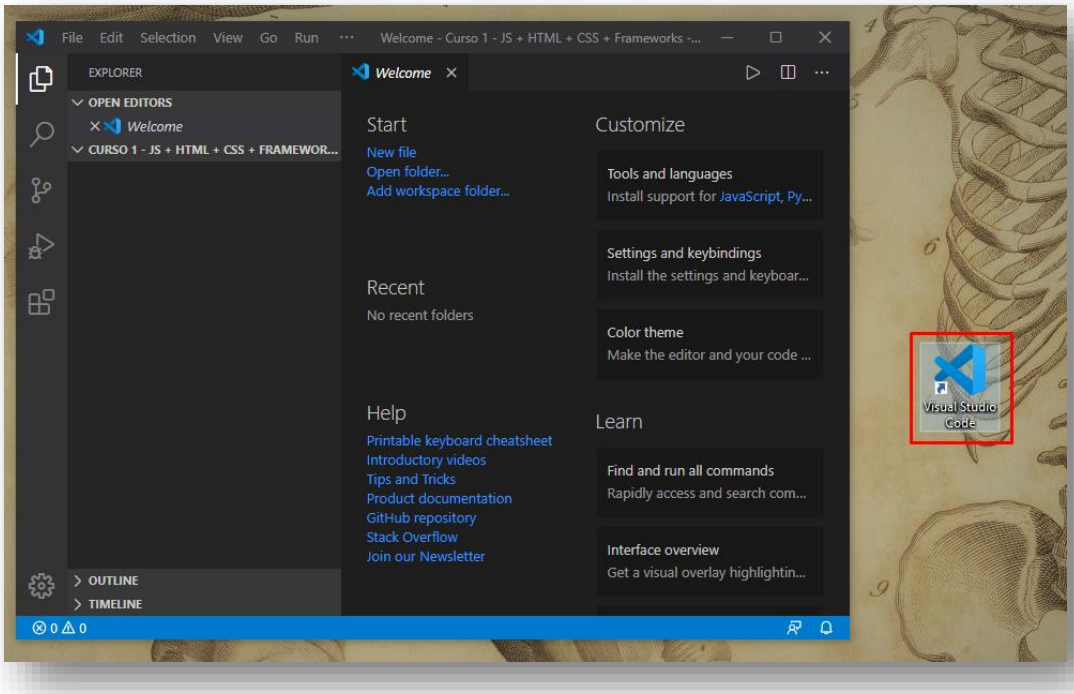
1 INTRODUÇÃO

JavaScript é uma linguagem de programação interpretada estruturada, de script em alto nível com tipagem dinâmica fraca e multiparadigma. Juntamente com HTML e CSS, o JavaScript é uma das três principais tecnologias da World Wide Web.

2 CONFIGURAÇÃO DO AMBIENTE

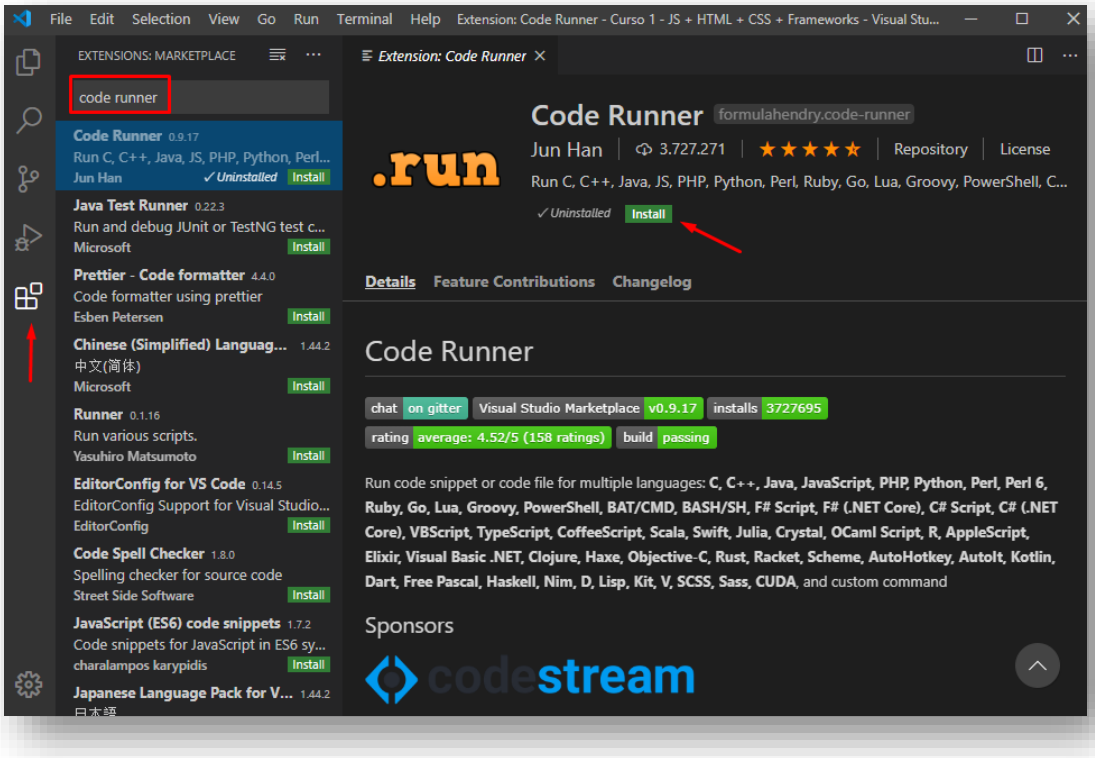
Devemos realizar o download da aplicação NodeJS [https://nodejs.org/en/]. Com ele é possível criar aplicações Javascript para rodar como uma aplicação *standalone* em uma máquina, não dependendo de um browser para a execução, como estamos acostumados.

