

# Programação Funcional

## Aula 4

**Raul Ikeda**  
**2025-1**

Insper



[Insper.edu.br](https://insper.edu.br)

# Esta Aula

---

- Pattern Matching
- Data Structures

# Retomando

---

Seja um exemplo simples de implementação do Fatorial:

```
def fat(n):  
    result = 1  
    for i in range(n):  
        result *= i + 1  
    return result
```

E com recursão:

```
def fat(n):  
    if n <= 1:  
        return n  
    return n * fat(n-1)
```

# Em OCaml

---

Agora em OCaml

```
let rec fat n =  
  if n <= 1 then 1  
  else n * fat (n - 1)
```

Alternativa:

```
let rec fat n =  
  match n with  
  | 0 -> 1  
  | 1 -> 1  
  | n -> n * fat (n-1)
```

E se n for negativo?

## Correção

---

```
let rec fat = function
| n when n <= 1 -> 1
| n -> n * fat (n - 1)
```

ou

```
let rec fat = function
| n when n < 0 -> failwith "n must be positive"
| n when n <= 1 -> 1
| n -> n * fat (n - 1)
```

Qual a grande diferença das duas?

# Exceptions

---

failwith é um wrapper para o raise

```
failwith "n must be positive"  
raise (Invalid_argument "n must be positive")
```

Para capturar o erro:

```
try  
  (* Algo que gera Exception *)  
with  
| Exception1 -> (* Handle type Exception1 *)  
| Exception2 -> (* Handle type Exception2 *)  
| _ -> (* Handle any other exception *)
```

# Pattern Matching

---

```
let int_to_bool i =  
  match i with  
  | 0 -> false  
  | _ -> true (* Caso contrário *)
```

Equivalente em Python:

```
def int_to_bool(i):  
    match i: # Após o Python 3.10  
        case 0:  
            return False  
        case _:  
            return True  
    # Na prática: return i!=0
```

1. Faça uma função chamada *describe\_number* que retorna o nome em string de um número até 5. Retorne "outro" para números maiores que 6.

## Pattern Matching - Mais interessante

---

```
type day = Sun | Mon | Tue | Wed | Thu | Fri | Sat

let int_to_day (i : int) : day =
  match i mod 7 with
  | 0 -> Sun
  | 1 -> Mon
  | 2 -> Tue
  | 3 -> Wed
  | 4 -> Thu
  | 5 -> Fri
  | _ -> Sat
```

Nesse exemplo Sun,...,Sat são **constructors**. Isso também é conhecido como **enumeration** (enum)

Fica ainda mais interessante com uma estrutura de dados



# Listas

---

```
let rec sum_list lst =  
  match lst with  
  | [] -> 0 (* Lista vazia *)  
  | h :: t -> h + sum_list t;; (* Desempacota head do tail *)  
  
let my_list = [1; 2; 3; 4; 5];;  
Printf.printf "Sum: %d\n" (sum_list my_list);;
```

Equivalente em Python:

```
def sum_list(lst):  
    return lst[0] + sum_list(lst[1:])  
  
my_list = [1, 2, 3, 4, 5]  
print("Sum:", sum_list(my_list))  
  
# Ou funcional:  
from functools import reduce  
print("Sum:", reduce(lambda x, y: x + y, my_list, 0))
```

# Listas

---

Listas em OCaml é imutável!

```
let my_list = [1; 2; 3] ;;  
my_list @ [4] ;;  
my_list (* Qual a saída? *)
```

Dica: para grandes listas, é mais performático adicionar o item a esquerda na lista reversa:

```
let append lst x = List.rev (x :: List.rev lst)
```

## Listas - Acessando um elemento

---

```
let x = List.nth [10; 20; 30; 40; 50] 2 (* x = 30 *)

(* por baixo do capô: *)

let rec get_element lst idx =
  match lst with
  | [] -> failwith "Index out of bounds" (* Empty list case *)
  | h :: t -> if idx = 0 then h else get_element t (idx - 1)
```

Qual a complexidade do algoritmo?

1. Faça uma função que retorna a quantidade de elementos em uma lista. Não usar *built-in* function `List.length`.
2. Faça uma função que retorna o maior elemento de uma lista.
3. Faça uma função que retorna o segundo maior elemento de uma lista.

# Listas

---

Como apagar?

```
let delete lst x =  
  List.filter (fun y -> y <> x) lst ;;  
  
let my_list = [1; 2; 3; 4; 2] ;;  
let new_list = delete my_list 2 (* Result: [1; 3; 4] *)
```

"Alteração"

```
let rec update lst old_val new_val =  
  match lst with  
  | [] -> []  
  | h :: t -> if h = old_val then new_val :: t  
               else h :: update t old_val new_val  
  
update [1; 2; 3; 2; 4] 2 9 (* Result: [1; 9; 3; 2; 4] *)
```

# Tuplas

---

```
let person = ("Alice", 30);;  
let (name, age) = person;; (* unpack *)  
Printf.printf "%s is %d years old\n" name age;;
```

Funciona de forma análoga ao Python:

```
person = ("Alice", 30) # Também é imutável  
name, age = person  
print(f"{name} is {age} years old")
```

1. Faça uma função que recebe uma tupla ponto (x,y) e retorna um texto indicando se x e y são iguais, x é maior ou y é maior.

## Record

---

```
type person = { name : string; age : int; country : string ; };;  
  
let alice = { name = "Alice"; age = 9; country = "Japan" };;  
let raul = { age = 40 ; name = "Raul"; country = "Brazil" };;  
  
Printf.printf "%s is %d years old from %s\n" alice.name alice.age alice.country;;
```

Pontos:

- Preciso realmente definir o tipo person?
- A ordem dos itens não importa
- Qual é a alternativa mais próxima do custom type em Python?

# Em Python

---

```
class Person:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

alice = Person(name="Alice", age=30)
print(f"{alice.name} is {alice.age} years old")
```

## Record - Pattern Matching

---

```
let describe_person person =  
  match person with  
  | { name = "Raul"; country = "Brazil"; _ } -> "Raul from Brazil"  
  | { country = "Brazil"; _ } -> "Someone from Brazil"  
  | _ -> "Someone from Earth";;
```

1. Faça uma função que entra um record no formato do exemplo e retorna se a pessoa é maior de idade ou não
2. Altere o type Person para incluir o e-mail. Faça uma função para verificar se um e-mail existe ou não no record.



## Exercícios - Resumo

---

1. Faça uma função chamada *describe\_number* que retorna o nome em string de um número até 5. Retorne "outro" para números maiores que 6.
2. Faça uma função que recebe uma tupla ponto (x,y) e retorna um texto indicando se x e y são iguais, x é maior ou y é maior.
3. Faça uma função que retorna a quantidade de elementos em uma lista. Não usar *built-in* function `List.length`.
4. Faça uma função que retorna o maior elemento de uma lista.
5. Faça uma função que retorna o segundo maior elemento de uma lista.
6. Faça uma função que entra um record no formato do exemplo e retorna se a pessoa é maior de idade ou não
7. Altere o type Person para incluir o e-mail. Faça uma função para verificar se um e-mail existe ou não no record.
8. Fazer uma função que altera todas as ocorrências de uma lista.

```
update_all [1; 2; 3; 2; 4] 2 9 (* trocar todos os 2 por 9: [1; 9; 3; 9; 4] *)
```

## Próxima Aula

---

- High-order functions