# Estruturas de controlo em Python - Recursividade

Carlos Tavares

September 29, 2024

# Python como uma linguagem de programação universal

Uma linguagem de programação universal deve possuir:

- Instruções com um conjunto completo de instruções aritméticas;
- Variáveis, sítios onde guardar informação;
- Sequências de instruções;
- Instruções Se... então... senão;
- Instruções Repita... até;

# Python como uma linguagem de programação universal

Uma linguagem de programação universal deve possuir:

- Instruções com um conjunto completo de instruções aritméticas;
- Variáveis, sítios onde guardar informação;
- Sequências de instruções;
- Instruções Se... então... senão;
- **Instruções Repita... até;**

# Recursividade

## O que é a recursividade?

É uma definição que é constituída por uma instância dela própria. Por exemplo:

**Na linguagem:**

**GNU** - GNU Not Unix

**Na matemática:**

Funções recursivas como a função de Fibonacci:

$$f(n) = f(n-1) + f(n-2); f(0) = 1, f(1) = 1 \tag{1}$$

## Na geometria:

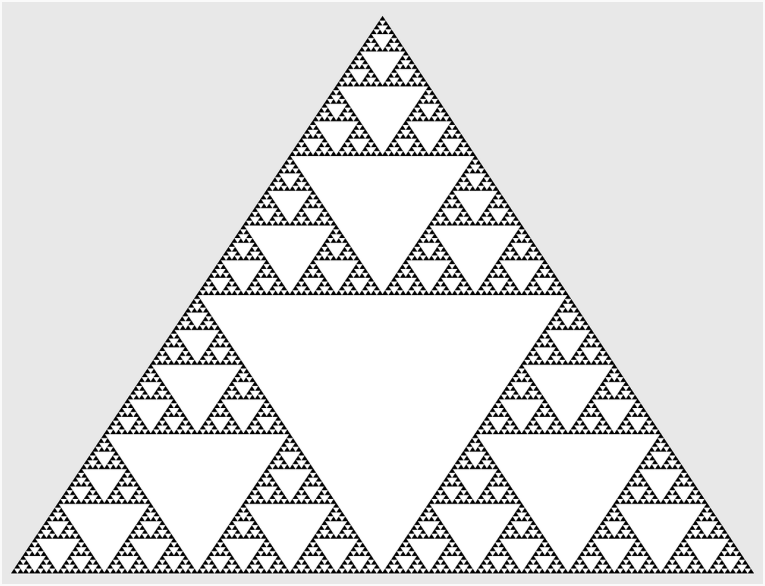

**Figure 1:** Triângulo de Sierpinski

Como pode ser definido o processo de construção do triângulo de Sierpinkski?

