배우기

Elixir Language

#elixir

1: Elixir

Elixir   .

Elixir   Erlang VM .

0.9	2013-05-23
1.0	2014-09-18
1.1	2015-09-28
1.2	2016-01-03
1.3	2016-06-21
1.4	2017-01-05

Examples

.

Elixir `erlang`   erlang `BEAM` (java `JVM`) .

elixir   : `iex` `elixir` .

`hello.exs` .

```
IO.puts "Hello world!"
```

Elixir .

```
$ elixir hello.exs
```

:

    !

`Elixir`   . , Elixir BEAM   .

elixir () `iex`   .   .

```
Interactive Elixir (1.3.4) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

`hello world`   :

```
iex(1)> IO.puts "hello, world"
hello, world
:ok
iex(2)>
```

iex . helloworld.ex

```
defmodule Hello do
   def sample do
       IO.puts "Hello World!"
   end
end
```

iex iex .

```
iex(1)> c("helloworld.ex")
[Hello]
iex(2)> Hello.sample
Hello World!
```

IEx Hello World

IEx (Interactive Elixir)   .

Linux Mac bash iex Enter .

```
$ iex
```

Windows   .

```
C:\ iex.bat
```

IEx REPL (Read, Evaluate, Print, Loop)   .

```
iex(1)> "Hello World"
"Hello World"
```

IEx REPL   .

```
$ iex script.exs
```

script.exs .   .

Elixir   : https://riptutorial.com/ko/elixir/topic/954/elixir--

2: Doctests

Examples

@doc   .

```
# myproject/lib/my_module.exs

defmodule MyModule do
  @doc """
  Given a number, returns `true` if the number is even, otherwise `false`.

  ## Example
    iex> MyModule.even?(2)
    true
    iex> MyModule.even?(3)
    false
  """
  def even?(number) do
    rem(number, 2) == 0
  end
end
```

.

```
# myproject/test/doc_test.exs

defmodule DocTest do
  use ExUnit.Case
  doctest MyModule
end
```

mix test   .

doctest HTML

markdown   .

1 / doctest   doctest ("" "").   4 HTML   .

2 / ,   "mix docs"   doc HTML .
$> mix docs

doctests

'...>'   doctest .

```
iex> Foo.Bar.somethingConditional("baz")
...>   |> case do
...>       {:ok, _} -> true
...>       {:error, _} -> false
```

```
...>      end
true
```

Doctests : https://riptutorial.com/ko/elixir/topic/2708/doctests

3: ExDoc

Examples

`@doc` `@moduledoc` `HTML` `ex_doc` markdown . ExDoc Earmark , Pandoc , Hoedown Cmark `mix.exs` `mix.exs` .

```
# config/mix.exs

def deps do
  [{:ex_doc, "~> 0.11", only: :dev},
   {:earmark, "~> 0.1", only: :dev}]
end
```

Markdown Markdown .

Elixir `@doc` `@moduledoc` Markdown .

`mix docs` .

ExDoc .

```
def project do
  [app: :my_app,
   version: "0.1.0-dev",
   name: "My App",
   source_url: "https://github.com/USER/APP",
   homepage_url: "http://YOUR_PROJECT_HOMEPAGE",
   deps: deps(),
   docs: [logo: "path/to/logo.png",
          output: "docs",
          main: "README",
          extra_section: "GUIDES",
          extras: ["README.md", "CONTRIBUTING.md"]]]
end
```

`mix help docs` .

ExDoc : https://riptutorial.com/ko/elixir/topic/3582/exdoc

4: ExUnit

Examples

`assert_raise` `.` `assert_raise` Exception `.`

```
test "invalid block size" do
  assert_raise(MerkleTree.ArgumentError, (fn() -> MerkleTree.new ["a", "b", "c"] end))
end
```

`assert_raise` `.`

ExUnit : https://riptutorial.com/ko/elixir/topic/3583/exunit

5: IEx

Examples

`recompile` .

```
iex(1)> recompile
Compiling 1 file (.ex)
:ok
```

`h '.

```
iex(1)> h List.last

                              def last(list)

Returns the last element in list or nil if list is empty.

Examples

| iex> List.last([])
| nil
|
| iex> List.last([1])
| 1
|
| iex> List.last([1, 2, 3])
| 3
```

`v` .

```
iex(1)> 1 + 1
2
iex(2)> v
2
iex(3)> 1 + v
3
```

: `v` .

'v' .

```
iex(1)> a = 10
10
iex(2)> b = 20
20
iex(3)> a + b
30
```

.

```
iex(4)> v(3)
30
```

.

