**SCRIPTS >**

Os scripts que são carregados usando o atributo “async” irão baixar o script sem bloquear a renderização da página e irão executar imediatamente após o script terminar de ser disponibilizado. Nesse modo você não tem garantia nenhuma que os scripts carregados irão rodar em uma ordem especifica, mas saberá que dessa forma eles não irão impedir o carregamento do restante da página. O melhor uso para o “async” é quando os scripts de uma página rodam de forma independente entre si e não dependem de nenhum outro script.

Por exemplo, se você tiver os elementos script:

**<script async src=”js/vendor/jquery.js”></script>**

**<script async src=”js/script2.js”></script>**

**<script async src=”js/script3.js”></script>**

Voce não pode garantir que o script, Jquery.js carregara antes ou depois do script2.js e script3.js. Nesse caso se alguma função desses scripts dependerem de algo vindo do jquery, ela produzirá um erro pois o jquery ainda não foi definido ou carregado quando os scripts executaram essa função.

“async” deve ser usado quando houver muitos scripts rodando no background, e você precisa que estejam disponíveis o mais rápido possível. Por exemplo, talvez você tenha um arquivo de dados de um jogo para carregar que serão necessários assim que o jogo iniciar, mas por enquanto, você so quer entrar e ver a tela de carreagmento, a do titulo do jogo eo lobby, sem ser bloqueado pelo carregamento desses scripts.

.....................................

Scripts que são carregados utilizando o atributo “defer” irão rodar exatamente na ordem em que aparecem na pagina e serão executados assim que o script e o conteúdo for baixado.

**<script async src=”js/vendor/jquery.js”></script>**

**<script async src=”js/script2.js”></script>**

**<script async src=”js/script3.js”></script>**

Todos os scripts com o atributo “defer” irão carregar na ordem que aparecem na pagina. No segundo exemplo, podemos ter a certeza que o script “jquery.js” ira carregar antes do “script2.js” e “script3.js”. Os scripts não irão rodar sem que antes todo o conteúdo da pagina seja carregado, que no caso, é muito útil se os seus scripts dependem de um DOM completamente disponibilizado em tela (por exemplo, scripts que modificam um elemento).

.....................

**ADICIONANDO VARIAVEIS>**

**“var ou let ” = significa variável**

**Vaga a1 = carro1 – sinal de “= (igual)” em varial significa “recebe”**

**Então se lê da seguinte forma (Vaga a1 “= recebe” carro1)**

**Para fazer o carro2 entrar na vaga a1, precisamos retirar o carro1 da seguinte forma,**

**A1 = carro 2**

**Ou deixar a “vaga” sem nenhum**

**A1=null**

**..................**

**var n1 = 5**

**var n2 = 8.5**

**var n3 = 15**

**estes números serão colocados dentro das “variáveis”**

**Existem variáveis de tamanhos diferentes**

**Var s1 = “Java Script”**

**Var s2 = ‘Curso em Vídeo’**

**Var s3 = `Guanabara`**

**Existem 3 formas de delimitar uma “string” em java script:**

**Em Java Script podemos utilizar 3 tipos de aspas “aspas dupla”, ‘apostrifi’ e `crazi`**

**Os nomes de cada variável a gente chama de identificadores, existem algumas regras para por nos identificadores:**

**A primeira regra é**

**\*Podem começar com letra, $ ou \_ /\*Mas quase não se usa\*/**

**\*Não podem começar com números**

**\*É possível usar letras ou números**

**\*É possível utilizar acentos e símbolos**

**\*Não podem conter espaços**

**\*Não podem ser palavras reservadas /\*Como por exemplo: function, alert, var e palavras que são utilizadas em JS como comandos\*/**

**Para abrir o TERMINAL NODE no VISUAL STUDIO CODE CLICAR CRTL + SHIFT + CRAZI**

**CTRL + SHIFT + ‘**

**Letras maiúsculas e minúsculas fazem diferença na hora de dar nome para o var**

**DATA TYPES**

**(typeof)**

**Number (infinity e NaN (Not a Number)**

**String**

**Boolean**

**Null – considerado (object)**

**Undefined**

**Object (array – considerado um vetor)**

**Function**

**Var n = 200 – seu typeof n é “number”**

**Agora se fizermos – var n = “200” – seu typeof passa a ser uma “string”**

**Typeof [] – object**

**Typeof {} – object**

**Typeof function [] {} – function**

**Typeof null – object**

**................**

**INTERAÇÃO COM O SITE**

<script> /\*dentro do <script> se programa em JAVA SCRIPT\*/

        var nome = window.prompt('Qual é o seu nome?') //Perguntar o nome..

        window.alert('É um grande prazer em te conhcer, ' + nome + '!')  //Concatenação //Quando quiser colocar exclamação faça, '' + nome + '!'

