Recursão

> "To understand recursion you must first understand recursion"

A definição da função fatorial é

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

No entanto, esta não é uma definição formal.

Uma definição formal da função fatorial é dada através da **recorrência**

$$n! = \begin{cases} 1 & \text{se } n = 0 & \textbf{caso base} \\ n * (n-1)! & \text{se } n > 0 & \textbf{recorrência} \end{cases}$$

Recursão como *efeito dominó*

- Caso base é a primeira pedra do dominó: $0! = 1$
- Recorrência é o *efeito dominó*: $n! = n * (n-1)!$

n	n!	
0	1	caso base
1	1	
2	2	
3	6	
4	24	
5	120	
...	...	

Recursão como cômputo

$$n! = \begin{cases} 1 & \text{se } n = 0 & \textbf{caso base} \\ n * (n-1)! & \text{se } n > 0 & \textbf{recorrência} \end{cases}$$

$$\begin{aligned} 5! &\rightarrow 5 * 4! \\ &\rightarrow 5 * 4 * 3! \\ &\rightarrow 5 * 4 * 3 * 2! \\ &\rightarrow 5 * 4 * 3 * 2 * 1! \\ &\rightarrow 5 * 4 * 3 * 2 * 1 * 0! \\ &\rightarrow 5 * 4 * 3 * 2 * 1 * 1 \\ &\rightarrow \ldots \\ &\rightarrow 120 \end{aligned}$$

Podemos definir fatorial em Python seguindo a definição formal

In [1]:

```
def fatorial(n):
#pré-condição: n >= 0
    if n == 0:
        return 1
    else:
        return n * fatorial(n-1)

fatorial(5)
```

Out[1]:

```
120
```

Como definir funções recursivas

Para **projetar** uma função recursiva **não pense operacionalmente**.

Recursão primitiva em números naturais

Siga os seguintes passos:

1. Escolha o argumento sobre o qual irá fazer recursão, digamos que seja `n`
2. Defina a função para o caso base `n == 0`.
3. Se `n > 0`, suponha que um **bom amigo** calcula a função (resolve o problema) para o caso `n-1`
4. Componha a solução para `n` usando a solução do bom amigo
5. *Tenha fé*, acredite no método

A definição da função `fatorial` anterior é um exemplo de recursão primitiva

Exemplo: Dados os números x e n, real e natural respectivamente, calcular a potência x^n

Escolhemos fazer recursão em n

In [4]:

```
def pot(x, n):
    if n == 0:
        return 1
    else:
        return x * pot(x, n-1)

pot(2.5, 4)
```

Out[4]:

```
39.0625
```

Exemplo: Cálculo do número de Euler e

Aproximações da constante e podem ser obtidas a partir da seguinte série

$$e = \sum_{i}^{\infty} \frac{1}{i!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{(n-2)!} + \frac{1}{(n-1)!} + \frac{1}{(n)!} + \ldots$$

Queremos escrever uma função que, dado n, calcule a aproximação de e usando n termos desta série

In [7]:

```
def eAprox(n):
    if n == 0:
        return 1
    else:
        return 1 / fatorial(n) + eAprox(n-1)

eAprox(10)
```

Out[7]:

```
2.7182818011463845
```

Recursão primitiva em listas e strings

Similar a recurão primitiva em naturais. Para listas:

1. Escolha o argumento sobre o qual irá fazer recursão, digamos que seja `xs`
2. Defina para o caso base, quando `xs == []` .
3. Se `xs != []` , suponha que um **bom amigo** calcula a função (resolve o problema) para uma lista que tira um elemento de `xs` .
4. Componha a solução para `xs` usando a solução do bom amigo
5. *Tenha fé*, acredite no método

Exemplo: somar todos os elementos de uma lista

Na recursão, tirando o primeiro elemento da lista :

In [11]:

```
def soma(xs):
    if xs == []:
        return 0
    else:
        return xs[0] + soma(xs[1:])

soma([2, 5, 6, 4, 2])
```

Out[11]:

```
19
```

Na recursão, tirando o último elemento da lista:

In [13]:

```python
def soma(xs):
    if xs == []:
        return 0
    else:
        return soma(xs[:len(xs)-1]) + xs[len(xs)-1]

soma([2, 5, 6, 4, 3])
```

Out[13]:

```
20
```

Exemplo: Contar quantas vogais tem um string

In [15]:

```python
def nroDeVogaisEm(txt):
    if txt == "":
        return 0
    else:
        if txt[0] in "aeiouAEIOU":
            return 1 + nroDeVogaisEm(txt[1:])
        else:
            return nroDeVogaisEm(txt[1:])

nroDeVogaisEm("A casa de Maria Joana")
```

Out[15]:

```
10
```

In [18]:

```python
def nroDeVogaisEm(txt):
    if txt == "":
        return 0
    else:
        if txt[len(txt)-1] in "aeiouAEIOU":
            return 1 + nroDeVogaisEm(txt[:len(txt)-1])
        else:
            return nroDeVogaisEm(txt[:len(txt)-1])

nroDeVogaisEm("A casa de Maria Joana")
```

Out[18]:

```
10
```