```
iex(5)> v(-1) # Retrieves value of row (5-1) -> 4
30
iex(6)> v(-5) # Retrieves value of row (5-4) -> 1
10
```

.

```
iex(7)> v(2) * 4
80
```

IEx .

```
iex(7)> v(100)
** (RuntimeError) v(100) is out of bounds
    (iex) lib/iex/history.ex:121: IEx.History.nth/2
    (iex) lib/iex/helpers.ex:357: IEx.Helpers.v/1
```

IEx

1. Ctrl + C, Ctrl + C .

```
iex(1)>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
```

2. Ctrl+ \ .

'i'

```
iex(1)> i :ok
Term
  :ok
Data type
  Atom
Reference modules
  Atom
iex(2)> x = "mystring"
"mystring"
iex(3)> i x
Term
  "mystring"
Data type
  BitString
Byte size
  8
Description
  This is a string: a UTF-8 encoded binary. It's printed surrounded by
  "double quotes" because all UTF-8 encoded codepoints in it are printable.
```

```
Raw representation
  <<109, 121, 115, 116, 114, 105, 110, 103>>
Reference modules
  String, :binary
```

PID

PID

```
iex(1)> self()
#PID<0.138.0>
iex(2)> pid("0.138.0")
#PID<0.138.0>
iex(3)> pid(0, 138, 0)
#PID<0.138.0>
```

IEx .

.iex.exs .iex.exs .iex.exs .

```
alias App.{User, Repo}
```

IEx   .

erlang-history Erlang IEx .

```
git clone git@github.com:ferd/erlang-history.git
cd erlang-history
sudo make install
```

IEx   .

...

.

```
iex(2)> receive do _ -> :stuck end
```

Ctrl-g . :

User switch command

.

- k ( )
- s ( )
- c ( )

Erlang :

```
Eshell V8.0.2  (abort with ^G)
1>
```

Elixir   .

```
'Elixir.IEx.CLI':local_start().
```

(!)

Elixir   .

```
Interactive Elixir (1.3.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> "I'm back"
"I'm back"
iex(2)>
```

" **#iex:break** , " "more for input" **#iex:break**   ( ) .

```
iex(1)> "Hello, "world"
...(1)>
...(1)> #iex:break
** (TokenMissingError) iex:1: incomplete expression

iex(1)>
```

"more-for-more-input"   .

IEX , IEX (...) `iex` .

IEx   `#iex:break` . IEX `TokenMissingError` "" ,   .

```
iex:1> "foo"
"foo"
iex:2> "bar
...:2> #iex:break
** (TokenMissingError) iex:2: incomplete expression
```

IEx   .

IEx

;   IEx `c/1` .

```
iex(1)> c "lib/utils.ex"
iex(2)> Utils.some_method
```

IEx   .

.

```
iex(3)> c "/path/to/my/script.exs"
```

```
Called from within the script!
```

6: IEx

IEx Elixir . Elixir `iex iex` . IEx `h` `h List.foldr` . : `h List.foldr`

Examples

Elixir

Elixir .

```
h Elixir.[TAB]
```

[TAB] . . `List`

```
h List.[TAB]
```

IEx : https://riptutorial.com/ko/elixir/topic/10780/iex--

7: IO.inspect

`IO.inspect`   .   .

Elixir 1.4.0 `IO.inspect label`   .

Elixir 1.4+   .

Examples

```
url
  |> IO.inspect
  |> HTTPoison.get!
  |> IO.inspect
  |> Map.get(:body)
  |> IO.inspect
  |> Poison.decode!
  |> IO.inspect
```

:

