# **O que é e como funciona o module.exports?**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/6/exercise/1#)

Durante o curso, estamos utilizando várias vezes a chamada module.exports para carregar novos módulos para a aplicação.

Essa é a maneira que o Node.js disponibiliza para fazermos tais carregamentos.

Mas de onde exatamente vem essa variável module?

O que ela faz para conseguir de fato carregar dinamicamente as funções dos módulos que passamos para ela?

O que garante que todo mundo que escreva um novo módulo para node esteja seguindo as convenções corretas para que este módulo possa ser carregado e funcione normalmente?

Quando aprendemos uma linguagem nova, é muito importante ter um bom conhecimento sobre seu funcionamento para poder utilizá-la da melhor maneira.

Pesquise sobre estes recursos e todos os outros que você achar necessário. Se quiser, dê uma olhada no próprio código do Node. É sempre legal poder fazer isso! =D

## **Resposta do instrutor**

Existe uma função interna do Node.js que recebe um path do módulo a ser carregado.

Essa função procura pelo local do módulo de acordo com o formato do path.

Para conseguir carregar a função do módulo, ele cria uma função com um nome do tipofuncaoDeCarregamento ou algo assim, que recebe uma função anônima e dentro dessa função, invoca a funçãoeval() do JavaScript passando como parâmetro o que foi carregado do módulo a partir do path recebido.

Pronto! Agora o código está carregado.

Para que ele fique visível de fora, é disponibilizado um objeto chamado module ou algo do tipo, que contém um objeto exports onde ficam armazenados todos os paths passados.

Por fim a funcaoDeCarregamento é invocada recebendo como parâmetro os próprios objetos module emodule.exports e é retornado o objeto module.

Essa é a maneira escolhida pelo Node.js para fazer carregamento dinâmico de módulos JavaScript. Outros frameworks podem fazer de formas diferentes.

Uma convenção que ajuda a padronizar essa estratégia é a CommonJS, que como já vimos, define várias Especificações para código JavaScript.

# **Porque utilizar os callbacks?**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/6/exercise/2#)

Durante o desenvolvimento da aplicação estamos sempre utilizando chamadas de funções que recebem como parâmetros outras funções. Esse funções que estamos enviando como parâmetros são chamadas de**callbacks** e servem para ser invocadas pelo próprio JavaScript assim que a função de fora terminar seu processamento.

Mas por que estamos utilizando essa estratégia constantemente em nossa aplicação com Node.JS?

O que ganhamos com isso?

Seria possível escrever o mesmo código sem utilizar os callbacks?

Quando uma função é invocada dentro da arquitetura Node.js o que acontece é que o fluxo do código não fica parado esperando o retorno dessa função.

Fala-se que as chamadas das funções são assíncronas ou não-blocantes, ou seja a chamada é feita e logo em seguida o processador já fica liberado para atender a novas chamadas.

Quem fica responsável por orquestrar esse escalonamento do processo é o próprio Sistema Operacional, que já é muito bom em fazer isso!

Por esse motivo é que o Node.js é muito útil para as aplicações que dependem de muitas operações de I/O (Input/Output - Entrada/Saída) em que existem várias chamadas a procedimentos que não necessariamente de lógica da aplicação, mas sim de consultas externas ou ações de infra-estrutura, como a leitura e escrita em um banco de dados, por exemplo.

Isso torna o Node.js muito performático e é uma das principais razões pelo seu sucesso!

Até seria possível escreve o código de uma maneira sequencial, mas como o Node.js simplesmente não espera o final do processamento de uma linha de código para executar a próxima, o funcionamento não seria o esperado.

# **Qual tipo de sistema tira mais proveito da infraestrutura provida pelo Node.js?**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/6/exercise/3#)

Neste capítulo vimos um pouco sobre o funcionamento do Node.js.

Pensando nisso, tente responder a pergunta feita com base nas características de execução do Node.js que acabamos de aprender. =)

* Sistema que precise de muita interação gráfica com o usuário
* Sistema que realiza muito processamento paralelo
* **Sistema que faz muito I/O**
* Sistema que realiza muito processamento em memória

Com certeza um código que faz muito I/O é excelente para o Node.js graças à sua forma de execução que é toda assíncrona.