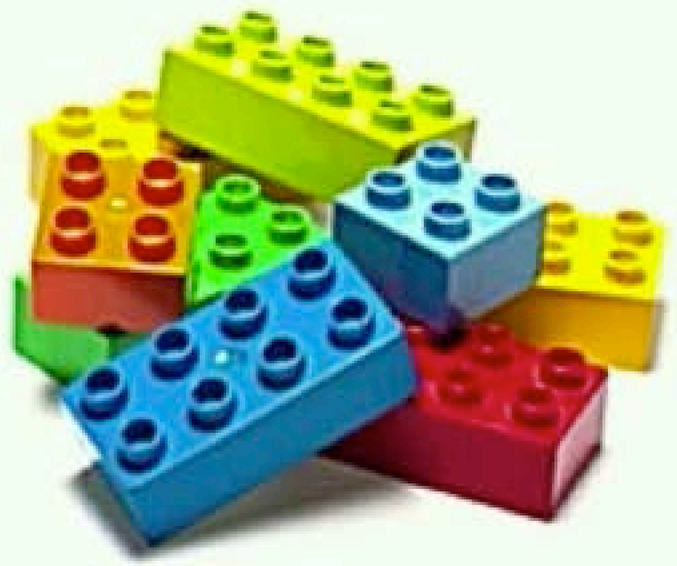
Metaprogramming Elixir

@chris_mccord

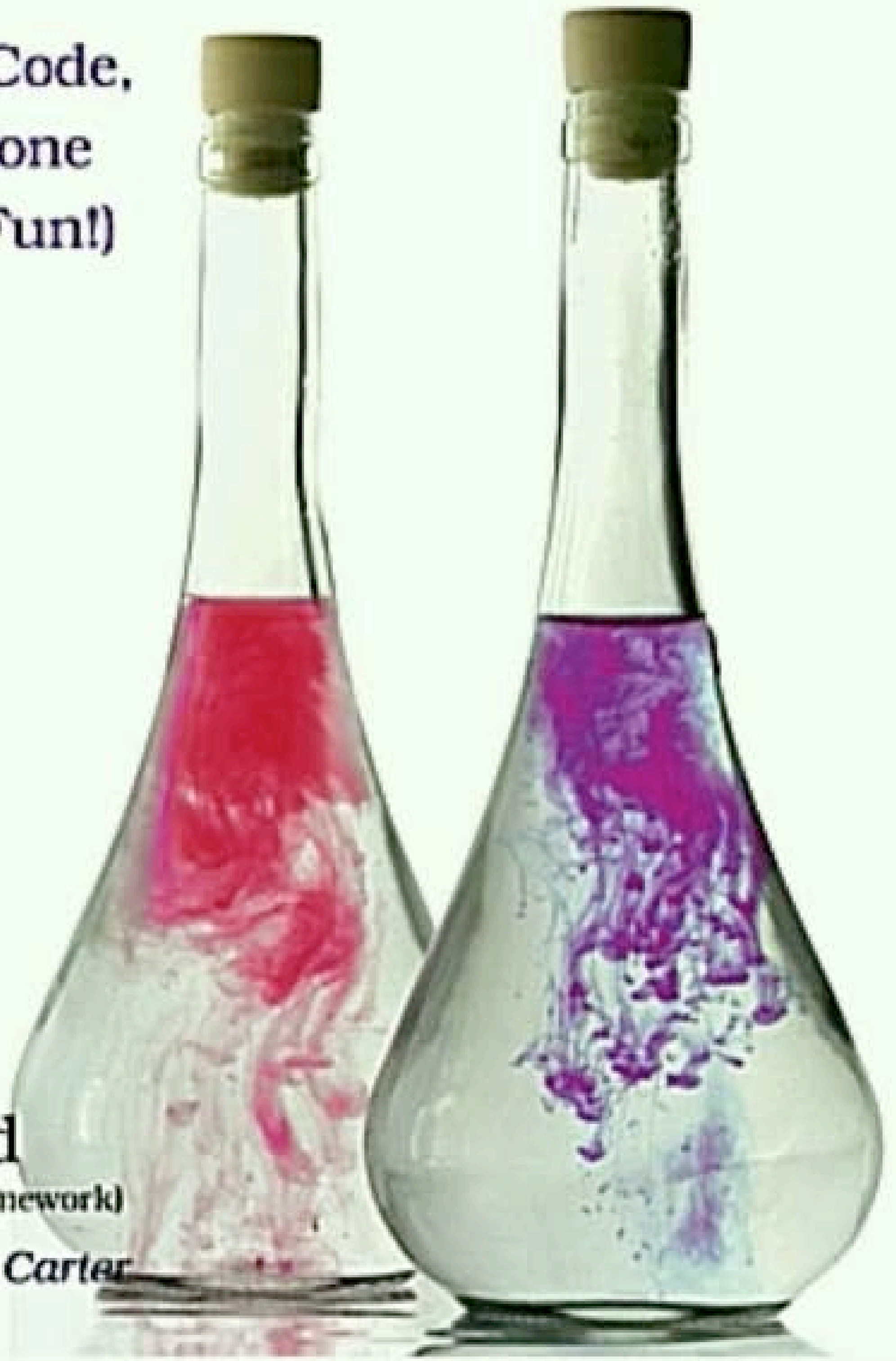


The Pragmatic Programmers

Pragmatic exPress

Metaprogramming Elixir

Write Less Code, Get More Done (and Have Fun!)



Chris McCord
(author of the Phoenix framework)

Edited by Jacquelyn Carter

The rules of macros

#1 Don’t write macros.

#2 Use macros gratuitously

Topics

- What is metaprogramming / macros
- What makes Elixir different
- Real-world usecases
  - Phoenix
  - Ecto
  - String.Unicode
  - MIME type matching

Metaprogramming in Elixir

- Code that writes code at compile time, with macros
- Inspects and generates Elixir code representation
- "code representation" - Abstract Syntax Tree (AST)

What is it good for?

- Extending the language to your needs
- Optimizations
  - Performance
  - Boilerplate removal
- DSLs - Domain Specific Languages

Macros

- Carry out metaprogramming in Elixir
- Produce Elixir ASTs

```elixir
defmodule Notifier do

  def ping(pid) do
    if Process.alive?(pid) do
      Logger.debug "Sending ping!"
      send pid, :ping
    end
  end

end
```

Elixir's AST

- Represented as a tree of three element tuples
  - First element is an atom representing a function, or another tuple
  - Second element is metadata
  - Third element is the arguments to the function
- `quote` returns AST of any expression

```
quote do: 5 + 2

{:+, _, [5, 2]}
```

Function Metadata Arguments

```
