Variáveis e tipos de dados

1 - Declaração de variáveis

Em programação uma variável é um espaço na memória do computador, que podemos usar para armazenar valores.
Se quisermos usar uma variável, a primeira coisa que devemos fazer é declará-la, isto é, criar essa "caixa" onde armazenaremos os valores.
Em Java Script, as variáveis são declaradas usando a palavra var seguido do nome com o qual iremos identificá-lo.
Por exemplo:

```
var preco
```

Temos algumas regras para criar nome de variaveis:

- Não pode começar com número
- Não pode ter caracteres especiais
- Não pode utilizar acentos
- Não pode utilizar espaços
- Deve começar com letra minuscula e seguir o padrão camelCase, onde cada palavra é iniciada com maiúsculas e unidas sem espaços.

Para passar para o próximo exercício, declare duas variáveis, onde uma se chamará idade e outra que se chamará peso. Em seguida, execute a sua resposta.

Exercício Feito:

```
var idade
var peso
```

---

2 - Armazenando dados

Nós já vimos como declarar uma variável. Agora precisamos atribuir um valor à ela.
Para salvar um valor em uma variável, usamos o sinal = e, em seguida, o valor que queremos armazenar.
Por exemplo:


```
var meses = 12
var quantidadeDeAlunos = 30
```

Para passar para o próximo exercício, declare duas variáveis (idade e peso) e atribua a elas um valor numérico. Em seguida, execute a sua resposta.

Exercício Feito:

```
var idade = 19
var peso = 98
```

---

3 - Sempre falta um ponto e vírgula?

Você pode ter visto em Java Script alguns comandos, instruções que terminam em ";".
Em Java script, o sinal ";" é opcional. Mas é importante que você utilize para o computador conseguir entender onde termina o comando.
Para começar, no exercício anterior vimos:

```
var meses = 12
var quantidadeDeAlunos = 30
```

Também poderia ter sido escrito da seguinte forma, como abaixo:

```
var meses = 12;
var quantidadeDeAlunos = 30;
```

A partir dos próximos exercícios, usaremos o ";" como no exemplo anterior.
Agora vamos praticar! Crie os variáveis meses e quantidadeDeAlunos, igual ao exemplo. Após criar as variáveis, basta executar a sua resposta. Neste exercício, estamos apenas praticando a declaração correta do código.

Exercício Feito:

```
var meses;
var quantidadeDeAlunos;
```

---

4 - Variáveis e Operações

Vimos como declarar uma variável e atribuir um valor a ela e, provavelmente, agora você se pergunta, e qual é a utilidade de armazenar dados em variáveis?
As variáveis nos permitem reutilizar os dados atribuídos, simplesmente invocando seu nome.

```
var numero = 124;
console.log(numero);
// Podemos usar console.log para imprimir o valor que tem atribuído à variável numero. E seu resultado será 124.
```

Algo muito importante também, é, assim como podemos fazer operações matemáticas como adicionar (+) ou subtrair (-) números, podemos fazer o mesmo com as variáveis que os referenciam.
Por exemplo:

```
var numero = 124;
var numeroSeguinte = numero + 1;
console.log(numeroSeguinte);
// O resultado que será impresso na tela será o valor atribuído a variável numero somado a 1, portanto, o valor atribuído a numeroSeguinte será 125.
```

Vamos ver se está entendido: **Declare e atribua** duas variáveis, numeroA e numeroB, e então nas variáveis resultadoSoma, resultadoSubtracao, resultadoMultiplicacao e resultadoDivisao armazene os cálculos feitos utilizando numeroA e numeroB nas variáveis de resultado, de modo que o cálculo matemático se altere de acordo com o título da variável, por exemplo. Exemplo var resultadoSoma = (numeroA + numeroB) Como nossa variável resultadoSoma indica nós devemos fazer uma soma utilizando as variáveis numeroA e numeroB. Para testar a sua resolução, você precisa apenas executar a sua resposta.
Exemplos:
soma: a + b
subtração: a - b
multiplicação: a * b
divisão: a / b

Exercício Feito:

```
var numeroA = 2;
var numeroB = 2;
var resultadoSoma = (numeroA + numeroB);
var resultadoSubtracao = (numeroA - numeroB);
var resultadoMultiplicacao = (numeroA * numeroB);
var resultadoDivisao = (numeroA / numeroB);
```

---

5 - Outro tipo de dado

Além dos números, há mais um tipo de dados em Java Script.

