Cheatography

Elixir Cheat Sheet

by liebus via cheatography.com/180711/cs/37582/

Hello world

```
# hello.exs
defmodule Greeter do
   def greet( name) do
        message = " Hello, " <>
name <> " !"
        IO.puts message
   end
end
Greete r.g ree t("w orl d")
```

elixir hello.exs
Hello, world!

Pattern matching

```
user = %{name: "Tom", age: 23}
%{name: username} = user
```

This sets username to "Tom".

If

```
if false do
    "This will never be seen"
else
    "This will"
end
```

Types (Primitives)

nil	Nil/null
true / false	Boolean
?a	Integer (ASCII)
23	Integer
3.14	Float
'hello'	Charlist
<<2, 3>>	Binary
"hello"	Binary string
:hello	Atom
[a, b]	List
{a, b}	Tuple
%{a: "hello"}	Map
%MyStruct{a: "hello"}	Struct
fn -> ... end	Function

Modules (Importing)

require Redux # compiles a module
import Redux # compiles, and you can use without the `Redux.` prefix
use Redux # compiles, and runs Redux._-_using__/1
use Redux, async: true
import Redux, only: [duplicate: 2]
import Redux, only: :functions
import Redux, only: :macros
import Foo.{Bar, Baz}

Maps

```
user = %{
    name: " Joh n",
    city: " Mel bou rne "
}
```

IO.puts "Hello, " <> user.name

Variables

```
age = 23
```

Pattern matching in functions

```
ef greet(%{name: username}) do
    IO.puts " Hello, " <>
username
end
user = %{name: " Tom ", age: 23}
```

Pattern matching works in function parameters too.

Case

```
case {1, 2, 3} do
   {4, 5, 6} ->
        "This clause won't
match"
   {1, x, 3} ->
        "This will match and
bind x to 2"
   _ ->
      "This will match any
value"
end
```

With

```
case Users.create_user(attrs) do
    {:ok, user} -> ...
    {:e rror, changeset} -> ...
end
with {:ok, user} <- Users.c re -
ate _us er( attrs) do
   ...
else
    {:e rror, changeset} -> ...
end
```

Lists

```
users = [ "Tom", "Dick", "Harry"
]

Enum.m ap( users, fn user ->
    IO.puts " Hello " <> user
end)
```

Piping

```
source
|> transf orm (:h ello)
|> print()

# Same as:
print( tra nsf orm (so urce,
:hello))
```

These two are equivalent.

Cond

```
cond do
   1 + 1 == 3 ->
        "I will never be seen"
   2 * 5 == 12 ->
        "Me neithe r"
   true ->
        "But I will (this is
essent ially an else)"
end
```

Errors

```
try do
    thr ow( :hello)
catch
    message -> "Got #{mess -
age }."
after
    IO.p ut s("I'm the after
clause.")
end
```

Type checks

is_atom/1	is_binary/1	is_nil/1
is_bitstring/1	is_list/1	is_number/1
is_boolean/1	is_map/1	is_pid/1
is_function/1	is_tuple/1	is_port/1
is_function/2		is_reference/1
is_integer/1		
is_float/1		

Operators

left != right # equal
left !== right # match
left ++ right # concat lists
left <> right # concat string/binary
left =~ right # regexp

By **liebus**
cheatography.com/liebus/

Not published yet.
Last updated 9th March, 2023.
Page 2 of 2.