Episode 08: Comprehensions

Summary

Three main elements of the `for` macro:

- Generators
- Filters
- `:into`

`for` macro

`for` combines features of `map`, `filter` and `into`.

```
for element <- Enumerable do
  element
end
# => Returns a list
```

Generators

Multiple generators

```
suits = [:hearts, :diamonds, :clubs, :spades]
faces = [2, 3, 4, 5, 6, 7, 8, 9, 10,
         :jack, :queen, :king, :ace]

for suit <- suits,
    face <- faces,
    do: {suit, face}

# => [{:hearts, 2}, {:hearts, 3}, {:hearts, 4},
#      {:hearts, 5}, {:hearts, 6}, {:hearts, 7}, ...]
```

Filtering with a pattern

```
for {:spades, face} <- deck do
  {:spades, face}
end
```

Filters (similar to guards)

Can have multiple filters such that each filter follows a comma, just like for generators.

```
for number <- [1, 2, 3, 4],
    letter <- ["a", "b"],
    is_integer(number),
    number > 2,
    do: {number, letter}
# => [{3, "a"}, {3, "b"}, {4, "a"}, {4, "b"}]
```

`:into` (similar to `Enum.into/2`)

Use `:into` to collect results into a `Collectable`. The following types support `Collectable`:

- Map
- List
- IO.Stream
- Bitstring (Binary)

This example filters a map and uses `:into` to turn the resulting keyword list back into a map.

```
user = %{name: "Peyton", dob: 2000, email: "..."}

for {key, val} <- user,
    key in [:name, :email],
    into: %{},
    do: {key, val}
# => %{email: "...", name: "Peyton"}
```

For comparison, here is an equivalent expression using `Stream` and Enum.

```
user
|> Stream.filter(fn ({key, val}) -> key in [:name, :email] end)
|> Enum.into(%{})
# => %{email: "...", name: "Peyton"}
```

Binary comprehensions

A sneak peak at comprehensions using binaries (explored further in *Episode 23: Binary*).

```
pixels = <<213, 45, 132, 64, 76, 32, 76, 0, 0, 234, 32, 15>>
for <<r::8, g::8, b::8 <- pixels>>>, do: {r, g, b}

# => [{213, 45, 132}, {64, 76, 32,}, {76, 0, 0}, {234, 32, 15}]
```

Comparison: Enum vs. `Stream` vs. `for`

	Enum	Stream	`for`
`map`	YES	YES	YES
`filter`	YES	YES	YES
Lazy	NO	YES	NO
Iterations	DEPENDS	ONE	ONE
& Operator	YES	YES	NO

> Because `for` uses a do block, rather than an anonymous function, it does not support the capture operator. This can make either Enum or `Stream` a more elegant choice when you just want to run a function on each element.

General usage guideline

Scenario	Recommended	Other notes
One operation	Enum or `for`	Personal preference.

Scenario	Recommended	Other notes
Multiple operations	`for` or `Stream`	Use `for` most of the time. Consider `Stream` for building operations before doing work.
Generating a list	`for` or a `Stream` generator	Use `for` most of the time.
Multiple lists	`for`	Think of using `for` first.

> As a beginner, try to overuse the `for` macro for a while to learn what it's good and bad at.