Este tipo de dado é:

**** String***

Os dados do tipo **String** são conhecidos como cadeia de caracteres e nos permitem representar qualquer combinação de letras, números e/ou símbolos.
Para definir uma **string,** é necessário que o texto em questão esteja entre aspas:

```
"João"
"Meu nome é João"
"125 + 125 = 250!"
```

Para gerar uma String, posso simplesmente atribuir o texto a uma variável; por exemplo:

```
var meuPlaneta = "Terra";
console.log(meuPlaneta) //Isto irá imprimir na tela “Terra"
```

> Para passar para o próximo exercício, declare a variável saudacao e atribua o texto "Hello World". E finalmente imprima o valor da variável na tela usando o `console.log`

Exercício Feito:

```
var saudacao = "Hello World";
console.log(saudacao)
```

6 - Mais operações

E o que acontece se eu quiser unir textos? Neste caso, o símbolo + nos ajudará a concatenar, isto é, juntar nossas strings.

FIQUE DE OLHO, NÃO ESQUEÇA OS ESPAÇOS!

Vamos tentar:

```
Var nome = "Ronaldo";
Var nacionalidade = "Brasileiro";
console.log(nome + " é " + nacionalidade)
// O resultado seria: "Ronaldo é Brasileiro"
```

Declare e atribuia duas variáveis, uma com o seu nome e outra com o seu sobrenome, e então na variávelnomeCompleto mantenha tudo junto.
Exemplo: "Cesar Michelin"
Observação: Não esqueça de concatenar um espaço entre as variáveis!

Exercício Feito:

```
var nome = "Diego";
var  sobrenome = "Fonseca";
var nomeCompleto = nome + " " + sobrenome;
console.log(nomeCompleto);
```

---

7 - Outras operações

E o que acontece se eu adicionar números com textos?

```
var rua = "Av Paulista";
var numero = 1578;
console.log (rua + " " + numero)
// O resultado seria: "Av Paulista 1578"
```

Como você viu, se eu adicionar um número e um texto, ele se tornará parte da string.

Declarar e atribuir três variáveis, uma com o seu 'nome', outra com o seu 'sobrenome' e outra com a sua 'idade'. Então, declare uma variável com nome resultado e atribua uma string unindo suas variaveis da seguinte forma: ***" João Silva terá 30 anos"*** e utilize o console.log para mostrar o que acontece. Respeite os espaços!

Exercícios Feitos:

```
var nome = "Diego";
var sobrenome = "Fonseca";
var idade = 30;
var resultado = nome + " " + sobrenome + " " + "terá" + " " + idade + " " + "anos";
console.log (resultado)
```

---

8 - Trocar valores

As variáveis além de armazenar informações, e como seu nome diz, podem variar o valor que armazenam. Por exemplo:

```
vencedor = "Pelé";
vencedor = "Marta";
console.log (vencedor);
// resultado seria: "Marta"
```

Como você deve ter notado, chamando a variável e fazendo uma retribuição a um valor eu posso mudar seu conteúdo.

Vamos ver se entendeu: Defina a variável numeroA com um valor de 30 e defina a variável numeroB com um valor de 45. Em seguida, faça a variável numeroA armazenar o valor da variável numeroB e vice-versa.
Preste atenção na dica para realizar o exercício.

