random.

Funções para números

- `'toString()'` converte número em string
- `'toFixed()'` define quantidade de casas decimais
- `'parseFloat()'` converte valor para fracionário
- `'parseInt()'` converte valor para inteiro

Funções para números

```
1  //Converter número para string
2  let num1 = 5
3  let texto1 = num1.toString()
4  console.log(typeof texto1, texto1)
5
6  //Define quantidade de casas decimais
7  let num2 = 2.8244
8  console.log(num2.toFixed(2))
9
10 //Converte string para número
11 let texto2 = '1010'
12 let num3 = parseInt(texto2)
13 console.log(typeof num3, num3)
```

Uso das funções toString, toFixed e parseInt.

Funções para data e hora

- `'new Date()'` cria um objeto do tipo data para trabalhar com data e hora
- `'getFullYear()'` obtém o ano com 4 dígitos
- `'getMonth()'` obtém o mês de 0 a 11
- `'getDate()'` obtém o dia do mês de 1 a 31
- `'getHours()'` obtém as horas de 0 a 23
- `'getMinutes()'` obtém os minutos de 0 a 59
- `'getSeconds()'` obtém os segundos de 0 a 59
- `'toLocaleString()'` formata data e hora para formato local

Funções para data e hora

```
1  //Instanciar data e imprimir ano
2  let dataHora = new Date()
3  console.log(dataHora.getFullYear())
4
5  //Imprimir dia e Mês
6  let diaMes = dataHora.getDate() + '/' + (dataHora.getMonth() + 1)
7  console.log(diaMes)
8
9  //Imprimir data e hora formatada com toLocaleString
10 let dataHora2 = new Date()
11 console.log( dataHora2.toLocaleString('pt-br') )
```

Trabalhando com
data e hora.

Exercícios

- Questão 1
 - Crie uma função que realiza um sorteio com 5 números entre 1 e 80
 - Os números sorteados não podem ser repetidos
 - Imprima os números sorteados

Exercícios

- Questão 2
 - Crie uma função que conte quantas vogais um determinado texto possui
 - A função recebe uma String como parâmetro
 - A função retorna a quantidade de vogais que tem a String

Exercícios

- Questão 3
 - Crie uma função que imprime uma barra de progresso baseado no número de segundos
 - Em um texto de sessenta posições, será impresso N '*', sendo N o número de segundos
 - Os demais caracteres, até completar 60, devem ser preenchidos com underline '_'
 - Adicione um pipeline '|', no início e no fim do texto
 - A função não retorna valores, apenas imprime a barra

Funções de callback e sincronismo

- Função que chama outra função, criando uma dependência
- Técnica para definir o sincronismo de funções
- Operações custosas executam em background
- Amarra uma sequência concisa no código

Funções de callback e sincronismo

```
1 //Funções não executarão em ordem
2 const {tarefa1, tarefa2, tarefa3} = require('./Auxiliar')
3 tarefa1()
4 tarefa2()
5 tarefa3()
6
7 //Funções executarão em ordem
8 const {tarefa1b, tarefa2b, tarefa3b} = require('./Auxilar2')
9 tarefa1b( () => {
10     tarefa2b( () => {
11         tarefa3b()
12     } )
13 } )
```

Execução de funções de forma síncrona e assíncrona.

Funções de callback e sincronismo

```
1  function tarefa1 () {
2    |    setTimeout( () => { console.log('Tarefa 1') }, 4000 )
3  }
4  function tarefa2 () {
5    |    setTimeout( () => { console.log('Tarefa 2') }, 2000 )
6  }
7  function tarefa3 () {
8    |    setTimeout( () => { console.log('Tarefa 3') }, 1000 )
9  }
10
11  module.exports = {tarefa1, tarefa2, tarefa3}
```

Definição de funções custosas.

