Aprendimos sobre mapas:

```
iex> map = %{a: 1, b: 2}
%{a: 1, b: 2}
iex> map[:a]
1
iex> %{map | a: 3}
%{a: 3, b: 2}
```

Las estructuras son extensiones construidas sobre mapas que proporcionan comprobaciones en tiempo de compilación y valores predeterminados.

Definiendo structs

Para definir una estructura, se utiliza la construcción defstruct:

```
iex> defmodule User do
...>   defstruct name: "John", age: 27
...> end
```

La lista de palabras clave utilizada con defstruct define qué campos tendrá la estructura junto con sus valores predeterminados.

Las estructuras toman el nombre del módulo en el que están definidas. En el ejemplo anterior, definimos una estructura llamada User.

Ahora podemos crear estructuras User utilizando una sintaxis similar a la utilizada para crear mapas (si ha definido la estructura en un archivo separado, puede compilar el archivo dentro

de IEx antes de continuar ejecutando c “file.exs”. Ten en cuenta es posible que reciba un error que indique “the struct was not yet defined” si prueba el siguiente ejemplo en un archivo directamente debido a cuando se resuelven las definiciones):

```
iex> %User{}
%User{age: 27, name: "John"}
iex> %User{name: "Jane"}
%User{age: 27, name: "Jane"}
```

Las estructuras proporcionan garantías en tiempo de compilación de que solo los campos (y todos ellos) definidos a través de defstruct podrán existir en una estructura:

```
iex> %User{oops: :field}
** (KeyError) key :oops not found in: %User{age: 27, name: "John"}
```

Acceder y actualizar structs

Cuando hablamos de mapas, mostramos cómo podemos acceder y actualizar los campos de un mapa. Las mismas técnicas (y la misma sintaxis) también se aplican a las estructuras:

```
iex> john = %User{}
%User{age: 27, name: "John"}
iex> john.name
"John"
iex> jane = %{john | name: "Jane"}
%User{age: 27, name: "Jane"}
iex> %{jane | oops: :field}
```

```
** (KeyError) key :oops not found in: %User{age: 27, name: "Jane"}
```

Cuando se usa la sintaxis de actualización, la VM es consciente de que no se agregarán nuevas claves a la estructura, lo que permite que los mapas debajo compartan su estructura en la memoria. En el ejemplo anterior, tanto John como Jane comparten la misma estructura clave en la memoria.

Las estructuras también se pueden usar en la coincidencia de patrones, tanto para la coincidencia en el valor de claves específicas como para garantizar que el valor de coincidencia sea una estructura del mismo tipo que el valor coincidente.

```
iex> %User{name: name} = john
%User{age: 27, name: "John"}
iex> name
"John"
iex> %User{} = %{}
** (MatchError) no match of right hand side value: %{}
```

Structs son mapas desnudos debajo

En el ejemplo anterior, la coincidencia de patrones funciona porque debajo de las estructuras hay mapas desnudos con un conjunto fijo de campos. Como mapas, las estructuras almacenan un campo “especial” llamado __struct__ que contiene el nombre de la estructura:

```
iex> is_map(john)
true
iex> john.__struct__
```

```
User
```

Tenga en cuenta que nos referimos a las estructuras como mapas desnudos porque ninguno de los protocolos implementados para los mapas están disponibles para estructuras. Por ejemplo, no puede enumerar ni acceder a una estructura:

```
iex> john = %User{}
%User{age: 27, name: "John"}
iex> john[:name]
** (UndefinedFunctionError) function User.fetch/2 is undefined
(User does not implement the Access behaviour)
             User.fetch(%User{age: 27, name: "John"}, :name)
iex> Enum.each john, fn({field, value}) -> IO.puts(value) end
** (Protocol.UndefinedError) protocol Enumerable not implemented
for %User{age: 27, name: "John"}
```

Sin embargo, dado que las estructuras son solo mapas, funcionan con las funciones del módulo Map:

```
iex> jane = Map.put(%User{}, :name, "Jane")
%User{age: 27, name: "Jane"}
iex> Map.merge(jane, %User{name: "John"})
%User{age: 27, name: "John"}
iex> Map.keys(jane)
[:__struct__, :age, :name]
```

Valores predeterminados y claves requeridas

Si no especifica un valor de clave predeterminado al definir una estructura, se supondrá nil:

```
iex> defmodule Product do
...>   defstruct [:name]
...> end
iex> %Product{}
%Product{name: nil}
```

Puede definir una estructura que combine ambos campos con valores predeterminados explícitos y valores nil implícitos. En este caso, primero debe especificar los campos que están implícitamente predeterminados en nil:

```
iex> defmodule User do
...>   defstruct [:email, name: "John", age: 27]
...> end
iex> %User{}
%User{age: 27, email: nil, name: "John"}
```

Hacerlo en orden inverso generará un error de sintaxis:

```
iex> defmodule User do
...>   defstruct [name: "John", age: 27, :email]
...> end
** (SyntaxError) iex:107: syntax error before: email
```

También puede exigir que ciertas claves tengan que especificarse al crear la estructura:

```
iex> defmodule Car do
...>   @enforce_keys [:make]
...>   defstruct [:model, :make]
...> end
iex> %Car{}
** (ArgumentError) the following keys must also be given when
building struct Car: [:make]
    expanding struct: Car.__struct__/1
```