Paradigma funcional

Sumário

O paradigma funcional

Programação funcional é um paradigma de programação onde as aplicações são construídas através da criação e composição de funções. É uma abordagem declarativa onde as definições das funções são expressões que retornam valores, ao invés de uma sequência de instruções imperativas que alteram o estado da aplicação.

Linguagens que implementam o *paradigma funcional* possuem Funções de primeira classe.

Funções de primeira classe

Uma linguagem possui *funções de primeira classe* se ela trata as funções como *cidadãos de primeira classe* (first-class citzens), que, por sua vez, significa que funções são tratadas como as demais entidades da linguagem (tipos, objeto, valores, etc.). Isso quer dizer que é possível atribuir funções à variáveis, passar funções no parâmetro de outras funções e retornar funções como retorno de funções.

Exemplo:

```
//Código em javascript

function dobro(x){ return 2*x;}

var retornoDobro = dobro(4);
console.log('Retorno de dobro(4): ', retornoDobro); // escreve '8' no console

//-----------------------------------------

var minhaFuncao = dobro; //atribuindo a função dobro à variável minhaFuncao

var retornoMinhaFuncao = minhaFuncao(4); //posso chamar minha variável como se
fosse a função dobro;
```

```
console.log('Retorno de minhaFuncao(4): ', retornoMinhaFuncao); // escreve '8' no
console
```

Funções de primeira ordem (High order functions)

Uma função é dita ser uma *função de primeira ordem* se ela recebe (via parâmetro) ou retorna uma outra função.

Exemplo:

```
//Código em javascript

function dizerBomDia(nome){ console.log('Bom dia,', nome);}
function dizerBoaTarde(nome){ console.log('Boa tarde,', nome);}

// Essa função recebe a função cumprimentoFunc como parâmetro. Logo a função falar
é uma função de primeira ordem.
function falar(cumprimentoFunc, nomePessoa, textoASerDito)
{
   cumprimentoFunc(nomePessoa);
   console.log(textoASerDito);
}

var nomeEscolhido = 'Miranda';
var textoPrevisaodoTempo = "Hoje a previsão é fazer 27 graus.";

falar(dizerBomDia,nomeEscolhido,textoPrevisaodoTempo);// escreve 'Bom dia,
Miranda. Hoje a previsão é fazer 27 graus.' no console

falar(dizerBoaTarde,nomeEscolhido,textoPrevisaodoTempo);// escreve 'Boa tarde,
Miranda. Hoje a previsão é fazer 27 graus' no console
```

Funções puras e funções impuras

Funções puras são funções que respeitam as propriedades:

- **Dada a mesma entrada, o valor de saída será sempre o mesmo**: Isso significa que a função não pode depender de nenhum estado mutável da aplicação (data do sistema, configuração do sistema, Math.Random(), etc.)
- **Não causa efeitos colaterais**: Isso significa que a função não acessa e não altera valores globais. Inserção de dados no banco de dados, escrita de log e etc. são exemplos de efeitos colaterais. Funções que que não respeitam as propriedades acima são chamadas de *funções impuras*.

Exemplo:

```
//Código em javascript
```

```
function gerarNomeArquivo_Impura(nomePrincipal){
   var agora = new Date(Date.now());
   var dia = agora.getDate();
   var mes = agora.getMonth() + 1;
   var ano = agora.getFullYear();

   return nomePrincipal + "_" + ano + '.' + mes + '.' + dia;
}

var meuArquivoImpuro = gerarNomeArquivo_Impura('meuArquivo'); // o retorno da
função dependerá da data em que ela for chamada, mesmo que o parâmetro seja o
mesmo. Portanto é uma FUNÇÃO IMPURA;


function gerarNomeArquivo_Pura(nomePrincipal, dataAtual){

   var dia = dataAtual.getDate();
   var mes = dataAtual.getMonth() + 1;
   var ano = dataAtual.getFullYear();

   return nomePrincipal + "_" + ano + '.' + mes + '.' + dia;
}

var meuArquivoPuro = gerarNomeArquivo_Pura('meuArquivo'); // o retorno da função
dependerá APENAS dos parâmetros passados. Portanto é uma FUNÇÃO PURA;

//----------------------------------------

function dizerOi_Impura(){

   console.log('Oi,', fulano); // está acessando o valor de uma variável externa
(global), portanto é uma FUNÇÃO IMPURA.

}
var fulano = 'Sidnei';
dizerOi_Impura(); // escreve 'Oi, Sidnei' no console.

//----------------------------------------

function exibirMensagem_Impura(mensagem){
   // a linha abaixo altera o valor da propriedade texto do objeto mensagem, logo
é uma função impura;
   mensagem.texto = '-----Início da mensagem-----\n' +
      mensagem.texto +
      'autor: ' + mensagem.autor +
      '\n-----Fim da mensagem-----';
   console.log(mensagem.texto);
}

var minhaMensagem = {texto: 'Cuidado com a alteração das variáveis dentro de
funções', autor:'DEV que se importa com funções puras'};
// antes de chamar a função abaixo, minhaMensagem.texto está OK;
exibirMensagem_Impura(minhaMensagem);
```

```
//após a chamada da função acima o valor de minhaMensagem.texto foi alterado
console.log(minhaMensagem.texto);
```

Imutabilidade

Em programação funcional, *imutabilidade* significa que, uma vez definido o valor de uma variável, não é possível mudá-lo.

Pergunta: E como eu faço um laço de repetição que usa uma variável de 'contador' para controlar a repetição senão posso alterar o valor do contador?

Resposta: Você pode usar Recursão! Ou pode usar funções (e não instruções!) que a linguagem fornece (map, forEach, filter, etc.).