Funções de callback e sincronismo

```
1  function tarefa1b (callback) {
2      |   setTimeout( () => {
3          |       console.log('Tarefa 1b')
4          |       if (callback) callback()
5          |   }, 9000 )
6  }
7  function tarefa2b (callback) {
8      |   setTimeout( () => {
9          |       console.log('Tarefa 2b')
10         |       if (callback) callback()
11         |   }, 8000 )
12  }
13 function tarefa3b () {
14     |   setTimeout( () => {
15         |       console.log('Tarefa 3b')
16         |   }, 7000 )
17  }
18
19 module.exports = {tarefa1b, tarefa2b, tarefa3b}
```

Definição de funções custosas, mas usando callback.

Classes e objetos

- Molde para criação de objetos
- Padroniza objetos
- Agrupa atributos e comportamentos
- Introduzido no ES 2015
- Usar a palavra 'class'
- Pode herdar de outra classe

Classes e objetos

- 'new' cria um objeto de uma determinada classe
- 'this' referencia o próprio objeto em uso
- Uma classe pode ter um construtor definido
- Métodos estáticos são chamados sem instanciar o objeto

Classes e objetos

```
24 //Uso de método estático
25 class Carro {
26     static alo () {
27         console.log('Sou um carro incompleto')
28     }
29 }
30 Carro.alo()
```

Uso de método
estático.

Classes e objetos

```
1  class DispositivoEletronico {
2      constructor(nome) {
3          this.nome = nome
4          this.ligado = false
5      }
6
7      ligar() {
8          this.ligado = true
9      }
10 }
11
12 class Smartphone extends DispositivoEletronico {
13
14 }
15
16 const d1 = new DispositivoEletronico('Smart TV')
17 console.log(d1)
18
19 const s1 = new Smartphone('Iphone 12')
20 s1.ligar()
21 console.log(s1)
```

Uso de classe e herança.

Módulos e exportações

- Permite separar o código em diversos arquivos
- Facilita manutenção e reuso de código
- Variáveis e funções podem ser exportadas
- Pode ser exportado uma ou mais variáveis

Módulos e exportações

```
1 //Reuso de função em outro arquivo
2 const robo = require('./Auxiliar')
3 robo.dizOi()
4 console.log(robo.nome)
5
6 //Obtendo mais de uma variável
7 const {nomes, cidades} = require('./Auxiliar2')
8 console.log(nomes, cidades)
```

Importando variáveis de outro arquivo com o require.

Módulos e exportações

```
1  const robo = {  
2    nome: 'Robo 1',  
3    dizOi: () => { console.log('Oi') }  
4  }  
5  
6  module.exports = robo
```

Exportando uma variável.

Módulos e exportações

```
1 let nomes = ['Beatriz', 'Marcos']
2
3 let cidades = ['Crato', 'Juazeiro do Norte']
4
5 module.exports = {nomes, cidades}
```

Exportando duas variáveis.

Módulos e exportações

- Outra forma de reusar módulos é com o padrão ES6
- Usando a sintaxe para exportar 'export default'
- Importando com import x from 'arquivo.js'
- É necessário habilitar o padrão no arquivo 'package.json'
- Definir com a linha 'type: module'

Exercícios

- Questão 4
 - Crie uma função classe chamada 'Utilitario' que possui
 - Uma função chamada 'geraNomeArquivo'
 - Essa função gera uma palavra com 6 letras, sendo 3 pares de consoante e vogal (Ex: bacato)
 - A palavra não precisa existir no dicionário de português
 - Em seguida a palavra é concatenada com um '-' e com um timestamp (tempo em milisegundos)
 - Depois a palavra é concatenada com '.js'
 - O nome completo é impresso pela função
 - Um exemplo de nome de arquivo seria 'tufaro-983749387493847.js'
 - Instancie um objeto 'Utilitario' e chame a função 'geraNomeArquivo'

Exercícios

- Questão 5
 - Crie uma função com `async/await` chamada 'lerDados' que realiza a leitura em um arquivo txt
 - Essa função usa o comando `readFile` da biblioteca `fs/promises`
 - Realize a impressão do conteúdo do arquivo