7. Estrutura de Função

Neste capítulo você aprenderá a:

- Reaproveitar o código.
- Trabalhar com estrutura de funções no *javascript*.

7.1 Reaproveitamento de Fluxo

Nossa, olha só onde você chegou! No último capítulo. Eu estou feliz por você ter chegado até aqui, esse capítulo fecha este livro com "chave de ouro"! Vamos aprender agora como poderemos reaproveitar um código, transformando-o em uma função. Uma função na programação é como uma função na matemática, lembra? Talvez não, né..., mas vamos relembrar um pouco para que consigamos fazer uma conexão aqui.

Suponhamos que temos a seguinte função: $f(x,y) = 8x - y + 20$. Se acalme, não é para resolver! Essa função podemos dizer que é constituída de duas partes: os parâmetros, x e y, e a resposta, $8x - y + 20$. Se eu lhe informar o valor de x e y você saberia a resposta, certo? Vamos tentar, se $x = 3$ e $y = 10$ qual o valor da resposta? Resposta correta: $8 \times 3 - 10 + 20 = 34$. De forma similar faremos na programação. Definiremos os parâmetros que serão incorporados no escopo da função e, se quisermos, definiremos um retorno que poderá ser utilizado em outra parte do programa.

Vamos partir para uma ideia de utilização mais prática do reaproveitamento do fluxo. Digamos que queiramos calcular a seguinte função:

$$C(n,p) = \frac{n!}{(n-p)!p!}. \tag{7.1}$$

Onde, a exclamação indica o sinal de fatorial; exemplo $4! = 4 \times 3 \times 2 \times 1 = 24$. Se tivéssemos uma estrutura que calcula o fatorial nossa vida seria bem mais fácil. Só teríamos o trabalho de *invocá-la*. Daqui a pouco te explico como vai ser isso.

7.2 Função

Na determinação de uma função precisamos de duas coisas, os parâmetros e o retorno. Mas, além disso, podemos colocar o que quisermos em nossas funções. Usualmente, o que queremos retornar tem a ver com o parâmetro. Na Equação 7.1, vemos isso nitidamente, o retorno depende dos valores de n e p. Porém pode ocorrer de uma função não possuir retorno, neste caso a chamamos de procedimento. Quando uma função é chamada está associado eu gosto de utilizar o termo invocação, ao invocá-la o programa irá aguardar o retorno da função.

Podemos também definir funções sem parâmetros, elas funcionariam como funções constantes, onde não importa os valores que passarmos o retorno sempre será igual. Por exemplo, $f(x) = 5$; não importa o valor de x o retorno será sempre 5, nesses casos não precisamos colocar parâmetros, concorda? Bem, é isso que quero que você mentalize, por hora, a respeito de funções:

1. Funções podem possuir parâmetros ou não;
2. Funções que não possuem retorno são chamadas de procedimento;
3. Quando uma função é invocada o programa aguarda o seu retorno para continuar.

7.3 Função no Javascript

No *javascript* primeiramente declaramos uma função, e só depois a invocamos, podemos declará-la com a seguinte estrutura:

```
1 function <identificador> (<parâmetro(s)>) {
2     <instruções>
3     return <dado>;
4 }
```

Primeiramente você precisará definir um `identificador` para função, a definição do `identificador` deve seguir as mesmas regras aplicadas na definição de variáveis que falamos na Seção 3.4. Logo após, na `function`, você pode colocar quantos parâmetros quiser, separados por vírgula, (ou não colocar nenhum), e entre os parênteses.

> ⚠ Na ausência de parâmetros você deve manter os parênteses, caso contrário será acusado um erro. Os parâmetros, quando existirem, devem seguir as mesmas regras da definição de variáveis.

É como se fossem as variáveis que ficam na parte de declaração do `for`, porém, aqui você não precisará colocar o `var`, mas, se quiser, poderá inicializá-las. Em instruções, você pode colocar os comandos que quiser. E, opcionalmente, você pode colocar um `return`. Após o `return` você pode especificar o dado que você quer retornar. Pode ser, por exemplo, um `number`, um `boolean` ou uma `string`. Esse comando tem um poder similar ao do `break` em uma repetição. Tudo que estiver abaixo desse comando não será processado na função. Vamos a um exemplo prático...

No Código 7.1, temos a declaração da função `fatorial` para o cálculo do fatorial de um número `n`.

