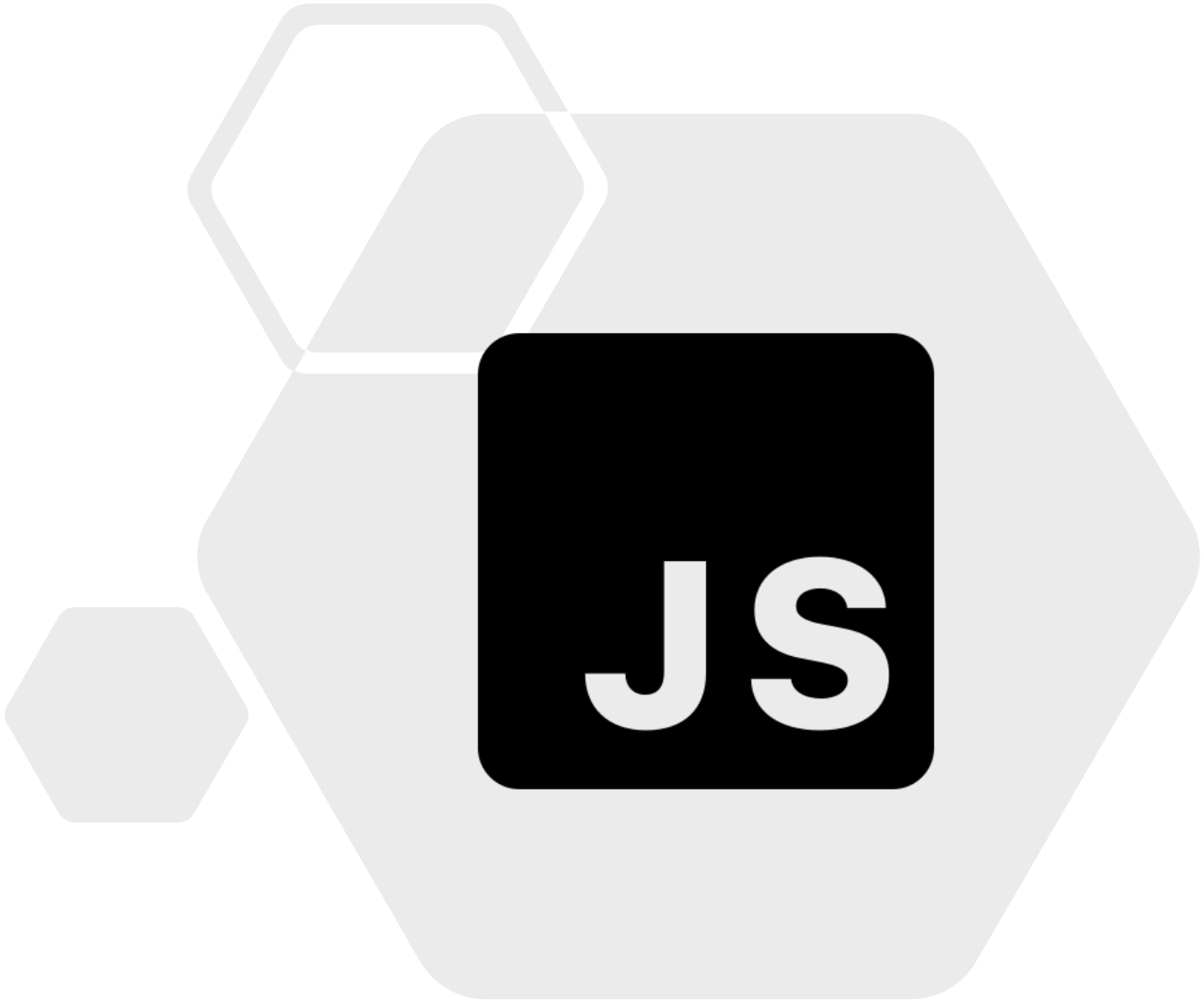
Módulos



Antes uma breve contextualização...

No seu início, a linguagem JavaScript era usada para fazer programas pequenos, scripts. A maior parte desses scripts serviam para dar interatividade à páginas html. Nesta época não era comum usar scripts grandes.

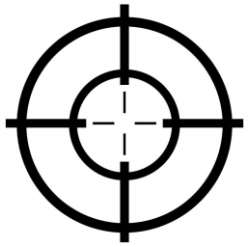
O avanço rápido da tecnologia possibilitou a criação de aplicativos completos, que são executados em navegadores. Esses aplicativos utilizam, e muito, JavaScript.