## Nas artes:

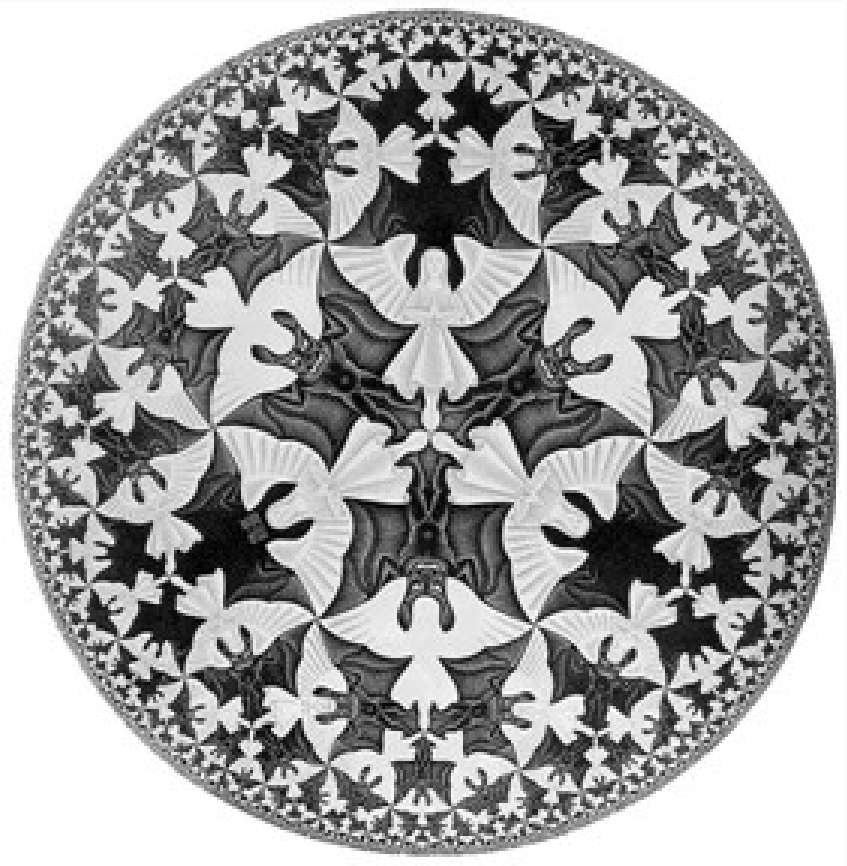

**Figure 2:** Desenho recursivo do Escher

## Na ciência de computação

Recursividade é essencialmente um processo que pode ser definido a partir dele próprio!

Recursividade + instruções **Se... então ... senão** constitutem uma linguagem de programação universal!

Todos os processos computáveis são definíveis através de processos recursivos.

**No Python**

O Python permite recursividade na medida em que nada nos impede de fazer uma função que se invoca a ela própria, por exemplo:

```
def funcao_recursiva_I (x):
    novo_x = (...)
    funcao_recursiva (novo_x)
```

**ou**

```
def funcao_recursiva_II (x):
    return operador (x) ◦ funcao_recursiva (novo_x)
```

◦ - Uma operação qualquer

# Exemplos de funções recursivas

**Função factorial**

!3 = 3 * !2 = 3 * 2 * !1

**Generalizando:**

$!n = n * (n - 1) * (n - 2) * ... * !1 = n * !(n - 1)$

**factorial (n)** = ?

# Exemplos de funções recursivas

**Função factorial**

!3 = 3 * !2 = 3 * 2 * !1

**Generalizando:**

$!n = n * (n-1) * (n-2) * ... * !1 = n * !(n-1)$

**factorial (n)** = n * **factorial (n-1)**

**Implementacao em Python: (?)**

# Exemplos de funções recursivas

**Função factorial**

!3 = 3 * !2 = 3 * 2 * !1

**Generalizando:**

$!n = n * (n-1) * (n-2) * ... * !1 = n * !(n-1)$

**factorial (n)** = n * **factorial (n-1)**

**Implementacao em Python:**

```
def factorial (n):
    return n * factorial (n-1)
```

Qual o resultado? Ciclo infinito! Falta uma condição de paragem.

Deve parar em 1. A condição de paragem normalmente corresponde ao caso base da função. Re-implementando:

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

**Exercicio 1.** Como seria uma implementação em Python da função de Fibonacci?

$$f(n) = f(n-1) + f(n-2); f(0) = 1; f(1) = 1 \tag{2}$$

**Exercicio 2.** Como seria o *esboço* de uma função recursiva para desenhar o triangulo de Sierpinski. Assuma que existe uma funcao desenha_triangulo $(x_1, y_1, x_2, y_2, x_3, y_3)$ para desenhar o triangulo invertido no ecrã.

Como seria uma função para mostrar todos os elementos de 1 a 10?

```
def mostra_elementos (x):
    print ("Elemento: ", x)
    if x == 10:
        return
    else:
        mostra_elementos (x+1)
```

**Exercício.1** E uma função que soma todos os elementos de 1 até n?

**Exercício.2** E uma função que multiplica todos os elementos de 1 até n que sejam multiplos de 3?

# Questions?