

Réursion

Tristan Crolard

Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

`tristan.crolard@cnam.fr`

`cedric.cnam.fr/sys/crolard`

Recursion

A function is **recursive** when it **calls itself** (but on **smaller values**).

A recursive definition consists in a **recursive step** and a **base case**.

Example: factorial of n (written $n!$)

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{otherwise} \end{cases}$$

- ▶ **Recursive step.** Computing the factorial $n!$ can be done by computing $(n-1)! \times n$. We check that $n-1$ is smaller than n .
- ▶ **Base case.** For the factorial, we know that $0! = 1$.
- ▶ **Evaluation.** For instance with $n = 4$:

$$\begin{aligned} 4! &= 3! \times 4 \\ &= 2! \times 3 \times 4 \\ &= 1! \times 2 \times 3 \times 4 \\ &= 0! \times 1 \times 2 \times 3 \times 4 \\ &= 1 \times 1 \times 2 \times 3 \times 4 \\ &= 24 \end{aligned}$$

Recursive function – factorial

Here is the Python function that computes the factorial recursively:

```
>>> def factorial(n: int) -> int:
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

```
>>> factorial(0)
```

1

```
>>> factorial(3)
```

6

```
>>> factorial(4)
```

24

Recursion over lists

Example: length of a list l (written $\#l$)

$$\#l = \begin{cases} 1 + \#t & \text{if } l = [h, *t] \\ 0 & \text{otherwise} \end{cases}$$

- ▶ **Recursive step.** Computing the length of a list l , can be done by computing $1 + \#t$, where $t = [x_1, \dots, x_{n-1}]$, if l is not empty, and $l = [x_0, x_1, \dots, x_{n-1}]$. Again, we check that t is smaller than l .
- ▶ **Base case.** For an empty list, we know that $\#[] = 0$.
- ▶ **Evaluation.** For instance with $l = [10, 21, 32, 43]$:

$$\begin{aligned} \#[10, 21, 32, 43] &= 1 + \#[21, 32, 43] \\ &= 1 + 1 + \#[32, 43] \\ &= 1 + 1 + 1 + \#[43] \\ &= 1 + 1 + 1 + 1 + \#[] \\ &= 1 + 1 + 1 + 1 + 0 \\ &= 4 \end{aligned}$$

Recursive function – length

Here is the Python function that computes the length function recursively:

```
>>> def length[A](l: list[A]) -> int:
    match l:
        case [_, *t]:
            return 1 + length(t)
        case _:
            return 0
```

```
>>> length([10, 21, 32, 43])
```

4

```
>>> length([])
```

0

Recursive function – sum

Here is the Python function that computes the sum function recursively:

```
>>> def sum(l: list[int]) -> int:
    match l:
        case [h, *t]:
            return h + sum(t)
        case _:
            return 0
```

```
>>> sum([10, 21, 32, 43])
```

106

```
>>> sum([])
```

0

Recursive function – join

Here is the Python function that computes the join function recursively:

```
>>> def join(l: list[str], delim: str = " ") -> str:
    match l:
        case [h]:
            return h
        case [h, *t]:
            return h + delim + join(t, delim)
        case _:
            return ""
```

```
>>> print(join(["I", "am", "only", "human"]))
```

I am only human

```
>>> print(join(["I", "am", "only", "human"], "_"))
```

I_am_only_human

```
>>> print(join([]))
```