quote do: 5 + 2

{:+, _, [5, 2]}
(+ 5 2)
```

```
quote do: (5 * 2) - 1 + 7
```

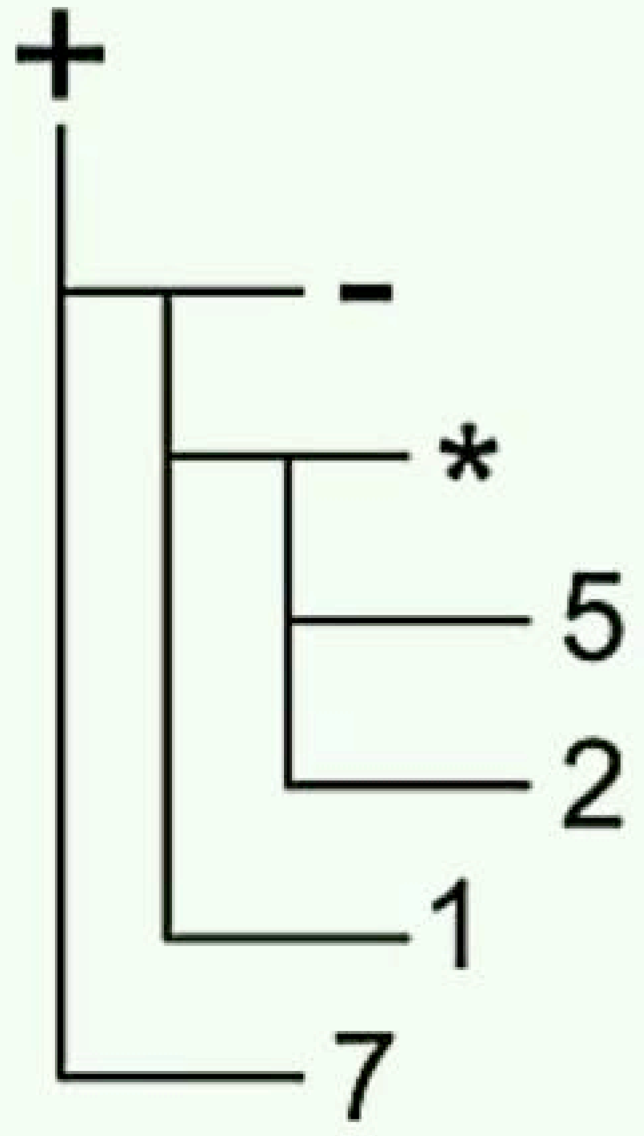
```
{:+, [],
 [{:-, [],
   [{:*, [], [5, 2]}, 1]}, 7]}
```

```
quote do: (5 * 2) - 1 + 7
```

```
{:+, [],
 [{:-, [],
   [{:*, [], [5, 2]}, 1]}, 7]}
```

```
(+ (- (* 5 2) 1) 7)
```

```elixir
quote do: (5 * 2) - 1 + 7
```



```
{:+, […],
  [{:-, […],
    [{:*, […], [5,
                2]},
    1]},
  7]}
```

```
defmodule MathTest do
  use ExUnit.Case

  test "maths" do
    assert 1 + 2 == 5
  end
end
```

```
quote do: 1 == 2
```

```
{:==, _, [1, 2]}
```

```
1) test maths (MathTest)
   iex:20
   Assertion with == failed
   code: 1 == 2
   lhs:  1
   rhs:  2
```

```
quote do: 1 == 2
{:==, _, [1, 2]}

quote do: 5 > 10
{:> _, [5, 10]}
```

```
{operator, _meta, [lhs, rhs]}
```

```elixir
defmodule Assertion do
  defmacro assert({op _, [lhs, rhs]}) do
    quote do
      Assertion.__perform__(unquote(op),
                            unquote(lhs),
                            unquote(rhs))
    end
  end

  def __perform__(:==, lhs, rhs) when lhs == rhs do
    IO.write "."
  end
  def __perform__(:==, lhs, rhs) do
    IO.puts """
    FAILURE:
      expected: #{inspect lhs}
      to equal: #{inspect lhs}
    """
  end
  # ...
end
```

```
defmodule MathTest do
  import Assertion

  def run do
    assert 5 > 2
    assert 1 + 1 == 2
    assert 5 == 6
  end
end

iex> MathTest.run()
..FAILURE:
  expected: 5
  to equal: 6
```

