eli

Maps

```
%{ key => value, key => value }
value = map[key]
value = map.key (if key is an atom)
newmap = %{ oldmap | key => newval}
Dict.put_new/3 to add a key
```

Protocols

```
defprotocol module.name do
 @moduledoc description
  @only [list of types] (optional)
 def name(parms)
end

defimpl mod.name, for: type do
   @moduledoc description
   def name(type, value) do
      expr
   end
end
```

Allowed types:
Any Atom BitString Function List Number PID Port Record Reference Tuple

Regexp

~r{pattern}opts

f	match beg of ml string
g	use named groups
i	case insensitive
m	^ and $ match each line in ml
r	reluctant (not greedy)
s	. matches newline
u	unicode patterns
x	ignore whitespace and comments

Processes

```
pid = spawn(anon_function)
pid = spawn(mod, func, args)
(also spawn_link)

receive do
 {sender, msg, … } ->
   send sender { :ok, value }
 after timeout ->
   ...
end
```

Predefined Names

__MODULE__ __FILE__ __DIR__ __ENV__
__CALLER__ (macros only)

Pipelines

```
expr |> f1 |> f2(a,b) |> f3(c)
```

(same as)

```
f3(f2(f1(expr), a, b), c)
```

Control Flow

```
if expr do
   exp
else
   exp
end

case expr do
 match [guard] -> exp
 match [guard]  -> exp
 …
end
```

```
unless expr do
   exp
else
   exp
end

cond do
   bool -> exp
   bool -> exp
end
```

Metaprogramming

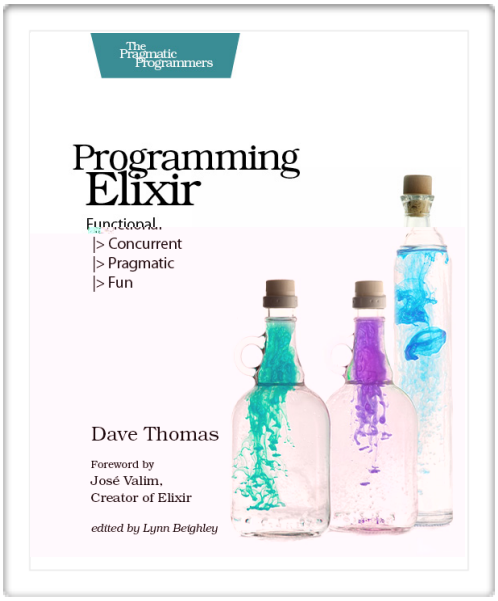
```
defmacro macroname(parms) do
          parms are quoted args
          return quoted code which
          is inserted at call site
end

quote do:      returns internal rep.
quote bind_quoted:    name    name
do:  ...

unquote do:  … only inside quote, injects
code fragment without evaluation
```

Sigils

~*type*{ *content* }

Delimiter: { }, [ ], ( ), / /, | |, " ", or ' '

%S	string (no interpolation)
%s	string (with interpolation)
%C	character list (no interpolation)
%c	character list (with interpolation)
%R	regexp
%r	regexp w/interpolation
%W	words (white space delim)
%w	words w/interpolation



pragprog.com/books/elixir

Structs

```
defmodule Name do
   defstruct field: default, …
end

%Name{field: value, field: value, …}

new_struct = %{ var | field: new_value }
```