Exemplo:

```
//Código em javascript
var minhaVariavel = 123;
console.log('Minha variável:', minhaVariavel); //escreve 'Minha variável: 123' no
console
minhaVariavel = 789; // valor da variável foi alterado! Portanto não é um código
IMUTÁVEL.
console.log('Minha variável depois da alteração:', minhaVariavel);//escreve 'Minha
variável: 789' no console
```

Recursão

Em programação, *recursão* significa que uma função ou método pode, em seu corpo, chamar a si mesmo.

> O conceito de *recursão* é independente do paradigma adotado. Podemos utilizá-lo em programação estruturada, programação funcional, programação orientada a objetos, etc.

Exemplo:

```
// Linguagem: C#

// Função que chama a si mesma em seu corpo. Função recursiva.
function int somarNumerosDeUmAte(int valorMaximo)
{
   if (valorMaximo <=0){ // este é o CASO BASE da função recursiva, que indica
quando a função termina sua recursividade.
      return 0;
   }
   return valorMaximo + (somarNumerosDeUmAte(valorMaximo-1)); // chamando a si
mesma!
}

function int main()
{
```

```
    int resultado = somarNumerosDeUmAte(5);
    Console.WriteLine($"A soma de 1 até 5 é {resultado}");
}
```

> ATENÇÃO: Funções recursivas mal definidas podem causar um problema chamado Stack Overflow, que, como o nome sugere, é o transbordamento da pilha de chamadas das funções. Esse problema é causado quando uma função é chamada muitas vezes de maneira recursiva e faz com que a pilha de chamadas não tenha mais espaço para armazenar outras chamadas.

Vantagens do paradigma funcional

- Menos efeitos colaterais graças ao conceito de Imutabilidade;
- Eficiência na programação concorrente e paralela já que não teremos concorrência de estados da aplicação;
- Facilidade em testar o código, uma vez que funções puras não possuem dependências e nem efeitos colaterais;

Desvantagens do paradigma funcional

- Consumo elevado de memória pois uma nova cópia do objeto é gerada na memória a cada vez que ele precisa ser alterado ou utilizado;
- Pensar de forma funcional não é muito intuitivo;

Exercícios

1. Crie uma aplicação em **javascript** que seja capaz de realizar as 4 operações básicas da matemática **utilizando funções de primeira ordem**. Fluxo da aplicação:
    - A aplicação deve **somar** os números `3.854` com `6.523` e apresentar o resultado (`10.377`) no console;
    - A aplicação deve **subtrair** o número `2856` do número `438` e apresentar o resultado (`2.418`) no console;
    - A aplicação deve **multiplicar** os números `39.564` e `2.756` e apresentar o resultado (`109.038.384`) no console;
    - A aplicação deve **dividir** o número `1.209.127` pelo número `593` e apresentar o resultado (`2.039`) no console;
2. Crie uma aplicação em **javascript** que calcule o enésimo (a posição n) termo da Sequência Fibonacci **sem utilizar laços de repetição** (loops); Fluxo da aplicação:
    - A aplicação deve calcular o termo `5` (n = 5) da sequência e apresentar o resultado (`5`) no console;
    - A aplicação deve calcular o termo `10` (n = 10) da sequência e apresentar o resultado (`55`) no console;
    - A aplicação deve calcular o termo `20` (n = 20) da sequência e apresentar o resultado (`6765`) no console;
    - A aplicação deve calcular o termo `50` (n = 50) da sequência e apresentar o resultado (`12.586.269.025`) no console;

> Dica: Para os exercícios 3 a 5 abaixo, existe uma função nativa do javascript que 'fatia' arrays e que te ajudará muito na resolução. Ela se chama `slice` (procure na internet para ver como ela é usada).

3. Crie uma função em **javascript** que retorne um array onde cada elemento é o quadrado (x elevado ao quadrado) de cada elemento do array abaixo **sem utilizar laços de repetição** e **sem utilizar funções de primeira ordem**.
    - array: `[181, 82, 192, 104, 65, 75, 57, 156, 92, 69, 147, 142, 184, 134, 155]`
    - O resultado deve ser o array `[32761, 6724, 36864, 10816, 4225, 5625, 3249, 24336, 8464, 4761, 21609, 20164, 33856, 17956, 24025]`
4. Crie uma função em **javascript** que retorne um array contendo apenas números ímpares a partir do array abaixo **sem utilizar laços de repetição** e **sem utilizar funções de primeira ordem**.
    - array: `[181, 82, 192, 104, 65, 75, 57, 156, 92, 69, 147, 142, 184, 134, 155]`
    - O resultado deve ser o array `[181, 65, 75, 57, 69, 147, 155]`
5. Crie uma função em **javascript** que retorne a soma de todos os elementos do array abaixo **sem utilizar laços de repetição** e **sem utilizar funções de primeira ordem**.
    - array: `[181, 82, 192, 104, 65, 75, 57, 156, 92, 69, 147, 142, 184, 134, 155]`
    - O resultado da soma deve ser **1835**
6. O mesmo do exercício 3, mas agora utilizando funções de primeira ordem fornecidas pela linguagem.
7. O mesmo do exercício 4, mas agora utilizando funções de primeira ordem fornecidas pela linguagem.
8. O mesmo do exercício 5, mas agora utilizando funções de primeira ordem fornecidas pela linguagem.

Referências

Artigos

- Programação funcional para iniciantes
- Wikipedia: Functional Programming (EN)
- Programação Funcional: O que é e qual a importância?
- Programação funcional: O que é?
- O que é programação funcional e o que isso tem a ver com o Nubank?
- Wikipedia: Sequência Fibonacci

Vídeos

- CODECASTS: Talk - Programação Funcional Direto ao Ponto - Vinícius Reis
- Programação Funcional - Alura Live #15
- Talk #57 - Começando com Programação Funcional
- Programação Funcional 101