Exercício Feito:

```
var numeroA = 30;
var numeroB = 45;

var numeroC = numeroA;
var numeroC = numeroB;

console.log (numeroA)

console.log (numeroB)
```

---

9 - Mais tipos de dados

Já vimos os valores numéricos e as strings, mas em JavaScript há mais um tipo de dado:

Booleano

O Boolean é conhecido como booleano e permite representar dois valores lógicos, são eles:
true
Representa o valor de algo ser verdadeiro
false
Representa o valor de algo sendo falso
Para gerar um booleano, posso simplesmente atribuir o valor true ou false a uma variável.
Por exemplo:

```
var valorVerdadeiro = true;
console.log (valorVerdadeiro) // Isso irá imprimir "true" na tela
```

Para continuar, defina uma variável gostoDeSorvete e atribua a ela um valor booleano. Então, utilize console.log(gostoDeSorvete) para ver o que acontece!

Exercício Feito:

```
var gostoDeSorvete = true;
console.log(gostoDeSorvete)
```

10 - Mais sobre os booleanos

O poder real dos booleanos é que eles podem surgir ao fazer comparações de valores diferentes com alguns operadores matemáticos.
Por exemplo, sabemos que se perguntarmos a alguém "2 é maior que 1?" A pessoa nos dirá "Sim é verdade, 2 é maior que 1", o mesmo acontece em JavaScript quando escrevemos o seguinte:

```
console.log(2 > 1) // Isso imprimirá "true" na tela
```

Isso significa que "2 > 1" tem um valor de **true**. Nós também poderíamos ter escrito o mesmo código da seguinte forma

```
var valorVerdade  = 2 > 1;
// Como vimos 2> 1, ele retorna um valor de verdade e o atribuímos a uma variável
console.log (valorVerdade) // Isso imprimirá "true" na tela
```

E se perguntarmos a alguém "2 é menor que 1?" A pessoa dirá "Isso é falso, 2 não é menor que 1", o mesmo acontece em JavaScript quando escrevemos o seguinte:

```
console.log(2 < 1) // Isso imprimirá "false" na tela
```

Para continuar, defina duas variáveis: umNumeroPequeno e umNumeroGrande, e atribua a elas valores numéricos diferentes de acordo com seus nomes. Então defina a variável eMenor e atribua o resultado da comparação se umNumeroPequeno é **menor** do que umNumeroGrande; e defina a variável eMaior, com o resultado da comparação umNumeroPequeno é **maior** que umNumeroGrande.

Funções

1 – Declaração.

Vamos começar com o básico:
As funções consistem em 4 partes. **A palavra reservada** function**,** o **nome** com o qual vamos chamar a função, os parênteses onde irá os **parâmetros** (Caso a função não precise de parâmetros, ainda sim, devemos colocar os **parênteses**), as **{} chaves** que dentro teremos o código que queremos que a função **execute.**
Se quisermos que uma função retorne um valor, precisaremos da palavra-chave return. Depois disso, tudo o que queremos irá retornar.
Com as funções, podemos fazer muitas coisas, mas vamos começar com algo fácil. Veja esse exemplo onde a função a seguir dobra o valor do número que passamos para ela:

```
function  dobro(numero)  {
    return  numero*2;
}
```

Como podemos ver, acima nós declaramos uma função com a palavra reservada **function** e o nome dobro. Entre os parênteses, colocamos o parâmetro **numero** que será substituído pelo número que damos ao fazer a chamada da função. Para terminar, com a palavra **return** dizemos que queremos que a função retorne 2 vezes o **numero** que lhe demos.
Para executar a função e fazer o que queremos, nós precisamos chamá-la pelo nome e digitar o número que desejamos usar entre os parênteses.

```
dobro(5);//Isto irá devolver 10
dobro(1.5); //E isto irá devolver 3
```

Neste exercício, crie a **função dobro** tendo como base o exemplo acima. O objetivo é fixar o seu entendimento sobre a estrutura de declaração de uma função.

