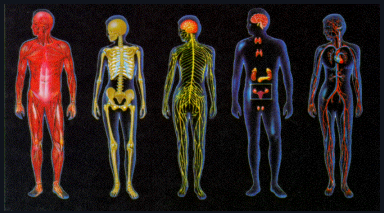
Records

Mitsiu Alejandro Carreño Sarabia



Agenda

- Recap
- Records
- Aliases
- Homework

Recap

Code recap

What can we infer from the following type annotation?

```
mystery : Char -> (Char -> Int) -> (Int -> Bool) -> Bool
```

Let's dig deeper into mystery how can we apply it?

```
Char.toCode 'A' -- Reduces to 65
Char.toCode 'a' -- Reduces to 97

lesserThan97 : Int -> Bool
lesserThan97 x =
    x < 97
```

Code recap

```
Char.toCode 'A' -- Reduces to 65
Char.toCode 'a' -- Reduces to 97

lesserThan97 : Int -> Bool
lesserThan97 x =
    x < 97

mystery 'C' Char.toCode lesserThan97
mystery : Char -> (Char -> Int) -> (Int -> Bool) -> Bool
```

Does this type check?

Code recap

```elm
Char.toCode 'A' -- Reduces to 65
Char.toCode 'a' -- Reduces to 97

lesserThan97 : Int -> Bool
lesserThan97 x =
    x < 97

mystery : Char -> (Char -> Int) -> (Int -> Bool) -> Bool
mystery letter fun1 fun2 =
    fun2 (fun1 letter)


mystery 'C' Char.toCode lesserThan97
```

What does mystery do?

Commands recap

Create an elm project?

- `elm init`

Start an interactive session?

- `elm repl`

Check format (coding style convetions)?

- `elm-format src/`

Validate the code is elm compliant code?

- `elm make src/*`

Enforce format rules on our code?

- `elm-format src/`

Homeworks recap

In your homework repositories, please refrain from modifying the `tests/TestSuite.elm` file.

Points will be deducted if you modify it!

Records

Records

Sometimes primitives are not enough to express all the information we want. A person might be more complex than a simple String, we might want to add the age, and the email, that's exactly why we have records:

```
mit : { name : String, age : Int, email : String}
mit =
    { name = "Mitsiu"
    , age = 32
    , email = "mitsiu.carreno@domain.com"
    }
```

Records

We can access the properties like variable.property

```
mit : { name : String, age : Int, email : String}
mit =
    { name = "Mitsiu"
    , age = 32
    , email = "mitsiu.carreno@domain.com"
    }

mit.name
```

Or we can treat the property as a function

```
.name : { b | age : a} -> a
.name mit
```

Records

Let's make a function that output's the person email:

```
getEmail : { name : String, age: Int, email : String } -> String
getEmail person =
    .email person
```

Records

Let's update our variable:

```
mit : { name : String, age : Int, email : String}
mit =
    { name = "Mitsiu"
    , age = 32
    , email = "mitsiu.carreno@domain.com"
    }


{ mit | email = "mitsiu.carreno@upa" }
```

We can read it out loud as "Create a new record that takes everything from mit, and update email to mitsiu.carreno@upa"

Records

```
mit : { name : String, age : Int, email : String}
mit =
    { name = "Mitsiu"
    , age = 32
    , email = "mitsiu.carreno@domain.com"
    }


{ mit | email = "mitsiu.carreno@upa" }
```

We can read it out loud as "Create a new record that takes everything from mit, and update email to mitsiu.carreno@upa"

What would mit evaluate to?

Records

mit remains unchanged! Remember we don't modify the state.

```
{ age = 32, email = "mitsiu.carreno@domain.com", name = "Mitsiu" }
```

But we can bind the new value to a new variable.

```
mit : { name : String, age : Int, email : String}
mit =
    { name = "Mitsiu"
    , age = 32
    , email = "mitsiu.carreno@domain.com"
    }

mitV2 = { mit | email = "mitsiu.carreno@upa" }
```

Records

How about this expression, what does it evaluate to?

```
List.map : (a -> b) -> List a -> List b
List.map .age [mit, mit, mit]
```

Which is it's data type? What does it output?

```
[32, 32, 32]
```

What about?

```
List.map .email [mit, mit, mit]

List.map .email [mitV2, mitV2, mit]
```

Records exercises

1.0 Let's define a record for programming languages with:

- name : String
- releaseYear : Int
- currentVersion: String

1.1 Create a list with at least two programming languages
1.2 Create a function "languageNames" that receives the list from point 1.1 and generates a String list with only the names eg:

- Input : [{name="elm", releaseYear= 2012, currentVersion="0.19.1"},{name="javascript", releaseYear= 1995, currentVersion="ECMAScript 2025"}]
- Output: ["elm", "javascript"]

Records exercises

2.0. Let's define a record for user with:

- name : String
- uType : String

2.1 Create a list of users (uType can be "Student" or "Professor")
2.2 Create a function "onlyStudents" that receives the list from point 2.1 and generates a String list with only the name of the "Student" (professors names are returned as an empty string) eg:

- Input : [{name="Roberto", uType= "Student"}, {name="Mitsiu", uType="Professor"}]
- Output: ["Roberto", ""]


Aliases

Aliases

As you might experience, writting a record type annotations can be exahusting:

```
getType : {name: String, uType: String } -> String
getType user =
    case .uType user of
        "Student" ->
            .name user ++ " es un estudiante"
        "Professor" ->
            .name user ++ " es un maestro"
        _ ->
            .name user ++ " no es maestro ni estudiante"
```

Specially when we already defined that {name : String, uType: String } in the contex of our program is a user.

Aliases

We can define a type alias which is simply a shorter name for a type

```
type alias User =
    {name: String, uType: String }

-- getType : {name: String, uType: String} -> String
getType : User -> String
getType user =
    case .uType user of
        "Student" ->
            .name user ++ " es un estudiante"
        "Professor" ->
            .name user ++ " es un maestro"
        _ ->
            .name user ++ " no es maestro ni estudiante"
```

Aliases exercise

3.0 Let's define a record for games aliased "Videogame":

- title : String
- releaseYear : Int
- available: Bool
- downloads: Int
- genres : List String

3.1 Create a list with at least two videogames
3.2 Create a function "getVideogameGenres" that receives the list from point 1.2 and generates a List of List of strings with only the genres eg:

- Input : [{title="Control", releaseYear=2019, ... genres=["Action", "Shooter"]}, {title="Ocarina of time", ... genres=["Action", "Adventure"]}]
- Output: [["Action", "Shooter"], ["Action", "Adventure"]]

Homework

Let's define a record named "Computer" with:

- ram: String
- model: String
- brand: String
- screenSize: String

And create a variable "myLaptop" of type Computer

Finally, let's make a variable "main" that reduces to:

```
<div>
  <h1>My laptop<h1>
  <div>
    <ul>
      <li>Ram: {{.ram myLaptop}}</li>
      <li>Modelo: {{.model myLaptop}}</li>
      <li>Marca: {{.brand myLaptop}}</li>
      <li>Pulgadas: {{.screenSize myLaptop}}</li>
    </ul>
  </div>
</div>
```