Level 3–2

Tuples & Maps

Keyword Lists & Defaults

Listing Account Balance

An existing Account.balance **function prints a balance based on a list of transactions.**

```
Account.balance(transactions)
```

```
Balance: 200
```

We want to pass formatting options, like <u>currency</u> (dollars, euros, GBP) and <u>symbols</u> ($, £, €)...

```
Account.balance(transactions, ............ )
```

Options argument

```
Balance in dollars: $200
```

```
Balance in GBP: £200
```

```
Balance in euros: €200
```

* MIXING IT UP * with * ELIXIR *

Passing Options With Keyword Lists

A keyword list is a **list of two-value tuples.** They are typically used as the last argument in function signatures, representing **options** passed to the function.

Keyword list shortcut

```elixir
Account.balance(..., currency: "dollar", symbol: "$")
```

Same thing

Keyword list full version

```elixir
Account.balance(..., [{:currency, "dollar"}, {:symbol, "$"}])
```

This is a tuple...

...and this is a tuple too!

Reading Keyword Lists

To read values from keyword lists, we can use `[]` and the variableName[keyName] notation.

```
defmodule Account do
  def balance(transactions, options) do
    currency = options[:currency]
    symbol = options[:symbol]

    balance = calculate_balance(transactions)
    "Balance in #{currency}: #{symbol}#{balance}"
  end
  ...
end
```

formatting options

Read values

Values read from options

Running With Options

The Account.balance **function now accepts formatting options!**

```elixir
defmodule Account do
  def balance(transactions, options) do
    currency = options[:currency]
    symbol = options[:symbol]

    balance = calculate_balance(transactions)
    "Balance in #{currency}: #{symbol}#{balance}"
  end
  ...
end
```

```elixir
Account.balance(transactions,
         currency: "euros", symbol: "€")
```

```
Balance in euros: €200
```

Must Pass All Arguments

The code currently expects options to **always be passed.** Otherwise, it raises an error.

```
defmodule Account do
  def balance(transactions, options) do
    currency = options[:currency]
    symbol = options[:symbol]
    ...
  end
  ...
end
```

Expects second argument to always be passed

```
Account.balance(transactions)
```

Passing a single argument breaks the code

```
** (UndefinedFunctionError) function Account.balance/1
is undefined or private. Did you mean one of:

      * balance/2
```

Default Function Arguments

The `\\` symbol sets a default value to be used when none is passed during function call.

```
defmodule Account do
  def balance(transactions, options \\ []) do
    currency = options[:currency]
    symbol = options[:symbol]
    ...
  end
  ...
end
```

Defaults the options *argument to empty list*

No values returned!

```
Account.balance(transactions)
```

Code does not break anymore...

...but it's missing options!

```
Balance in          :  200
```

Defaults for Reading Keyword Lists

The logical **OR** operator `||` can be used to return a **default value** when a key is not present.

```
defmodule Account do
  def balance(transactions, options \\ []) do
    currency = options[:currency] || "dollar"
    symbol = options[:symbol] || "$"
    ...
  end
  ...
end
```

If left side of || does not return a value...

...then return this value on right side.

```
Account.balance(transactions)
```

```
Balance in dollars: $200
```

animated these dotted lines and this side-text last

er defaults! 👍

Using Keyword Lists With the Ecto Library

The Ecto library uses keyword lists to build SQL statements from Elixir code.

This is a keyword list

```elixir
Repo.all( from u in User,
  where: u.age > 21,
  where: u.is_active == true )
```

Generated SQL

```sql
SELECT * FROM users
WHERE age >= 21 AND is_active = TRUE
```

* MIXING IT UP * with * ELIXIR *