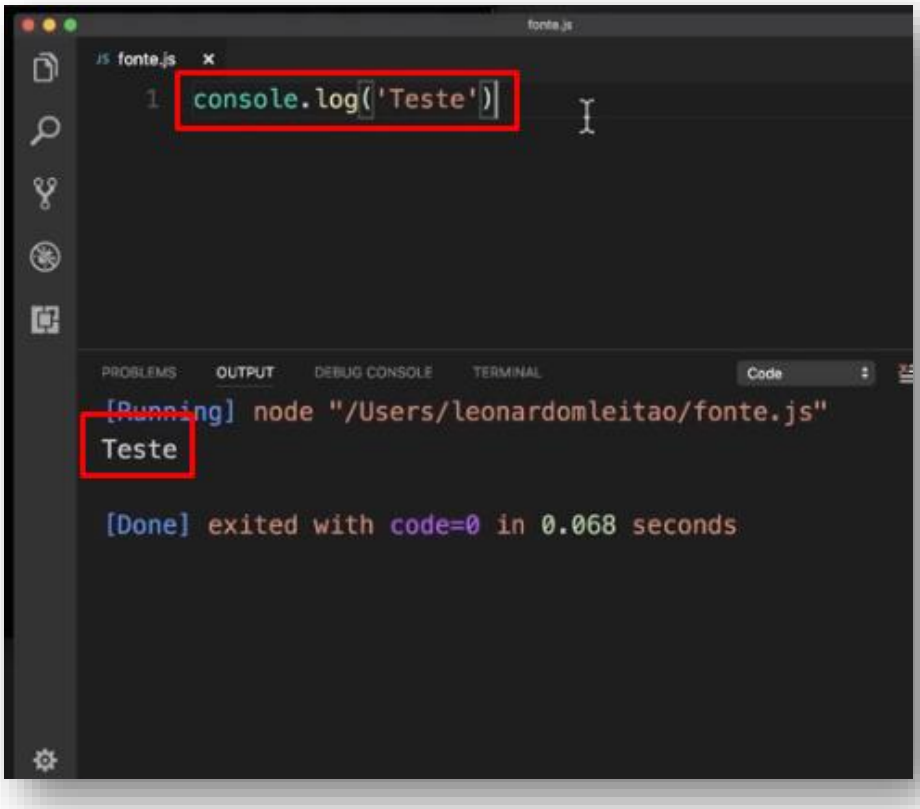
A IDE (ambiente desenvolvimento) que iremos utilizar será o Visual Studio Code, mais conhecido como VS Code [https://code.visualstudio.com/]. Esta IDE permite uma vasta gama de aplicações, é organizada, e dispõe de diversas ferramentas para desenvolvedores.



No VS Code, para nossas aplicações, devemos instalar uma extensão, existente na aba de extensões - seu nome é “Code Runner”. Esta permitirá a execução de arquivos JavaScript diretamente no ambiente de desenvolvimento. Esta extensão entrega o arquivo para o NodeJS, e o produto da execução do código do arquivo é repassado para o VS.