Por que módulos são importantes?

O avanço da tecnologia provocou uma grande mudança, tornou-se necessário fornecer mecanismos para dividir programas JavaScript em partes separadas, que podem ser importadas quando necessário, como bibliotecas. Estas partes são chamadas de módulos.

O node.js é um software amplamente utilizado para construção de aplicações com JavaScript. Ele é capaz de gerenciar módulos e auxiliar o desenvolvedor a criar, importar ou exportar módulos.

Quando surgiram os módulos?



Antes do ES6, o Javascript possuía sistemas de controles de módulos como o RequireJS e CommonJS.

Mas, a partir de 2015 foi que os módulos evoluíram e, desde então, são importantíssimos para o desenvolvimento de aplicações, principalmente com o node.js.

Antes de entender módulos...

Algo importante na utilização de módulos é o “strict mode”. Provavelmente as palavras “use strict” já foram vistas no topo de um script.

O strict mode do ES5 é uma forma de optar por uma variante restrita do JavaScript.

O modo estrito é ativado por padrão quando se trabalha com módulos ES6.

Mas, o que o modo estrito faz?

O strict mode faz várias mudanças nas semânticas normais do JavaScript.

Primeiro, elimina alguns erros silenciosos do JavaScript, fazendo-os lançar exceções. Segundo, evita equívocos que dificultam que motores JavaScript realizem otimizações: código strict mode pode às vezes ser feito para executar mais rápido que código idêntico não-strict mode.

Modo estrito

```
NaN = 1
```

NaN é uma palavra reservada que significa “Not a Number”. É usada para representar que algo está errado com a atribuição ou operação de um ou mais números. Portanto, não deveria ser permitido atribuir valor ao NaN.

O “use strict” garante que este erro não fique despercebido.

```
"use strict";
NaN = 1
```

```
TypeError: Cannot assign to read only property 'NaN' of object '#<Object>'
    at Object.<anonymous> (c:\Users\gerso\vscode-projetos\js-conceitos-basicos\js-conceitos-basicos.js:3:5)
    at Module._compile (node:internal/modules/cjs/loader:1101:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1153:10)
    at Module.load (node:internal/modules/cjs/loader:981:32)
    at Function.Module._load (node:internal/modules/cjs/loader:822:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:79:12)
    at node:internal/main/run_main_module:17:47
```

Node.js

O node.js foi projetado para construir aplicativos escalonáveis, como JavaScript.

O node.js pode ser definido como um ambiente de execução Javascript “server-side”, um servidor para aplicações JavaScript.

node.js é um software de código aberto, multiplataforma, baseado no interpretador V8 da Google.

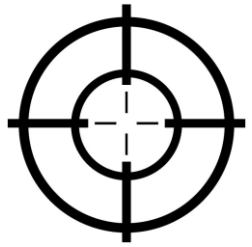
Para diversão...

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Oi, eu sou o node.js');
});

server.listen(port, hostname, () => {
    console.log('Server running at http://' + hostname + ':' + port + '/');
});
```

Inclusão de um módulo, usando a forma tradicional do JavaScript.

Iniciando o servidor node...

Aproveitando o embalo do node.js...

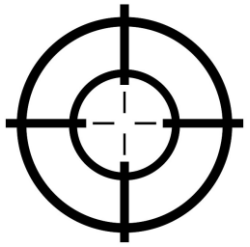


Existem algumas formas de concatenar (juntar) textos (strings) no JavaScript. Mas os desenvolvedores gostam de usar a interpolação de strings.

Em JavaScript, pode-se inserir ou interpolar variáveis em strings usando padrões (templates) literais.

Para isso, utiliza-se a notação ${}, sendo que dentro das chaves deve-se colocar a variável que deseja-se interpolar na string.

Interpolação de strings...

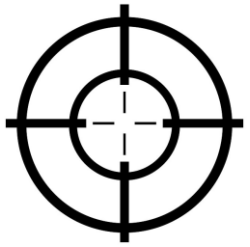


```
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Na interpolação de strings utiliza-se crase no lugar de aspas para identificar a string.

Basicamente é uma nova forma de criar ou juntar strings e tornar o seu código um pouco mais legível.

Antes de começar a criar módulos...