Exercício Feito:

```
function dobro(numero) {
  return numero * 2;
}
```

2 - Fórmulas e funções.

Já vimos que podemos fazer cálculos matemáticos simples e de maior complexidade.
Agora vamos para algo mais interessante.
Queremos criar funções que nos permitam calcular a área e o perímetro de um círculo.
Vamos criar uma função perímetro que nos diga o perímetro de um círculo quando damos a ele o raio como parâmetro.
Também a função área que nos dá a área de um círculo quando recebe o raio como parâmetro.
Lembre-se de usar o valor de 3.14 no lugar do π.

Exercício Feito:

A função perímetro pode ser desenvolvida da seguinte forma:

```
function perimetro (raio){

return (3.14 * raio * 2)
};
```

A função área pode ser desenvolvida da seguinte forma:

```
function area (raio){

return (3.14 * raio * raio)
};
```

O perímetro consiste na medida do contorno de um objeto bidimensional, isto é, compreende a somatória de todos os lados de uma figura geométrica. O cálculo do perímetro pode ter diversas aplicações práticas.

3 - Mais funções

Então, para montar uma função, há coisas que não podemos deixar de fora. A palavra function, o **nome da função**, os **parâmetros** dentro dos parenteses (se necessário), o **código** dentro das **{}** e um return dentro do código, se quisermos que ele retorne algo para nós.

Vamos praticar juntos: escreva agora uma função metade, que pega um número como **parâmetro** e retorna sua metade.

Exercício Feito:

```
function metade(numero) {
  return numero / 2;
}
```

---

4 - Fazendo contas.

Nós já vimos os operadores matemáticos básicos +, -, /e*
Agora podemos utilizar funções para fazer estas contas de uma forma mais fácil.

```
function somar(numero1, numero2){
 return numero1 + numero2;
}
```

Escreva a função multiplicar que receba dois parâmetros, multiplique-os e retorne o resultado da multiplicação.

Exercício Feito:

```
function multiplicar(numero1, numero2){
  return numero1 * numero2;
}
```

5 - Usando funções.

Tendo essas pequenas funções, podemos combiná-las para fazer coisas mais complexas. Por exemplo, se quisermos somar dois números e depois multiplicá-los por 3, poderíamos fazê-lo da seguinte maneira:

```
function somar (num1, num2) {
  return num1 + num2;
}
function triploDaSoma(num3, num4) {
  var resultadoDaSoma = somar(num3, num4);
  return resultadoDaSoma * 3;
}
```

Aqui vemos que tendo a função **somar()** definida, podemos chamá-la dentro de outra função(neste caso dentro da função triploDaSoma()), guardar o seu resultado da execução em uma nova variável e tornar o trabalho mais fácil.

Agora pedimos que você declare uma função chamada triploDaSoma() que recebe dois parâmetros. Então você tem que adicionar ambos e retornar três vezes o valor do resultado da soma dos dois parâmetros . Para fazer isso, você já conta (mesmo que não veja declarado) com a **função triplo**, que recebe **um parâmetro** e retorna o valor dele multiplicado por três.

Exercício Feito:

```
function triploDaSoma(num1, num2){

var resultadoDaSoma = num1 + num2;

return triplo (resultadoDaSoma);
}
```

6 - Fórmulas e funções.

Já vimos que podemos fazer cálculos matemáticos simples e de maior complexidade.
Agora vamos para algo mais interessante.
Queremos criar funções que nos permitam calcular a área e o perímetro de um círculo.

Vamos criar uma função perimetro que nos diga o perímetro de um círculo quando damos a ele o raio como **parâmetro**.
Também a função area que nos dá a área de um círculo quando recebe o raio como **parâmetro**.
Lembre-se de usar o valor de 3.14 no lugar do π.

Exercício Feito:

```
function perimetro (raio){
return (3.14 * raio * 2)
};
function area (raio){
return (3.14 * raio * raio)
};
```

---

7 - Operando strings.