```
"https://jsonplaceholder.typicode.com/posts/1"
%HTTPoison.Response{body: "{\n  \"userId\": 1,\n  \"id\": 1,\n  \"title\": \"sunt aut facere
repellat provident occaecati excepturi optio reprehenderit\",\n  \"body\": \"quia et
suscipit\\nsuscipit recusandae consequuntur expedita et cum\\nreprehenderit molestiae ut ut
quas totam\\nnostrum rerum est autem sunt rem eveniet architecto\"\n}",
 headers: [{"Date", "Thu, 05 Jan 2017 14:29:59 GMT"},
  {"Content-Type", "application/json; charset=utf-8"},
  {"Content-Length", "292"}, {"Connection", "keep-alive"},
  {"Set-Cookie",
   "__cfduid=d56d1be0a544fcbdbb262fee9477600c51483626599; expires=Fri, 05-Jan-18 14:29:59 GMT;
path=/; domain=.typicode.com; HttpOnly"},
  {"X-Powered-By", "Express"}, {"Vary", "Origin, Accept-Encoding"},
  {"Access-Control-Allow-Credentials", "true"},
  {"Cache-Control", "public, max-age=14400"}, {"Pragma", "no-cache"},
  {"Expires", "Thu, 05 Jan 2017 18:29:59 GMT"},
  {"X-Content-Type-Options", "nosniff"},
  {"Etag", "W/\"124-yv65LoT2uMHrpn06wNpAcQ\""}, {"Via", "1.1 vegur"},
  {"CF-Cache-Status", "HIT"}, {"Server", "cloudflare-nginx"},
  {"CF-RAY", "31c7a025e94e2d41-TXL"}], status_code: 200}
"{\n  \"userId\": 1,\n  \"id\": 1,\n  \"title\": \"sunt aut facere repellat provident
occaecati excepturi optio reprehenderit\",\n  \"body\": \"quia et suscipit\\nsuscipit
recusandae consequuntur expedita et cum\\nreprehenderit molestiae ut ut quas totam\\nnostrum
rerum est autem sunt rem eveniet architecto\"\n}"
%{"body" => "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit
molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto",
  "id" => 1,
  "title" => "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "userId" => 1}
```

`label`   .

```
url
```

```
  |> IO.inspect(label: "url")
  |> HTTPoison.get!
  |> IO.inspect(label: "raw http resonse")
  |> Map.get(:body)
  |> IO.inspect(label: "raw body")
  |> Poison.decode!
  |> IO.inspect(label: "parsed body")

url: "https://jsonplaceholder.typicode.com/posts/1"
raw http resonse: %HTTPoison.Response{body: "{\n  \"userId\": 1,\n  \"id\": 1,\n  \"title\":
\"sunt aut facere repellat provident occaecati excepturi optio reprehenderit\",\n  \"body\":
\"quia et suscipit\\nsuscipit recusandae consequuntur expedita et cum\\nreprehenderit
molestiae ut ut quas totam\\nnostrum rerum est autem sunt rem eveniet architecto\"\n}",
 headers: [{"Date", "Thu, 05 Jan 2017 14:33:06 GMT"},
  {"Content-Type", "application/json; charset=utf-8"},
  {"Content-Length", "292"}, {"Connection", "keep-alive"},
  {"Set-Cookie",
   "__cfduid=d22d817e48828169296605d27270af7e81483626786; expires=Fri, 05-Jan-18 14:33:06 GMT;
path=/; domain=.typicode.com; HttpOnly"},
  {"X-Powered-By", "Express"}, {"Vary", "Origin, Accept-Encoding"},
  {"Access-Control-Allow-Credentials", "true"},
  {"Cache-Control", "public, max-age=14400"}, {"Pragma", "no-cache"},
  {"Expires", "Thu, 05 Jan 2017 18:33:06 GMT"},
  {"X-Content-Type-Options", "nosniff"},
  {"Etag", "W/\"124-yv65LoT2uMHrpn06wNpAcQ\""}, {"Via", "1.1 vegur"},
  {"CF-Cache-Status", "HIT"}, {"Server", "cloudflare-nginx"},
  {"CF-RAY", "31c7a4b8ae042d77-TXL"}], status_code: 200}
raw body: "{\n  \"userId\": 1,\n  \"id\": 1,\n  \"title\": \"sunt aut facere repellat
provident occaecati excepturi optio reprehenderit\",\n  \"body\": \"quia et
suscipit\\nsuscipit recusandae consequuntur expedita et cum\\nreprehenderit molestiae ut ut
quas totam\\nnostrum rerum est autem sunt rem eveniet architecto\"\n}"
parsed body: %{"body" => "quia et suscipit\nsuscipit recusandae consequuntur expedita et
cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet
architecto",
  "id" => 1,
  "title" => "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "userId" => 1}
```

IO.inspect   : https://riptutorial.com/ko/elixir/topic/8725/io-inspect-----

8:

Examples

Elixir   .   .

```
iex(1)> my_func = fn x -> x * 2 end
#Function<6.52032458/1 in :erl_eval.expr/5>
```

.

```
fn args -> output end
```

.

```
iex(2)> my_func = fn (x, y) -> x*y end
#Function<12.52032458/2 in :erl_eval.expr/5>
```

.   .

```
iex(3)>my_func.(7, 5)
35
```

.

```
iex(4)> my_func2 = fn -> IO.puts "hello there" end
iex(5)> my_func2.()
hello there
:ok
```

& . , :

```
iex(5)> my_func = fn (x) -> x*x*x end
```

:

```
iex(6)> my_func = &(&1*&1*&1)
```

1 .

