Usando funções para otimizar o código

Até agora você entendeu o fluxo de um programa simples que é basicamente definir variáveis, receber valores e executar algo com isso.

A má notícia é que quanto mais complexo o programa vai ficando, mais é difícil de manter e adicionar novas funcionalidades.

Por isso é importante você entender como organizar corretamente o seu código e permitir reutilizá-los.

Imagine o exemplo da nossa calculadora:

```
var x = 10;
var y = 20;

var resultado = x + y;

console.log(resultado);
```

A operação de soma precisa ser melhor "reutilizada", sendo assim podemos criar uma função para tal.

As funções são espaços de código com um "nome" escolhido pelo programador (vide as regras de nome) que podem ser invocadas pelo programa principal. É como uma caixa de ferramentas, no momento certo você abre a caixa e escolhe a ferramenta mais adequada para o serviço.

Se você precisar de um martelo, não necessita se preocupar em fabricar o cabo, basta martelar seu dedo! Com funções é o mesmo conceito, elas ficam armazenadas na "caixa de ferramentas" prontas para o uso do programa principal.

Como criar uma função?

No JavaScript você precisa definir uma função utilizando a palavra-chave `function`, veja:

```
var soma = function(x, y) {
    var resultado = x + y;
    return resultado;
}

console.log(soma(10, 20));
```

As funções podem receber uma série de parâmetros de entrada (no exemplo `x` e `y`) ou então não receber nada (bastando deixar os `()` vazios).

O corpo da função deve conter os códigos que você deseja executar, no exemplo acima a soma. E também caso seja aplicável um retorno de valor com o `return` (como é necessário na função de exemplo).

Veja uma função "útil" utilizada para fins diversos:

```
var plim = function() {
    console.log("Plim plim...");
}

plim();
```

Como você pode ver no exemplo acima a função [in]útil não recebe parâmetros e também não retorna nada. Apenas executa comandos.

Parâmetros e escopos

Os parâmetros de uma função se comportam como variáveis locais. Isso quer dizer que você utiliza apenas dentro da função declarada.

Essa característica de localidade de uma função aplica somente a parâmetros e variáveis que forem declaradas dentro da função usando o `var`. Variáveis declaradas fora do contexto da função (`{}`) são chamadas de **variáveis globais**.

As variáveis globais podem ser acessadas e alteradas no contexto da função desde que **não tenham sido redeclaradas dentro da função**.

Veja:

```
var x = "variável global";

var f1 = function() {
    x = "variável da função 1"; // altera a variável global a primeira vez
}
f1();

var f2 = function() {
    x = "variável da função 2"; // altera a variável global a primeira vez
}
f2();

console.log(x);
// → variável da função 2
```

Agora iremos redeclarar a mesma variável dentro de cada função:

```
var x = "variável global";

var f1 = function() {
    var x = "variável da função 1";
}
f1();
```

```
var f2 = function() {
    var x = "variável da função 2";
}
f2();

console.log(x);
// → variável global
```

O comportamento de localidade ajuda a prevenir interferências de outras funções. Se todas as variáveis fossem compartilhadas com o programa principal seria muito difícil garantir o correto funcionamento.

O excesso de uso de variáveis globais torna o seu programa complexo e instável, já que dado uma hora você pode descuidar-se e fazer besteira. Por isso o conceito de "micro-programas" é importante.

Torne cada função especialista naquilo que faz, fazendo algo bem pequeno e de forma correta. Assim a cadeia de funções criará um programa robusto.

Aninhamento de funções

Como comentado tudo que é declarado dentro de uma função pertence a um "micro-programa". Esse conceito se estende às próprias funções, veja:

```
var paisagem = function() {
    var res = "";

    var vale = function(quant) {
        for (var c = 0; c < quant; c++) {
            res += "_";
        }
    };

    var montanha = function(quant) {
        res += "/";

        for (var c = 0; c < quant; c++) {
            res += "'";
        }

        res += "\\";
    };

    vale(3);
    montanha(4);
    vale(6);
    montanha(1);
    vale(1);

    return res;
};
```

```
console.log(paisagem());
// → ___/''''\_____/'\_
```

As funções `vale` e `montanha` podem acessar a variável `res` porque elas estão dentro do mesmo escopo da função `paisagem`. Entretanto elas não podem enxergar as variáveis `c` e `res` uma da outra (somente dela mesmo), porque elas estão definidas em escopos diferentes.

E o programa principal (ambiente externo) não consegue enxergar a variável `res`, `vale` e `montanha` porque estão sob o escopo da função `paisagem`.

Por isso é importante você praticar e entender cada conceito para não se dar mal quando for criar algo do tipo.

Passando uma função no parâmetro de outra

O JavaScript permite atribuir funções para uma variável, como já vimos até agora. E isso permite que você passe esta variável como parâmetro para outra função, veja:

```
var f1 = function() {
    console.log("Executando f1");
}

var f2 = function(f) {
    console.log("Executando f2");
    f();
}
f2(f1);
// → Executando f2
// → Executando f1
```

Eu sei que isso não faz muito sentido agora... Mas garanto que nas próximas aulas fará. Criaremos uma verdadeira maravilha com isso 😃

Notação por declaração

Até o momento declaramos funções utilizando o `var`. Ou seja, atribuindo uma função para uma variável e depois utilizando-a.

No JavaScript é possível declarar com `var` e também diretamente:

```
function ola() {
    console.log("Hello...");
}
ola();
```

Como você pode perceber a declaração da função acontece primeiro com a palavra-chave `function` seguido do nome.

A forma de declarar não altera o comportamento de função. Se você for utilizar a função normalmente (sem repassar como parâmetro para outra função) ficará ao seu critério o formato.

Nomenclatura de funções

Assim como variáveis as funções obedecem ao mesmo critério de nomenclatura: sem espaços, acentos, caracteres especiais, sempre devem iniciar com uma letra, devem ser `camelCase` e de preferência com verbos no infinitivo.

Sendo assim, uma função que desempenhar uma função de cálculo de soma deveria se chamar: `calcularSoma(a, b)`.

Não tem problema nenhum deixar de seguir certas regrinhas. Contudo é a mesma coisa que ir mal vestido para uma festa, você vai passar vergonha 😉

Desafio

Criar uma função chamada somar para a nossa calculadora e utilizar esta função.

Resposta

```
var soma = function(x, y) {
    return x + y;
}

var v1 = Number(prompt("Digite o primeiro valor"));
var v2 = Number(prompt("Digite o segundo valor"));

alert("Resultado: " + soma(v1, v2));
```