E o que podemos fazer que não seja matemática? Podemos trabalhar com **strings!** E se você se recorda, as **strings** são um tipo de dados. Representamos as cadeias de texto escrevendo dentro de aspas (“ “) literalmente.

E o que podemos fazer então com as strings? Por exemplo, podemos calcular quantos caracteres existem no total, e para isso iremos utilizar um recurso chamado **length,** onde colocamos a **string** seguida **de .length.** Veja alguns exemplos:

```
"biblioteca".length // devolve 10
"babel".length    // devolve 5
```

E lembre-se que também podemos somar **strings**! Embora que, o termo correto seja concatenar, ou seja, obter uma nova string juntando duas ou mais strings.

```
"aa" + "bb"    // devolve "aabb"
"aa" + " " + "bb" // devolve "aa bb"
```

Vamos testar: Iremos criar uma função chamada tamanhoNomeCompleto, que recebe um nome e um sobrenome como parâmetros, e que irá devolver o tamanho total, contando um espaço extra para separar ambos:

```
tamanhoNomeCompleto("Enzo", "Silva")
// devolve 10
```

Exercício Feito:

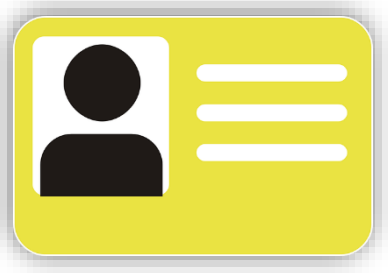
```
function tamanhoNomeCompleto(nome, sobrenome) {

var nomeTodo = nome + " " + sobrenome;
return nomeTodo.length;
}
tamanhoNomeCompleto("Diego","Fonseca");
```

---

8 - Criando cartões.

Para uma conferência importante, os organizadores nos pediram para escrever os cartões de identificação que cada participante terá.



Para isso, temos que colocar seu nome, seu sobrenome e seu título (dr., dra., lic., etc) e montar uma única string.

Escreva a função escreverCartao, que recebe como parâmetros um título, um nome e um sobrenome e retorna uma única string como resultado. Por exemplo:

```
escreverCartao("Dra.", "Ana", "Pérez")
"Dr. Ana Pérez"
```

Exercício Feito:

```
function escreverCartao(titulo, nome, sobrenome){
var credencial = titulo + " " + nome + " " + sobrenome;
return credencial;
}
escreverCartao("Dr", "Diego" , "Fonseca");
```

---

9 - Conhecendo funções Math.

Graças aos programadores de JavaScript, temos algumas funções que você pode usar sem declarar (porque eles fizeram isso por nós). Estas são algumas das funções matemáticas que podemos usar:

Math.abs () **retorna o valor absoluto do número que passamos para ele como parâmetro.**

```
Math.abs (4) //retorna 4
Math.abs (0) //retorna 0
Math.abs (-123) //retorna 123
```

Math.round () **arredonda um número para cima até o número inteiro mais próximo** e Math.floor () **arredonda um número para baixo até o número inteiro mais próximo.**

```
Math.round (4.6) //retorna 5
Math.round (4.3) //retorna 4
Math.floor (4.6) //retorna 4
Math.floor (4.3) //retorna 4
```

Math.max () pega dois parâmetros e retorna o maior número, enquanto Math.min () pega dois parâmetros e retorna o menor.

```
Math.max (4, 7) //retorna 7
Math.min (4, 7) //retorna 4
```

Experimente estas funções no terminal para exercitar o seu entendimento. Lembre-se que o objetivo desse exercício é apenas para treinar as funções.

Exercício Feito:

```
Math.abs(4)
Math.round(4.6)
Math.floor(4.3)
Math.max(4, 7)
Math.min(4, 7)
```

---

10 - Mais uma função.