    </script>

**STRING PARA NUMERO**

**Number.parseInt(n) – numero inteiro**

**Number.parseFloat(n) – numero parcial 1.5, 2.5 ... etc**

var n1 = NumberparseInt(window.prompt ('Digite um número: '))

        var n2 = NumberparseInt(window.prompt ('Digite outro número: '))

        var s = n1 + n2

        window.alert('A soma dos valores é : ' + s)

**NumberparseInt (significa numero inteiro)**

**NumberparseFloat (significa numero parcial 1.5, 2.5, 3.5 ...)**

**Number(Significa os dois)**

**Fazendo conversão de uma variável “string” para “number”**

**Ou podemos somente utilizar da seguinte forma**

**PODEMOS FAZER DESTA FORMA MAIS CORRETA TB**

<script>

        var n1 = Number(window.prompt('Digite um número:'))

        var n2 = Number(window.prompt('Digite mais um número: '))

        var s = n1 + n2

        window.alert(`A soma entre ${n1} e ${n2} é igual á ${s}`)

    </script>

**FORMATANDO STRINGS**

**Var s = ‘JavaScript’**

**‘i’m learning s’ //não faz interpolação**

**‘i’m learning + s’ //usa concatenação**

**`i’m learning ${s}` //usa template string**

**..**

**Podemos fazer da seguinte forma também**

**Var s = ‘JavaScript’**

**s.length //quantos caracteres a string tem**

**s.toUpperCase() // tudo para ‘MAIÚSCULAS’**

**s.toLowerCase() // tudo para ‘minúsculas’**

<script>

        var nome = window.prompt('Por favor, nos informe o seu nome!')

        document.write('Seu nome tem  ${nome.lenght} letras')

        document.writeln('Seu nome em minúsculo fica assim: ${nome.toLowerCase()} ')

    </script>

**Para fazer a conversão de ponto para virgula no caso de um numero muito grande que se utiliza virgula sera necessario fazer da seguinte forma**

**Var n1 = Number(‘numero 15560’)**

**N1.toFixed (2)**

**Ele vai ler assim 155.60**

**Para mudar de . “ponto” para , “virgula”**

**N1.toFixed(2).replace(‘.’ , ‘,’)**

**Ele vai ler assim = 155,60**

**> var n1 = 15560**

**undefined**

**> n1**

**15560**

**> n1.toFixed(2)**

**'15560.00'**

**> n1.toFixed(2).replace('.',',')**

**'15560,00'**

**PARA TRANSFORMAR ESTE VALOR DE 15560 EM REAL MOEDA REAL**

**FICARIA COM O CÓDIGO DESTA FORMA**

**> n1.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL'})**

**'R$ 15.560,00'**

**OPERADORES:**

**5 + 3 / 2 = 4 /\*Na verdade quando a mesma expressão tem soma (+) e divisão (/) primeiro se faz primeiro a divisão, então ficaria da seguinte forma 5 + 3 / 2 = 6.5 por que 3/2 = 1.5 e 1.5 + 5 = 6.5 \*/**

**Precedência:**

**Qualquer linguagem de programação tudo, será analisada nesta ordem >**

**() – Primeiro**

**\*\* / % - Multiplicação, Divisão e Resto da divisão**

**+ e – por último**

**Quando quisermos guardar esses dados precisamos fazer algumas atribuições simples:**

**Var a = 5+3 o resultado é 8 e este 8 irá ficar guardado na variável “a”**

**Var b = a “8” % 5 sendo que “a” “é igual á 8” e o resultado será 3**

**Var c = 5 \* b “3 resultado do var b = ” \*\* 2 então fica 3 ao quadrado igual 9 e 9 \* 5 = 45**

**Var d = 10 – a / 2 “var a = 8 / 2 = 4 e 10 – 4 = 6 “ficando com o resultado de 6”**

**Var e = 6 \* 2 / d “6\*2 = 12 / 6 = 2 “resultado é 2”**

**Var f = b % e + 4 / e “ o resultado é 3 primeiro faremos b % e que é igual á 1 e 4 + e é igual a 2 e 2+1 = 3**

**Aritméticos: +, -, \*, /, % (% resto da divisão) e \*\* (\*\* significa ao quadrado) (no caso da % significa o resto da divisão inteira ou seja 5%2 = 1 por que 5/2 é 2 e 2\*2 é 4 logo resta 1 para chegar até o 5) Operadores para Cálculo (OPERADORES BINARIOS QUE PRECISAM DE DOIS OPERANDO... OU SEJA 5 + 2 ) (5 / 2) (5\*2)**

**Quando quisermos fazer uma soma primeiro e depois a divisão devemos fazer da seguinte forma: (5+3) / 2 = 4 ... caso coloque desta forma 5 + 3 / 2 o resultado será 6.5 por fazer primeiro a divisão e depois a soma.**

**Atribuição:**

**Relacionais: <(menor) >(maior) >=(maior ou igual) <=(menor ou igual) ==(igual) !=(diferente)**

**5 > 2 = true, 7 < 4 = false, 8 >= 8 = true, 9 <= 7 = false, 5 == 5 = true, 4 != 4 = false (São chamados de booleans – True or False) (== igual) (=== idêntico) então 5 == ‘5’ = true ----5 === ‘5’ = false (por que são iguais mas não idênticos)**

**Lógicos : ! , && , || --**

**! Negação -- (Não, tratado como operador unário somente um operando, depois do ! colocar um (true or false ou uma expressão que vai dar true or false) ou seja quando algo que não é verdadeiro (false) uma coisa que não é falsa (true)**

**&& Conjunção -- (binário) true && true: true 🡪 true && false: false 🡪 false && true: false 🡪 false && false: false**

**|| Disjunção -- true || true: true 🡪 true || false: true 🡪 false || true: true 🡪 false || false: false**

**Ternário: ? e : (teste ? true : false) por exemplo: média >= 7.0 ? “aprovado” : “reprovado”**

**Média >= 7.0 ? ‘Aprovado’ : ‘Reprovado’ (ou seja) (se a média for maior ou igual 7.0, aparecerá na tela ‘aprovado’ no caso se for de 7 pra cima, no caso de um número menor aparecera na tela ‘reprovado’)**

**Typeoff:**