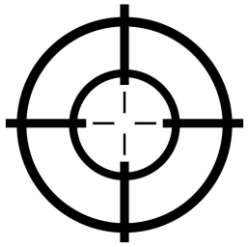


O node.js possui uma série de recursos, softwares que já vem com ele e facilitam seu uso. Principalmente a criação e instalação de módulos.

O npm (node package manager) é o gerenciador de pacotes (módulos) do node.js. Com ele tem-se acesso ao principal repositório de softwares, onde os desenvolvedores buscam módulos para colocar em suas aplicações, como bibliotecas.

Utiliza-se o npm como um comando cli (client line interface).

Mais detalhes do npm...

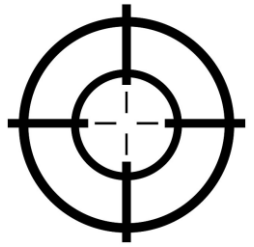


O npm é o gerenciador de pacotes padrão e mais popular no ecossistema node.js.

Também é comumente usado para instalar uma ampla variedade de ferramentas cli e executar scripts de projeto.

O npm rastreia os módulos instalados em um projeto com o arquivo package.json, que reside no diretório do projeto.

Criando um módulo...



O módulo deve ser iniciado (criado), a partir de um diretório vazio, onde pretende-se iniciar a codificação do módulo.

```
\vscode-projetos\modulo> npm init

This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (modulo)
```

Imediatamente será solicitado a inserção da descrição do módulo. Isto fica salvo no arquivo package.json.

Package.json

No arquivo package.json estão todos os módulos necessários para um projeto e suas versões instaladas, todos os metadados de um projeto, como o autor, a licença, etc e scripts, que podem ser executados para automatizar tarefas dentro do projeto.

```
{
  "name": "modulo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Este arquivo contém a descrição do módulo e é fundamental para seu gerenciamento e instalação em outros projetos.

Package.json

- Dica importante! Se o projeto for usar git para controle de versão, deve-se criar o repositório git primeiro e, em seguida, execute npm init. O comando entende automaticamente que está em uma pasta habilitada para git e preenche todas as informações necessárias para o controle de versão, automaticamente.

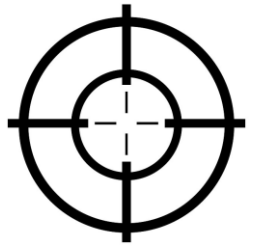
Indentificando as coisas...

```
{
  "name": "modulo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Nome do módulo, que será usada para instalação.

Arquivo que contém o código fonte de tudo que o módulo tem/faz. Este arquivo fica no mesmo diretório que o package.json.

Colocando coisas em um módulo...



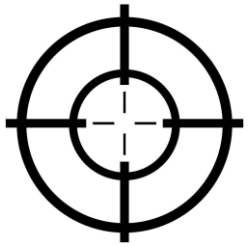
JS index.js > ...

```
1   boasVindas = (nome) =>{
2       return `Bem-vindo aos módulos, ${nome}`
3   }
4
5   opiniao = (oQueVoceAcha) =>{
6       return `Agora eu acho que usar módulos é ${oQueVoceAcha}`
7   }
8
9   exports.atributoDoModulo = boasVindas
10  exports.suaOpiniao = opiniao
```

Lembre-se, o código deve ser feito dentro do arquivo definido no package.json

No modelo CommonJS pode-se exportar valores atribuindo eles ao module.exports.

Instalando um módulo...

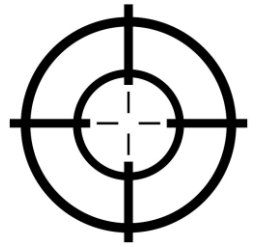


```
npm install ..\modulo\
```

Utiliza-se o npm para instalação de módulos. A partir disso, o módulo torna-se disponível para ser usado no projeto.

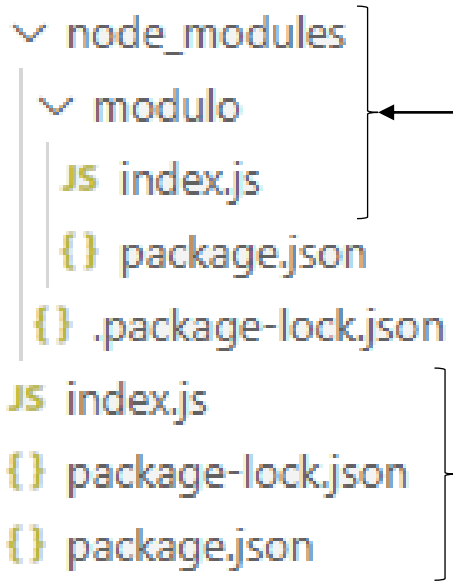
Para instalar um módulo é preciso informar seu nome e/ou endereço. Neste caso a instalação é de um módulo localizado localmente na máquina, mas poderia ser um endereço na nuvem, como no github por exemplo.