Além das funções vistas no exercício anterior, existe uma função já definida em JavaScript que é muito importante: Math.random (). Essa função gera um número **aleatório** decimal entre 0 e 1, e é a base para muitos cálculos usados na programação.

```
  Math.random()
0.056

 Math.random()
0.178
```

Escreva uma função gerarProbabilidade(), que não recebe parâmetros e retorna a porcentagem de probabilidade de chuva, calculada aleatoriamente usando Math.random ().

Exercício Feito:

```
function gerarProbabilidade(){

return Math.random()*100;
}
```

11 - CONDICIONAIS

Operadores Lógicos

Os operadores lógicos no JavaScript são os mesmos operadores aritméticos. Através deles podemos combinar valores booleanos(*true or false*), sendo utilizados rotineiramente no dia a dia.

Abaixo segue alguns exemplos de operadores lógicos e suas respectivas formas no JavaScript:

! → negação

== → equivalente

&& → e

|| → ou

> → maior

>= → maior ou igual

< → menor

<= → menor ou igual

% → resto da divisão

!= → diferente de...

Em linguagens de programação é muito comum utilizarmos esses operadores, assim como o *if* e *else*. Essas condicionais nos permitem elaborarmos u código que segue mais de um caminho, por exemplo se um número for maior q 0, ou seja, positivo, imprima na tela "Positivo!", e caso for menor que 0, ou seja, negativo, imprima "Negativo!".

Abaixo segue o exemplo citado em JavaScript:

```
function positivoNegativo(num) {
if (num > 0) {
console.log("Positivo !");
} else {
console.log("Negativo !");
}
}
PositivoNegativo(10)
PositivoNegativo(-5)
```

Como podemos observar no nosso programa, após criarmos a nossa função positivoNegativo, escrevemos *if* (num > 0) {, ou seja, se (numero maior que zero). Note que após declarar a condição da função abrimos as chaves novamente, para então dizer o que deve ocorrer caso a condição seja atendida.

Com isso, podemos estabelecer agora em nossos códigos se um número é maior que outro, por exemplo. Segue abaixo o código desse exemplo:

```
function eMaior(num1, num2) {
if (num1 > num2) {
return num1;
} else {
return num2;
}
}

eMaior(2,4);
eMaior(9,10);
EMaior(2,1);
```

Como vimos no exemplo anterior, ao "chamar" a função eMaior, utilizamos parâmetros(primeiro exemplo 2 e 4) diferentes e compreendemos melhor como as condicionais funcionam(*if* e *else*).

Podemos tirar mais vantagem ainda dos operadores lógicos, condicionais e as funções juntos. É possível retornar um valor na função, como vimos no tópico Funções, com expressões lógicas(operadores). Segue o enunciado de um exercício resolvido a seguir para demonstrar:

Defina a função filosofoHipster que recebe como parâmetro: a profissão de uma pessoa (string), nacionalidade (string) e o número de quilômetros que ele anda por dia (número). Com esses parâmetros avalie se essa pessoa é ou não (true / false), um filósofo Hipster. Tenha em mente que um filósofo Hipster é um Músico, nascido no Brasil e que gosta de andar mais de 2 quilômetros por dia.

Exemplo:

filosofoHipster ('Músico', 'Brasil', 5) // verdadeiro

filosofoHipster ('Jogador de futebol', 'Alemanha', 12) // false

filosofoHipster ('Músico', 'Argentina', 1) // false

Agora veremos o código feito para analisarmos como funciona:

```
function filosofoHipster(profissao, nacionalidade, km) {
return (profissao == 'Músico' && nacionalidade == "Brasil" && km > 2);
}
console.log(filosofoHipster('Músico', 'Brasil', 5));
console.log(filosofoHipster('Jogador de futebol', 'Alemanha', 12));
console.log(filosofoHipster('Músico', 'Argentina', 1));
```

