Pattern Matching

RODRIGO LYRA

Controle de fluxo

- Em linguagens imperativas é muito comum o uso de controle de fluxo divididos principalmente em duas categorias: desvio condicional (com if/else e switch/case) e laços de repetição (com for, foreah e do/while). Eles são padrões mais imperativos do que funcionais, em linguagens funcionais eles são evitados.

Começando

- Vamos fazer um pequeno desvio condicional, façam uma função que receba um número que r

- O código que estamos fazendo com if/else não é errado, mas em programação funcional costumamos trabalhar com o “pattern matching”, algo como correspondência de padrões.

Pattern Matching

- Essa é a estrutura do Pattern Matching

- match <valor> with
- | padrao1 -> expressao1
- | padrao2 -> expressao2
- ...

Exemplo

```
let verificaPar numero =
    match numero % 2 = 0 with
    | true -> "Par"
    | false -> "Ímpar"
```

Vamos aumentar um pouco

- Agora o problema é um pouco diferente, além do que o problema anterior fazia, caso seja 0, o teste deve retornar “Zero”.

```
let verificaPar numero =
    if numero % 2 = 0 then
        if numero = 0 then
            "Zero"
        else
            "Par"
    else
        "Ímpar"
```

- Vemos que o código começa a apresentar uma estrutura de pirâmide, que começa a deixar o código mais difícil de entender.

Pattern Matching

- Vamos mudar um pouco agora, normalmente o pattern matching recebe o valor e as expressões estão em cada opção de matching. Percebam a expressão “when” quando testa o número e o “_” como coringa para qualquer valor.

```
let verificaMM numero =
    match numero with
    | 0 -> "Zero"
    | numero when numero % 2 = 0 -> "Par"
    | _ -> "Ímpar"
```

Pattern Matching para cálculo de fatorial

- Agora vamos fazer um algoritmo para o cálculo de fatorial, lembrando que o fatorial de um número é igual a multiplicação de todos os número dele até 1.
  - 5! = 5 * 4 * 3 * 2 * 1
  - 3! = 3 * 2 * 1
  - 7! = 7 * 6 * 5 * 4 * 3 * 2 * 1

  Não temos variáveis, então qualquer somatório ou agrupamento de um número variável de valores deve ser feito por recursividade.

  Funções recursivas devem utilizar a palavra reservada **rec** após **let**.

Pattern Matching em listas

Outro uso interessante para o pattern matching é a navegação em listas, como veremos novamente dentro de programação lógica, em programação funcional podemos separar a lista em duas partes: **head** e **tail**. Onde separa o primeiro elemento do cabeçalho do resto da lista.

Pattern Matching em listas

Para algoritmos aplicados em lista podemos utilizar o operador de :: para separar esses dois elementos:

```
let rec sum list =
    match list with
    | head :: tail -> head + sum tail
    | [] -> 0
```

Exercício

- Faça uma função que recebe um número e retorne se ele é primo ou não.
- Faça em F#, utilizando recursividade e Pattern Matching, um programa que retorne o N-ésimo número da sequência de Fibonacci, lembrando que um termo da sequência é sempre a soma dos dois termos anteriores:
  - 1, 1, 2, 3, 5, 8, 13, 21, 34....
- Faça uma função que recebe uma lista e retorna a soma dos valores pares.