```
iex(7)> my_func = fn (x, y) -> x + y end

iex(8)> my_func = &(&1 + &2)    # &1 stands for x and &2 stands for y

iex(9)> my_func.(4, 5)
9
```

___

.

```
my_func = fn
  param1 -> do_this
  param2 -> do_that
end
```

Elixir   .

-     'options'   :

```
def myfunc(arg1, opts \\ []) do
  # Function body
end
```

.

```
iex> myfunc "hello", pizza: true, soda: false
```

.

```
iex> myfunc("hello", [pizza: true, soda: false])
```

opts.pizza opts.soda   .
: opts[:pizza] opts[:soda] .

```
defmodule Math do
    # one way
    def add(a, b) do
        a + b
    end

    # another way
    def subtract(a, b), do: a - b
end

iex> Math.add(2, 3)
5
:ok
iex> Math.subtract(5, 2)
3
:ok
```

```
defmodule Math do
    def sum(a, b) do
        add(a, b)
    end

    # Private Function
    defp add(a, b) do
        a + b
    end
```

```
end

iex> Math.add(2, 3)
** (UndefinedFunctionError) undefined function Math.add/2
Math.add(3, 4)
iex> Math.sum(2, 3)
5
```

Elixir   .

```
defmodule Math do
    def factorial(0): do: 1
    def factorial(n): do: n * factorial(n - 1)
end
```

`factorial(0)` . ( ). Elixir .   . `factorial(0)` `factorial(n)` ,

Guard   . `if cond`   .   .

.

```
defmodule Math do
    def factorial(0), do: 1
    def factorial(n) when n > 0: do: n * factorial(n - 1)
end
```

`0` ( ). `0`   .

. `when n > 0` ., `n` `0` .,   .

(guard) `FunctionClauseError` .   .

`FunctionClauseError` . `-1` `0` `" "` `-1`   .

. `param \\ value` :

```
defmodule Example do
    def func(p1, p2 \\ 2) do
        IO.inspect [p1, p2]
    end
end

Example.func("a")    # => ["a", 2]
Example.func("b", 4) # => ["b", 4]
```

`&` .   .

```
Enum.map(list, fn(x) -> String.capitalize(x) end)
```

`&`   .

```
Enum.map(list, &String.capitalize(&1))
```

(: `&String.capitalize/1` :

```
defmodule Bob do
  def say(message, f \\ &String.capitalize/1) do
    f.(message)
  end
end
```

: https://riptutorial.com/ko/elixir/topic/2442/

(: `&String.capitalize/1` :

```
defmodule Bob do
  def say(message, f \\ &String.capitalize/1) do
    f.(message)
```

9:

- []
- [1, 2, 3, 4]
- [1,2] ++ [3, 4] # - [1,2,3,4]
- hd ([1, 2, 3, 4]) # -> 1
- t1 ([1, 2, 3, 4]) # -> [2,3,4]
- [|]
- [1 | [2, 3, 4]] # -> [1,2,3,4]
- [1 | [2 | [3 | [4 | []]]]] -> [1,2,3,4]
- 'hello'= [? h,? e,? l,? l,? o]
- keyword_list = [a : 123, b : 456, c : 789]
- keyword_list [: a] # -> 123

Examples

.

```
keyword_list = [{:a, 123}, {:b, 456}, {:c, 789}]
```

.

```
keyword_list = [a: 123, b: 456, c: 789]
```

-    .

.

```
iex> keyword_list[:b]
456
```

.

```
defmodule TaskRunner do
  def run_tasks(tasks) do
    # Call a function for each item in the keyword list.
    # Use pattern matching on each {:key, value} tuple in the keyword list
    Enum.each(tasks, fn
      {:delete, x} ->
        IO.puts("Deleting record " <> to_string(x) <> "...")
      {:add, value} ->
        IO.puts("Adding record \"" <> value <> "\"...")
      {:update, {x, value}} ->
        IO.puts("Setting record " <> to_string(x) <> " to \"" <> value <> "\"...")
    end)
  end
end
```

.

```
iex> tasks = [
...>   add: "foo",
...>   add: "bar",
...>   add: "test",
...>   delete: 2,
...>   update: {1, "asdf"}
...> ]

iex> TaskRunner.run_tasks(tasks)
Adding record "foo"...
Adding record "bar"...
Adding record "test"...
Deleting record 2...
Setting record 1 to "asdf"...
```

Elixir "". Erlang "char list" Erlang   Elixir char .

" ,   ' :

```
string = "Hello!"
char_list = 'Hello!'
```

.

```
'hello' = [104, 101, 108, 108, 111]
```

to_charlist/1   char   .