Se repararmos bem, o *return* está fazendo a mesma função de um *if*, como dito anteriormente. Ao colocar uma expressão de operadores lógicos se relacionando entre si com o comando *return*, estamos pedindo simplesmente q a função retorne *true*, caso as condições sejam atendidas ou *false*, caso o contrário. Neste exemplo, a função retornará *true* apenas se profissão for equivalente(==) a 'Músico', nacionalidade equivalente(==) a 'Brasil' e km maior que 2. É muito interessante esse tipo de função pois ela reduz o tamanho do código e executa a mesma função que se criada de outra maneira, porém uma versão "compacta". O legal também é que podemos ver a tabela verdade de Matemática sendo aplicada nessas expressões, como por exemplo, && sendo equivalente a conjunção(^).

Agora que nossas capacidades e compreensão referente à linguagem cresceu, podemos tentar realizar esse exercício:

Escreva a função podeSeAposentar que recebe por parâmetro a idade, o sexo e os anos de contribuição previdenciária que uma pessoa tem, exemplo:

podeSeAposentar**(62, "F", 34)**

True

Tenha em mente que a idade mínima para se aposentar para mulheres é 60 anos, enquanto que para homens é 65. Em ambos os casos, você deve ter pelo menos 30 anos de contribuição.

Segue o código com a resolução:

```
function podeSeAposentar(idade, sexo, contribuicao) {

return (idade >= 60 && sexo == "F" && contribuicao >= 30) || (idade >= 65 &&  sexo == "M" && contribuicao >= 30);

}

console.log(podeSeAposentar(63, 'F', 25));

console.log(podeSeAposentar(58, 'F', 35));

console.log(podeSeAposentar(63, 'M', 35));

console.log(podeSeAposentar(68, 'M', 26));

console.log(podeSeAposentar(58, 'M', 35));
```

Nesse código, a função foi dividida em duas partes: masculino, ou seja, "M", e feminino, ou seja, "F". Vamos destrinchar o nosso código e aumentar nossa percepção.

```
return (idade >= 60 && sexo == "F" && contribuicao >= 30) ||
```

Aqui foi ampliado a linha do comando *return*. Com isso, se pode interpretar esse segmento de código de JavaScript em linguagem comum: "Retorne verdadeiro(*true)* se...". Nessa etapa, repare a condicional *if* embutida já no comando *return* com atenção. Seguindo o código: "...(idade >= 60 && sexo == "F" && contruibuicao >= 30) || ...". Nessa parte podemos traduzir para o seguinte: idade **maior ou igual a** 60 **E** sexo equivalente a "F" E contribuição maior ou igual a 30. No final, temos o símbolo "||", o qual é referente a "ou", ou seja, disjunção. Ao analisar qualquer código de perto, temos a oportunidade de compreender como ele funciona e facilitando a compreensão, agregando em nosso conhecimento e aperfeiçoando os nossos futuros programas.

Assim, fica claro nesse tópico da apostila o potencial de nossas condicionais junto com os outros conhecimentos passados, principalmente se relacionarmos com a parte de **funções**.

12 - CICLOS

Ciclos se referem a laços de repetições, ou seja, um *loop* o qual determinamos quando inicia e quando termina. São essenciais para não termos muitas linhas de código repetidos, podendo criar uma função para essa determinada repetição.

Para realizarmos os ciclos, usamos o comando *for* , o qual possui 3 parâmetros: início, fim, e por último, a etapa. O início se dá ao valor inicial o qual queremos começar o nosso laço, normalmente sendo criado uma variável a qual recebe o valor 0. O fim se dá ao determinarmos quando o ciclo deve parar, por exemplo, ou seja, quantas vezes queremos repetir determinada ação. Finalmente, a etapa, a qual se caracteriza o ritmo dessa repetição. Como praticamente tudo em programação, vejamos o seguinte exemplo para ajudar a visualização:


```
Function imprimirAzul14() {
        for(var i = 0; i < 4; i ++) {
                (“Azul”)
            }
}
imprimirAzul14();
```

