Recursion | Mitsiu Alejandro Carreño Sarabia

Agenda

- Recap
- Recursion
- Footnote on recursion
- Exercises
- Homework

Recap

Code recap

What can we infer from the following type annotation?

```
something : Bool -> String
```

What does this function do?

```
getIdByName : String -> Int
getIdByName name =
    case name of
        "Fernanda" ->1
        "Miguel" ->2
        "Juan" ->3
        _ ->0
```

```
getGradeById : Int -> Float
getGradeById id =
    case id of
        1 ->9.4
        2 ->8.7
        3 ->9.3
        _ ->0
```

```
getFinalGrade : String -> Float
getFinalGrade name =
    getGradeById (getIdByName name)
```

Commands recap

Create an elm project?

- elm init

Start an elm repl session?

- elm repl

Track changes in git?

- git add <file>

Commit changes in git?

- git commit -m'description'

Push to remote repo in git?

- git push origin main

Verify elm code compilation?

- elm make src/*

Our old ways

Remember how in imperative contexts we have:

```
1 int x = 0;
2 for (int i = 0; i < 5; i++){
3     x = x + 1;
4 }
```

- We agreed that `x = x + 1` on line 3 is no bueno because we modify the value
- By the same principle `i++` (`i = i + 1`) on line 2 must be gone

- Actually the whole concept of a for loop needs to be gone, so how is this done in functional programming?

Recursion

Recursion

> The definition of a concept or process depends on a simpler or previous version of itself.

Let's analyse factorial number for example:

$$fact(5) = 1 \times 2 \times 3 \times 4 \times 5 = 120$$
$$fact(4) = 1 \times 2 \times 3 \times 4 = 24$$
$$fact(3) = 1 \times 2 \times 3 = 6$$
$$\dots$$
$$fact(1) = 1$$

Recursion

Let's suppose we want to calculate fact(5), is there a simpler version of this problem?

$$fact(5) = 1 \times 2 \times 3 \times 4 \times 5 = 120$$
$$fact(4) = 1 \times 2 \times 3 \times 4 = 24$$
$$fact(3) = 1 \times 2 \times 3 = 6$$
$$\dots$$
$$fact(1) = 1$$

There is!

$$fact(5) = fact(4) \times 5 = 120$$
$$fact(4) = 1 \times 2 \times 3 \times 4 = 24$$
$$fact(3) = 1 \times 2 \times 3 = 6$$
$$\dots$$
$$fact(1) = 1$$

And we can keep reducing our problem until it's really trivial

$$fact(5) = fact(4) \times 5 = 120$$
$$fact(4) = fact(3) \times 4 = 24$$
$$fact(3) = fact(2) \times 3 = 6$$
$$fact(2) = fact(1) \times 2 = 2$$
$$fact(1) = 1$$

We can summarize this behaviour as, for all natural numbers n

if n=1 then
$$fact(1) = 1$$
else
$$fact(n) = fact(n-1) \times n$$

We can summarize this behaviour as, for all natural numbers n

if n=1 then

$$fact(1) = 1$$

else

$$fact(n) = fact(n-1) \times n$$

We can express this notion using case notation as:

$$fact(n) = \begin{cases} 1, & \text{if } n = 1 \\ n * fact(n-1), & \text{if } n > 0 \end{cases}$$

Coding factorial

$$fact(n) = \begin{cases} 1, & \text{if } n = 1 \\ n * fact(n-1), & \text{if } n > 0 \end{cases}$$

What's our function type?

```
factorial : Int -> Int
```

What's our base case? (The simplest version of the problem)

```
factorial : Int -> Int
factorial n =
    -- Requires n >= 0
    -- Ensures factorial of n
    case n of
        1 ->
            1
```

Coding factorial

$$fact(n) = \begin{cases} 1, & \text{if } n = 1 \\ n * fact(n-1), & \text{if } n > 0 \end{cases}$$

Current progress:

```
factorial : Int -> Int
factorial n =
    -- Requires n >= 0
    -- Ensures factorial of n
    case n of
        1 ->
            1
```

Let's assume that the function already works on a "smaller" input.

You dont have to think about all the computation and subcomputations, remember that:

$$fact(5) = fact(4) \times 5 = 120$$

Coding factorial

$$fact(n) = \begin{cases} 1, & \text{if } n = 1 \\ n * fact(n-1), & \text{if } n > 0 \end{cases}$$

Let's assume that the function already works on a "smaller" input.

You dont have to think about all the computation, remember that:

$$fact(5) = fact(4) \times 5 = 120$$

Let's take the recursive leap of faith:

```
factorial : Int -> Int
factorial n =
    -- Requires n >= 0
    -- Ensures factorial of n
    case n of
        1 ->
            1
        _ ->
            factorial(n-1) * n
```

Coding factorial

```
factorial : Int -> Int
factorial n =
    -- Requires n >= 0
    -- Ensures factorial of n
    case n of
        1 ->
            1
        _ ->
            factorial(n-1) * n
```