O que acontece após a instalação?



O npm atualiza o arquivo package.json com a dependência (módulo) instalada.

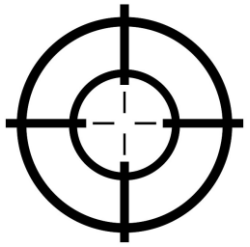
```
"dependencies": {
  "modulo": "file:../modulo"
}
```



É criado um diretório novo, chamado “node_modules”. Nele, geralmente, é colocado o código fonte do módulo instalado.

Também, é criado um novo arquivo de configuração, o package-lock.json.

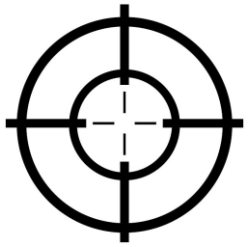
Para que serve o arquivo package-lock.json



Na versão 5, o npm introduziu o arquivo package-lock.json.

O objetivo do arquivo package-lock.json é manter o controle da versão exata de cada pacote instalado para que um produto seja 100% reproduzível da mesma maneira, mesmo que os pacotes sejam atualizados por seus mantenedores.

Usando um módulo...



Uma vez instalado, o módulo fica disponível e é possível acessar qualquer recurso dele. Para isso é preciso adicionar o módulo no topo do código que irá utilizá-lo.

```
const meuModulo = require('modulo')

console.log(meuModulo.atributoDoModulo('Gerson'))
console.log(meuModulo.atributoDoModulo('Muito legal!'))
```

Variável que recebe o módulo como um todo.

Utilizando as funções de dentro do módulo, através das variáveis exportadas.

Módulos do ES6

Uma das formas para adicionar o módulo, no código, é usar a palavra reservada “require”. Esta forma é mais antiga e foi definida no ES5, com o Common.JS.

Outra forma de adicionar o módulo no código é usar a palavra reservada “import”. Esta é uma das principais formas usadas.

O “import” foi definido no ES6. Para usá-lo é necessário incluir um atributo a mais no arquivo package.json.

Atributo type do package.json

A palavra reservada “import” só pode ser usada em módulos definidos pelo ES6.

Inclusão do atributo type.

```
{
  "name": "modulo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
```

O atributo type deve ser incluíndo no arquivo package.json, tanto do módulo que será exportado, quanto do que irá importar.

Export

A forma de exportar um módulo do ES6 é ligeiramente diferente.

```
let boasVindas = (nome) => {
    return `Bem-vindo aos módulos, ${nome}`
}

export default boasVindas
```

Importante! Lembre-se de declarar elementos com “const” ou “let”. Isto para evitar problemas devido ao conceito do “strict mode”.

Este tipo de export é chamado de padrão (default).

Import

A importação também ocorre de modo, ligeiramente, diferente para módulos ES6.

```
import boasVindas from 'modulo'

console.log(boasVindas('Novo programador'))
```

Nome ou caminho do módulo que será importado.

Neste exemplo, importou-se a função “boasVindas”, do módulo.

Exportando mais de um elemento...

Pode-se exportar mais de um elemento por vez. Para isso, usa-se a exportação explícita, que é diferente da padrão.

```
let boasVindas = (nome) => {
    return `Bem-vindo aos módulos, ${nome}`
}

let opiniao = (OqueVoceAchaDeModulos) => {
    return `Eu acho módulos ${OqueVoceAchaDeModulos}`
}

export { boasVindas, opiniao }
```

As duas funções estarão disponíveis após a importação.

Importando mais de um elemento...

Para a importação de mais de um elemento precisa-se usar {}. Os nomes dos elementos, que se deseja importar devem se colocados nas {}.

```
import {boasVindas} from 'modulo'

console.log(boasVindas('Novo programador'))
```

```
import { boasVindas, opiniao } from 'modulo'

console.log(boasVindas('Novo programador'))

console.log(opiniao('Muito top!'))
```

Importante! A principal vantagem é poder criar grandes bibliotecas dentro do mesmo módulo e importar somente o que for necessário.

JavaScript