Macro Expansion

```
assert 5 > 2
assert 1 + 1 == 2
assert 5 == 6
```

$$\downarrow$$

```
Assertion.__perform__(:>, 5, 2)
Assertion.__perform__(:==, 1 + 1, 2)
Assertion.__perform__(:==, 5, 6)
```

```
unless user.banned? do
  deliver_message()
end
```

expand ↓

```
if(user.banned?) do
  nil
else
  deliver_message()
end
```

expand ↓

```
case user.banned? do
  x when x in [false, nil] ->
    deliver_message()
  _ ->
    nil
end
```

	Macro?
unless	✓
if	✓
case	✗

Language Extension

Parallel for comprehension

```
for user <- users do
  calculate_user_salary(user)
end

parallel(for user <- users do
  calculate_user_salary(user)
end)
```

```elixir
quote do
  for user <- users do
    calculate_user_salary(user)
  end
end

{:for, [],
 [{:<-, [], [{:user, [], Elixir},
             {:users, [], Elixir}]},
  [do: {:calculate_user_salary, [],
    [{:user, [], Elixir}]}]]}
```

Real-world usecases

Ecto

```
from u in User,
    where: u.age > ^min_age,
order_by: [asc: u.age],
   limit: 10,
  select: u
```

```elixir
defmodule User do
  use Ecto.Model

  schema "users" do
    field :age,  :integer
    field :name, :string
  end
end
```

Phoenix Framework

Routing DSL

```elixir
defmodule Router do
  scope "/" do
    get "/", PageController, :index
    get "/pages/:page", PageController, :show
    post "/files/", FilesController, :create
    resources "/messages", MessageController
  end
end
```

Pattern-matched Route Dispatch

```
defmodule Router do
  def match(conn, "GET", [])
  def match(conn, "GET", ["pages", page])
  def match(conn, "POST", ["files"])
  def match(conn, "GET", ["messages"])
  def match(conn, "GET", ["messages", "new"])
  def match(conn, "GET", ["messages", id, "edit"])
  def match(conn, "POST", ["messages"])
  def match(conn, "PUT", ["messages", id])
  def match(conn, "DELETE", ["messages", id])
end
```

Generated Route Helpers

```
get "/", PageController, :index
get "/pages/:page", PageController, :show
post "/files/", FilesController, :create
resources "/messages", MessageController
```

```
iex> Router.Helpers.page_path(Endpoint, :show, "about")
"/pages/about"

iex> Router.Helpers.page_url(Endpoint, :show, "about")
"http://example.com/pages/about"

iex> Router.Helpers.message_path(Endpoint, :show, 123)
"/messages/123"
```