Repare que podemos, dentro da linha a qual utilizamos o *for* criar uma variável a qual existirá somente dentro da função, sendo normalmente chamada de i. Atribuímos o valor de **i** sendo equivalente a 0. Em seguida, foi definido que enquanto i for menor que 4 o ciclo irá se repetir. Vale ressaltar que caso **i** nunca ultrapasse esse valor estabelecido, o laço irá continuar infinitamente, a não ser que ele seja parado de forma forçada. No final, colocamos i++, o qual significa que para cada vez que o laço ocorrer, o valor de **i** será acrescentado uma unidade, ou seja, a cada repetição i = i + 1.

A seguir, fica registrado um exercício o qual irá colaborar como o *for* funciona dentro de nosso código:

Execute uma função chamada passandoPor que imprime no console "aqui eu tenho o valor de x", onde x será o valor de i em cada iteração para cada valor de 0 a 3.

A seguir, o código para conferir a resposta:

```
Function imprimirAzul14() {
        for(var i = 0; i < 4; i ++) {
                console.log(‘aqui eu tenho o valor de ${i}.’)
            }
}
imprimirAzul14();
```

Aqui, ao "chamar" a função, vemos que o valor de i sendo alterado, variando de 0 até 3. Inicia com 0 pois atribuímos esse valor ao criar essa variável dentro do parênteses do comando *for* e ele segue até 3 pois o próximo laço o valor de i seria equivalente a 4, ou seja, iria quebrar o parâmetro definido como final, o segmento `i < 4 .` Note que foi utilizado de uma maneira diferente dentro do `console.log()` a impressão do valor de i, porém podemos utilizar também outras formas. Teste você mesmo o seguinte comando na linha referente ao `console.log():`

```
console.log("aqui eu tenho o valor de", i)
```

Voltando ao assunto de ciclos, podemos somar um valor por n vezes utilizando os ciclos. Tente realizar o seguinte exercício:

Sabendo disso, escreva a função somar5MoedasDe25Centavos, que adiciona o valor de 5 moedas de 0,25 centavos e retorna o resultado.

Resultado esperado: 1.25.

Ou seja, a função realizou 5 vezes a soma de "0.25".

Essa é uma das possibilidades:

```
soma = 0

function somar5MoedasDe25Centavos() {

        for(var coin = 1; coin <= 5; coin ++) {

                soma += 0.25

                }

                Return console.log(soma);

}

somar5MoedasDe25Centavos();
```

Notou algo diferente ? A variável declarada não é mais chamada de i e sim de coin, apenas para lembrar que é possível dar o nome que quiser para a variável, desde que faça sentido para você e quem ler. Foi criado também a variável soma, a qual inicialmente recebeu 0 como valor e a cada repetição no *step*, ou seja, passo(coin++), foi somado o valor de 0.25 com a linha: soma += 25 (soma = soma + 25).

Os ciclos são muito úteis, facilitando algumas possibilidades e deixando nosso código mais harmonioso. Um exemplo simples em que os ciclos facilitam pode ser observado na seguinte tarefa:

```
function somDosPares(x) {
        soma = 0
        for(i = 0; i <=x; i ++) {
            if(i % 2 == 0) {
            soma += i
            }
        }
        return soma;
}
console.log(somaDosPares(0));
console.log(somaDosPares(4));
console.log(somaDosPares(10));
```

No exemplo acima, a função retorna a soma dos números positivos pares, como se pode ver nos resultados diferentes. No primeiro exemplo, seria 0 + 0, no segundo, 0 + 2 + 4 = 6, e no último, 0 + 2 + 4 + 6 + 8 + 10 = 30, facilitando e muito caso fossemos receber um exercício como esse:

Defina a função somadosPares(x) que retorna uma soma total de todos os números que são pares, ou seja, divisíveis por 2 obtendo resto 0. Desta forma: 0 + 2 + 4 + 6 + 8 + 10 + ... x.