Código 7.1 Função Fatorial[1].

```
1 function fatorial(n){
2   var fat = 1;
3   for(var i=n;i>1;i--){
4     fat *= i;
```

[1] <https://replit.com/@DaniloBorges/FuncaoFatorial>

```
5     }
6     return fat;
7   }
8
9   var n = parseInt(prompt("Informe o valor de n"));
10
11  console.log(`${n}! = ${fatorial(n)}`);
```

■

A função `fatorial` calcula o fatorial do número usando o comando de repetição e retorna esse valor (linha 6). O parâmetro necessário é o número, `n`, que se deseja calcular o fatorial. A invocação da função é feita na linha 11. Observe que a declaração de uma função não implicará em sua execução. A mesma só será executada quando invocada.

Você só poderá invocar uma função após tê-la declarado. E poderá invocar quantas vezes quiser.

Vamos agora criar uma função que calcula a Equação 7.1, no Código 7.2.

■ **Código 7.2** Função Combinação[2].

```
1   function fatorial(n){
2     var fat = 1;
3     for(var i=n;i>1;i--){
4       fat *= i;
5     }
6     return fat;
7   }
8
9   function combinacao(n,p){
10    return fatorial(n)/(fatorial(n-p)*fatorial(p));
11  }
12
13  var n = parseInt(prompt("Informe o valor de n"));
14  var p = parseInt(prompt("Informe o valor de p"));
15
16  console.log(`C(${n},${p}) = ${combinacao(n,p)}`);
```

■

Nesse código vemos o tal do reaproveitamento de forma mais evidente. Estamos reaproveitando a função `fatorial` (linha 1) na função `combinacao` (linha 10). Obtemos do usuário os valores de `n` e `p` e devolvemos o valor calculado (linha 16). O processamento disso é simples. Na linha 11, quando é realizada a primeira invocação ao fatorial, `fatorial(n)`, o programa executa a função e espera o retorno, e depois processa as demais invocações, `fatorial(n-p)` e `fatorial(p)`, após isso a operação de multiplicação e divisão é realizada com os valores retornados.

Oriento que você trabalhe aos poucos com a ideia de funções, pois, no *javascript* um dos tipos de dados que falei é `function` e com ela podemos fazer muitas manipulações, mas não falarei disso aqui. Fica para uma próxima oportunidade.

Com isso eu encerro esse o assunto deste livro, espero que tenha gostado do conteúdo e não fique por aqui, seja curioso. O primeiro passo foi dado, continue a sua jornada na programação!

[2] <https://replit.com/@DaniloBorges/FuncaoCombinacao>

7.4 Exercícios

Exercício 7.1 Faça uma função que recebe um valor e verifica se o valor é positivo ou negativo. A função deve retornar um valor booleano (`true` para positivo e `false` para negativo). ■

Exercício 7.2 Faça uma função que recebe um valor inteiro e verifica se o valor é par ou ímpar. A função deve retornar um valor booleano (`true` para par e `false` para ímpar). ■

Exercício 7.3 Escreva uma função que recebe por parâmetro um valor inteiro e positivo n e retorna o valor de S. Para $n \geq 1$:

$$S = \frac{2}{4} + \frac{5}{5} + \frac{10}{6} + \frac{17}{7} + \frac{26}{8} + \cdots + \frac{(n^2+1)}{(n+3)}$$

. ■

Exercício 7.4 Faça um programa que calcula o valor de função de primeiro grau $f(x)$:

$$f(x) = ax + b$$

o usuário irá informar os valores de x, a e b. ■

Exercício 7.5 Faça um programa que calcule o montante final de uma aplicação com juros simples baseado na seguinte função:

$$M(C,i,t) = C \times (1 + i \times t)$$

onde, C é o capital inicial, i é a taxa de juros, valor entre 0 e 1, e t é a quantidade de meses. Use uma função. Os parâmetros, C, i e t serão informados pelo usuário. ■

Exercício 7.6 Faça um programa que calcule o montante final de uma aplicação com juros compostos baseado na seguinte função:

$$M(C,i,t) = C \times (1 + i)^t.$$

Onde, C é o capital inicial, i é a taxa de juros, valor entre 0 e 1, e t é a quantidade de meses. Use uma função. Os parâmetros, C, i e t serão informados pelo usuário. ■