Precompiled Views

```
defmodule Chat.MessageView do
  use Chat.Web, :view

  def render("show.html", %{msg: msg}) do
    "Showing a message! ..."
  end
end
```

```
iex> View.render(MessageView, "show.html", %{msg: msg})
"Showing a message!"
```

web/templates/message/index.html.eex

```
<h1>Listing Messages</h1>

<%= for msg <- @messages do %>
<tr>
  <td><%= msg.body %></td>
  <td><%= msg.room_id %></td>
  <td>
    <%= link "Show", to: msg_path(@conn, :show, msg) %>
    <%= link "Edit", to: msg_path(@conn, :edit, msg) %>
  </td>
</tr>
<% end %>
```

Advanced Code Generation

- Turn datasets into code
- Eliminate Boilerplate
- Optimize Performance

String.Unicode

```
irb> "José".upcase
"JOSé"

iex> String.upcase("José")
"JOSÉ"
```

/lib/elixir/unicode/UnicodeData.txt

```
0041;LATIN CAPITAL LETTER A;Lu;0;L;;;;;N;;;
0042;LATIN CAPITAL LETTER B;Lu;0;L;;;;;N;;;
0043;LATIN CAPITAL LETTER C;Lu;0;L;;;;;N;;;
0044;LATIN CAPITAL LETTER D;Lu;0;L;;;;;N;;;
1F680;ROCKET;So;0;ON;;;;;N;;;;;
1F681;HELICOPTER;So;0;ON;;;;;N;;;;;
1F682;STEAM LOCOMOTIVE;So;0;ON;;;;;N;;;;;
1F683;RAILWAY CAR;So;0;ON;;;;;N;;;;;
1F684;HIGH-SPEED TRAIN;So;0;ON;;;;;N;;;;;
```

27,000 lines of unicode mappings

```elixir
defmodule String.Unicode do

  data_path = Path.join(__DIR__, "UnicodeData.txt")

  {codes, whitespace} = Enum.reduce File.stream!(data_path),
    #...
  end

  def upcase(string), do: upcase(string, "")

  for {codepoint, upper, _lower, _title} <- codes,
    upper && upper != codepoint do
    defp upcase(unquote(codepoint) <> rest, acc) do
      upcase(rest, acc <> unquote(upper))
    end
  end

  defp upcase(<<char, rest :: binary>>, acc) do
    upcase(rest, <<acc::binary, char>>)
  end
  defp upcase("", acc), do: acc
  # ...
end
```

```
defp upcase("é" <> rest, acc) do
  upcase(rest, acc <> "É")
end

defp upcase("ć" <> rest, acc) do
  upcase(rest, acc <> "Ć")
end

defp upcase("🚀" <> rest, acc) do
  upcase(rest, acc <> "🚀")
end
```

MIME-Type conversion in 10 LOC

mimes.txt

```
application/javascript  .js
application/json              .json
image/jpeg          .jpeg, .jpg
video/jpeg          .jpgv
```

~ 685 mime types

MIME-Type conversion in 10 LOC

```elixir
defmodule MIME do
  for line <- File.stream!(Path.join([__DIR__, "mimes.txt"]), [], :line) do
    [type, rest] = line |> String.split("\t") |> Enum.map(&String.strip(&1))
    extensions = String.split(rest, ~r/,\s?/)

    def exts_from_type(unquote(type)), do: unquote(extensions)
    def type_from_ext(ext) when ext in unquote(extensions), do: unquote(type)
  end

  def exts_from_type(_type), do: []
  def type_from_ext(_ext), do: nil
  def valid_type?(type), do: exts_from_type(type) |> Enum.any?
end
```

```
iex> MIME.exts_from_type("image/jpeg")
[".jpeg", ".jpg"]

iex> MIME.type_from_ext(".jpg")
"image/jpeg"

iex> MIME.valid_type?("text/html")
true

iex> MIME.valid_type?("text/emoji")
false
```

HTML DSL

```
markup do
  div do
    h1 "Latest Post"
  end
  div class: "row" do
    p post.body
    if post.published? do
      span "Posted #{post.publish_date}"
    end
  end
end
"<div><h1>Latest Post</h1></div>
<div class=\"row\"><p></p></div>"
```

```elixir
defmodule Hub do
  "https://api.github.com/users/chrismccord/repos"
  |> HTTPotion.get(["User-Agent": "Elixir"])
  |> Map.get(:body)
  |> Poison.decode!()
  |> Enum.each(fn repo ->
    def unquote(String.to_atom(repo["name"]))() do
      unquote(Macro.escape(repo))
    end
  end)

  def go(repo) do
    url = apply(__MODULE__, repo, [])["html_url"]
    System.cmd("open", [url])
  end
end
```

Recap

- ~~Don't write macros~~
- Use macros responsibly
- Don't be afraid to be a little irresponsible why you're learning
- Extend the language
- Have fun.

@chris_mccord

#elixir-lang freenode

www.elixir-lang.org

https://pragprog.com/book/cmelixir/metaprogramming-elixir