Para utilizar essa ferramenta, depois de instalada, e existindo um código JS para ser executado, clicamos CRTL + ALT + N. Desta forma, será aberto um terminal, com o produto do código.



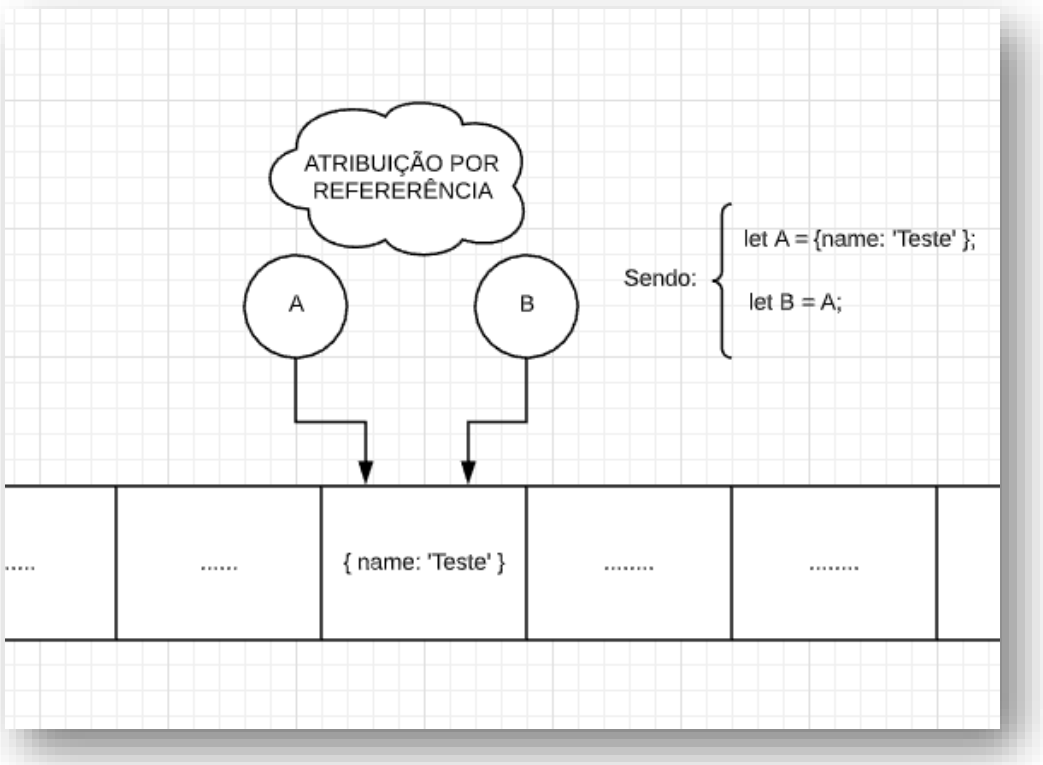


3 ATRIBUIÇÃO POR VALOR E ATRIBUIÇÃO POR REFERÊNCIA

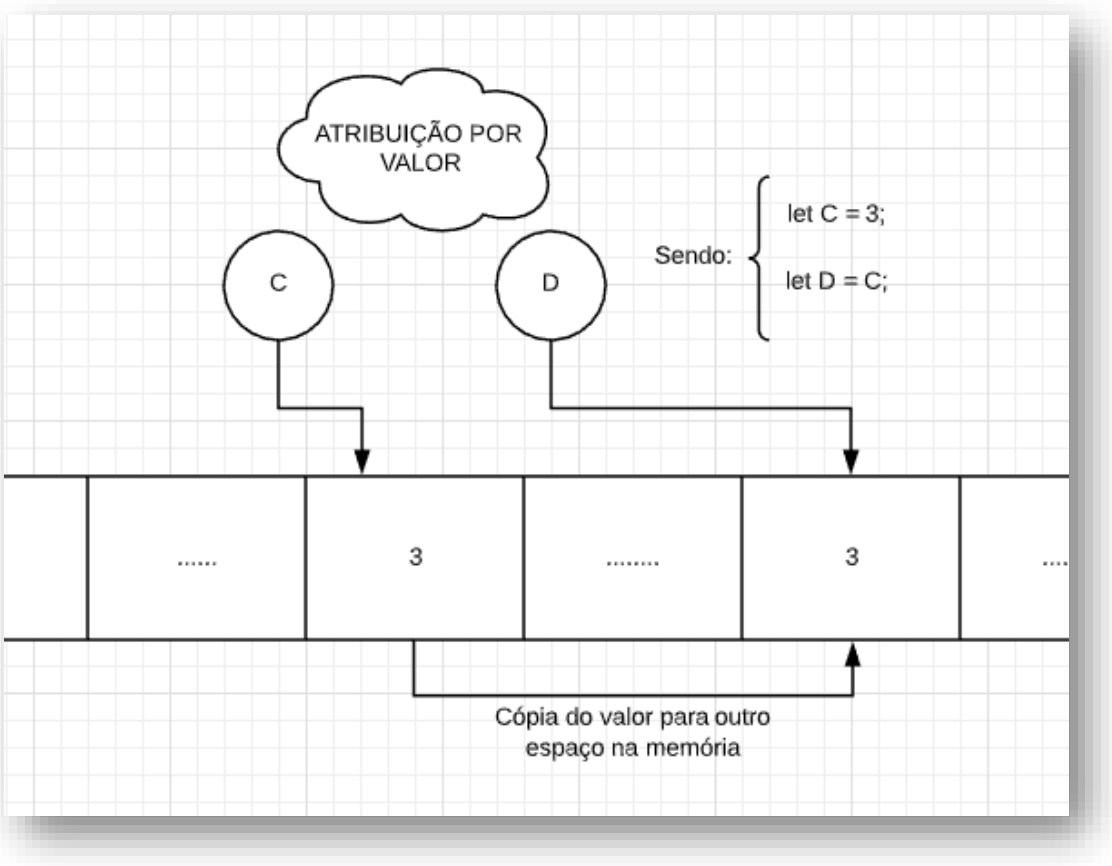
Quando atribuímos um valor a uma variável, ou a uma constante, na verdade estamos dizendo que estarão apontando para o local na memória em que o valor estará armazenado. Por exemplo, se existir uma variável **var A = {name : 'Teste'}**, este mesmo objeto estará sendo armazenado em um local da memória, e sendo apontado pela variável A; funcionará como um ponteiro de C/C++.

Mantendo em mente a mesma situação acima, em que existe uma variável A, que recebeu um objeto, **A = {name: 'Teste'}**, imagine uma segunda variável agora, B, que recebe A, sendo **let B = A**. Considere, em seguida, uma modificação realizada em B, em que escrevemos B = {name: 'Modificação'}. Se tentarmos imprimir o valor de B, este será o novo valor, mas e se tentarmos imprimir o valor de A? A variável B apenas recebeu uma cópia do valor de A? Não foi isto que ocorreu.

No caso descrito acima, temos uma atribuição por referência, na verdade B passou a apontar para o mesmo endereço que a variável A apontava. Assim, ao modificarmos o valor de B, modificamos também o valor de A, pois A e B apontam para o mesmo local da memória, que contém agora um novo valor: {name: 'Modificação'}. Agora A = {name: 'Modificação'}, assim como B.



Em contraste a isto, existe a atribuição por valor, que ocorre quando trabalhamos coma atribuições de valores de tipos primitivos. Considere agora uma variável let C = 3, e uma variável let D = C, e em seguida uma modificação em D, sendo D = D + 1. Se imprimirmos o valor de D, iremos ser informados que o valor é 4, mas se imprimirmos o valor de C, iremos ser informados que o valor é 3. Isto ocorre, porque neste caso D recebeu C, que aponta para um espaço na memória em que está alocado um valor de tipo primitivo, neste caso, um mero número. Assim, ao invés de C e D passarem a apontar para o mesmo local da memória, D recebeu uma cópia do valor , que passou a ser alocado em outro local da memória.



4 NULL E UNDEFINED

Null se refere a ausência de valor apontado por uma variável, isto é, a variável de valor Null existe, mas o fato de valer Null, significa que não está apontando para nenhum local na memória.

Por sua vez, Undefined é a variável declarada, mas sem valor, isto é, que existe, mas nunca foi inicializada. Embora pareça estranho, em JS “Undefined” é diferente de “is not defined”, este último erro aparece quando tentamos manipular uma variável que nunca foi declarada, isto é, que não existe.