Let's trace the factorial 5 application:

$$fact(5) = fact(4) \times 5$$

$$fact(4) = fact(3) \times 4$$

$$fact(3) = fact(2) \times 3$$

$$fact(2) = fact(1) \times 2$$

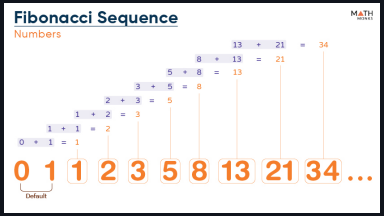
$$fact(1) = 1$$

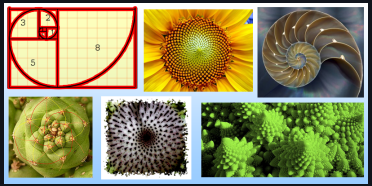
Coding factorial

$$fact(5) = fact(4) \times 5$$
$$fact(4) = fact(3) \times 4$$
$$fact(3) = fact(2) \times 3$$
$$fact(2) = fact(1) \times 2$$
$$fact(1) = 1$$

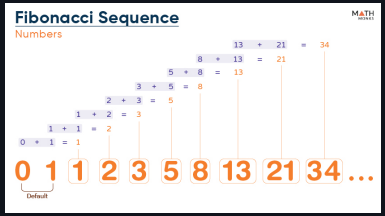
$$fact(2) = 1 \times 2$$
$$fact(3) = 1 \times 2 \times 3$$
$$fact(4) = 1 \times 2 \times 3 \times 4$$
$$fact(5) = 1 \times 2 \times 3 \times 4 \times 5$$
$$fact(5) \Rightarrow 120$$

Fibonacci


Fibonacci Sequence
Numbers
MATH
0 + 1 = 1
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 13
8 + 13 = 21
13 + 21 = 34
0 1 1 2 3 5 8 13 21 34 ...
Default


3
5
8

Fibonacci



To calculate the next number we have to add the two previous numbers

Fibonacci

Let's suppose we want to calculate fib(6), is there a simpler version of this problem?

$$fib(6) = 0 + 1 + 1 + 2 + 3 + 5 = 8$$
$$fib(5) = 0 + 1 + 1 + 2 + 3 = 5$$
$$fib(4) = 0 + 1 + 1 + 2 = 3$$
$$fib(3) = 0 + 1 + 1 = 2$$
$$fib(2) = 0 + 1 = 1$$
$$fib(1) = 1$$
$$fib(0) = 0$$

$$fib(6) = fib(5) + fib(4) = 8$$
$$fib(5) = 0 + 1 + 1 + 2 + 3 = 5$$
$$fib(4) = 0 + 1 + 1 + 2 = 3$$
$$fib(3) = 0 + 1 + 1 = 2$$
$$fib(2) = 0 + 1 = 1$$
$$fib(1) = 1$$
$$fib(0) = 0$$

Fibonacci

We can continue simplifing all the way down

$$fib(6) = fib(5) + fib(4) = 8$$
$$fib(5) = fib(4) + fib(3) = 5$$
$$fib(4) = fib(3) + fib(2) = 3$$
$$fib(3) = fib(2) + fib(1) = 2$$
$$fib(2) = fib(1) + fib(0) = 1$$
$$fib(1) = 1$$
$$fib(0) = 0$$

We can express this notion using case notation as:

$$fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{if } \ n > 1 \end{cases}$$

Coding Fibonacci

$$fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{if } n > 1 \end{cases}$$

What's our function type?

```
fib: Int -> Int
```

What's our base case? (The simplest version of the problem)

```
fib : Int -> Int
fib n =
    -- Requires n >= 0
    -- Ensures nth fibonacci number
    case n of
        0 ->
            0
        1 ->
            1
```

Coding Fibonacci

$$fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{if } n > 1 \end{cases}$$

Current progress:

```
fib : Int -> Int
fib n =
    -- Requires n >= 0
    -- Ensures nth fibonacci number
    case n of
        0 ->
            0
        1 ->
            1
```

Let's assume that the function already works on a "smaller" input.

You dont have to think about all the computation and subcomputations, remember that:

$$fib(6) = fib(5) + fib(4) = 8$$

Coding Fibonacci

$$fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{if } n > 1 \end{cases}$$

Let's assume that the function already works on a "smaller" input.

You dont have to think about all the computation, remember that:

$$fib(6) = fib(5) + fib(4) = 8$$

Let's take the recursive leap of faith:

```
fib : Int -> Int
fib n =
    -- Requires n >= 0
    -- Ensures nth fibonacci number
    case n of
        0 ->
            0
        1 ->
            1
        _ ->
            fib(n-1) + fib(n-2)
```


Footnote on recursion

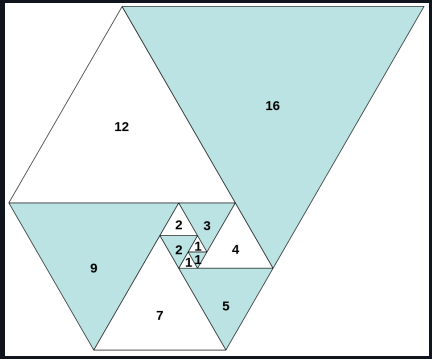
Footnote on recursion

Be aware that recursion in the context of functional programming is usefull but it has some risks.

- Recursion can lead to infinite loops if our base case isn't set properlly.

Exercises

Exercise Padovan



Exercise Padovan

```
[1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49, 65, 86, 114, 151, 200,
265,]
```

$$P(n) = P(n-2) + P(n-3) \text{ for } n \geq 3, with$$
$$P(0) = P(1) = P(2) = 1$$

▒▒▒ Exercise Pow

Create a function to calculate any:

$$n \hat{} p \text{ Given :}$$
$$n \hat{} 3 = n * n * n$$

This will require two parameters:

```
powerOf : Int -> Int -> Int
powerOf number power =
    ...
```