5 TIPAGEM DE VALORES

Em JavaScript a tipagem de valores é fraca, isto é, no geral é reconhecida automaticamente, não sendo necessário definir os tipos primitivos; um número como inteiro, como float, ou uma letra como char, uma palavra como string, e assim por diante. Ao atribuirmos um valor a uma variável, ou a uma constante, **o Node já reconhece o seu valor**, e o seu respectivo tipo. Podemos consultar isto com o comando console.log (typeof variável), que irá imprimir o tipo identificado do valor da variável.

Existem algumas exceções, sendo o caso mais importante de ser lembrado, o do tipo número, que deve ser definido como Number; sendo inteiro, ou não, um número em JavaScript pertence ao tipo de valor Number. Comumente o JavaScript, devido a algumas peculiaridades de sua forma de interpretação de dados, confunde, strings com números, ou até mesmo simbologia de soma (+), com concatenação de strings.

O símbolo +, em JS, serve tanto para somar números, quanto para concatenar strings. Devido a este duplo uso, desta simbologia, uma operação como soma = 10 + 10, pode resultar em uma string '1010', ao invés do valor numérico 20. Para resolver isto, e outros possíveis problemas não mencionados, para declarar uma variável de valor numérico, escrevemos, por exemplo, se quisermos uma variável do tipo let, de nome numero, que contenha o valor numérico 20: let numero = Number(20). Da mesma forma, em operações matemáticas, podemos escrever, por exemplo, soma = Number(20) + Number(20).

6 TIPAGEM DE "VARIÁVEIS": VAR, LET E CONST

6.1 TIPO CONST

Começando pelo tipo mais simples, o tipo Const é utilizado para a declaração de uma constante. Uma constante só pode ser declarada uma única vez, e também não pode ter o seu valor modificado, por isso o nome "constante". **Para variáveis, possuímos os tipos Var e Let.**

6.2 TIPO VAR

A variável Var pode ser declarada mais de uma vez dentro de um mesmo escopo, isto é, podemos redeclarar uma variável que tenha o mesmo nome de uma outra variável var, produzindo um processo de reescrita na memória. Também podemos modificar o valor de uma variável Var, sem necessariamente a redeclarar. **O ponto mais importante deste tipo de variável, é que possui apenas dois tipos de escopo:** Escopo Global, e o Escopo de Função.

Escopo é como um local na codificação, um bloco de códigos { } é um escopo, uma função é outro escopo, e o ambiente geral, de codificação, é o escopo Global. Em diversas linguagens, as

variáveis só são reconhecidas dentro do seu escopo de origem, isto é, uma variável declarada dentro de um bloco de código { }, só existe dentro deste escopo.

Portanto, quando dizemos que o tipo Var, em JavaScript, não possui o escopo de Bloco, dizemos que uma variável Var declarada dentro de um bloco{ }, tem seu escopo de Bloco ignorado, e é reconhecida como **pertence ao escopo Global**. Se declararmos uma variável var A, dentro de um bloco de código, qualquer manipulação que realizarmos fora do bloco de código, envolvendo a variável var A, irá afetar diretamente o seu valor.

No entanto, isto não funciona se a variável var A for declarada dentro de uma função, pois apesar de não possuir escopo de Bloco, o tipo var possui o escopo de função. Se a variável var A for declarada dentro de uma função, ela só existirá dentro do escopo da função, tendo sua existência completamente ignorada para o resto do código.

6.3 TIPO LET

Por sua vez, e em nítido contraste, a variável de tipo Let, não permite redeclaração dentro de uma variável dentro de um mesmo escopo, além de possuir três níveis de escopo: **Global, de Bloco e de Função**. Isto é, uma variável de tipo Let, declarada dentro de um bloco de código qualquer, só existirá dentro do bloco.

Com este nível a mais, a variável Let possui uma especificidade muito maior, o que evita possíveis falhas ao longo da codificação, como reescrita acidental de valores. Três variáveis Let de mesmo nome, escritas em escopos diferentes, são três variáveis diferentes, isoladas e independentes. Se declararmos uma variável Let dentro de um bloco, contido dentro de uma função, esta variável não irá existir nem mesmo para a função, mas apenas para o bloco de origem, em especifico. Veja a imagem abaixo, e compare-a com a imagem de cima:

```
let numero = 105;
{
    let numero = 2;
    console.log('dentro =', numero); //irá imprimir o numero 2
}
console.log('fora =', numero); //Irá imprimir, de novo, o numero 105
```