```
iex> to_charlist("hello")
'hello'
```

to_string/1   :

```
iex> to_string('hello')
"hello"
```

Erlang   Elixir :

```
iex> :os.getenv |> hd |> to_string
"PATH=/usr/local/bin:/usr/bin:/bin"
```

. ,   .   .

Cons "" "" "" "" .

A | ()   . 1 2 .

```
[1 | 2]
```

.

```
[1, 2, 3, 4] = [1 | [2 | [3 | [4 | []]]]]
```

[]   .

Elixir `[head | tail]` , head   tail   .

```
iex> [head | tail] = [1, 2, 3, 4]
[1, 2, 3, 4]
iex> head
1
iex> tail
[2, 3, 4]
```

`[head | tail]`   .

```
def sum([]), do: 0

def sum([head | tail]) do
  head + sum(tail)
end
```

map   ,   . Elixir `map/2` Enum .

```
iex> Enum.map([1, 2, 3, 4], fn(x) -> x + 1 end)
[2, 3, 4, 5]
```

:

```
iex> Enum.map([1, 2, 3, 4], &(&1 + 1))
[2, 3, 4, 5]
```

:

```
iex> Enum.map([1, 2, 3, 4], &to_string/1)
["1", "2", "3", "4"]
```

:

```
iex> [1, 2, 3, 4]
...> |> Enum.map(&to_string/1)
...> |> Enum.map(&("Chapter " <> &1))
["Chapter 1", "Chapter 2", "Chapter 3", "Chapter 4"]
```

. `Enum List` List Comprehensions .

List Comprehensions   .

   •

```
iex(1)> for value <- [1, 2, 3], do: value + 1
[2, 3, 4]
```

- , guard   when .

```
iex(2)> odd? = fn x -> rem(x, 2) == 1 end
iex(3)> for value <- [1, 2, 3], odd?.(value), do: value
[1, 3]
```

- into :

```
iex(4)> for value <- [1, 2, 3], into: %{}, do: {value, value + 1}
%{1 => 2, 2=>3, 3 => 4}
```

──────

```
iex(5)> for value <- [1, 2, 3], odd?.(value), into: %{}, do: {value, value * value}
%{1 => 1, 3 => 9}
```

──────

:

- for..do   into   ..
- .
- .
- .
- guard   for do into. .

. Enum List   .

```
iex> [1, 2, 3] -- [1, 3]
[2]
```

--   .

in   .

```
iex> 2 in [1, 2, 3]
true
iex> "bob" in [1, 2, 3]
false
```

Enum.chunk/2   Map.new/2 .

```
[1, 2, 3, 4, 5, 6]
|> Enum.chunk(2)
|> Map.new(fn [k, v] -> {k, v} end)
```

:

---

https://riptutorial.com/ko/home

```
%{1 => 2, 3 => 4, 5 => 6}
```

: https://riptutorial.com/ko/elixir/topic/1279/

10:

Examples

Elixir . 10 , 2 , 8 16 .

```
iex> x = 291
291

iex> x = 0b100100011
291

iex> x = 0o443
291

iex> x = 0x123
291
```

Elixir bignum .

IEEE-754 .

```
iex> x = 6.8
6.8

iex> x = 1.23e-11
1.23e-11
```

Elixir float .

```
iex> 1 + 1
2

iex> 1.0 + 1.0
2.0
```

. .

Elixir .

```
iex> 10 / 2
5.0
```

.

```
iex> 40.0 + 2
42.0

iex> 10 - 5.0
5.0
```

```
iex> 3 * 3.0
9.0
```

div/2   .

```
iex> div(10, 2)
5
```

.   .   .

```
:atom   # that's how we define an atom
```

.   .

```
iex(1)> a = :atom
:atom

iex(2)> b = :atom
:atom

iex(3)> a == b
true

iex(4)> a === b
true
```

true false .

```
iex(1)> true == :true
true

iex(2)> true === :true
true
```

.   ., ( ) - .

Kernel.SpecialForms << >> .

Elixir   .

"<<" ">>" . ",   .   (8).

```
<<222,173,190, 239>> # 0xDEADBEEF
```

.

```
iex> <<0, "foo">>
<<0, 102, 111, 111>>
```

"" ""   .

  •

-
-

"." .

```
<<102::integer-native>>
<<102::native-integer>> # Same as above
<<102::unsigned-big-integer>>
<<102::unsigned-big-integer-size(8)>>
<<102::unsigned-big-integer-8>> # Same as above
<<102::8-integer-big-unsigned>>
<<-102::signed-little-float-64>> # -102 as a little-endian Float64
<<-102::native-little-float-64>> # -102 as a Float64 for the current machine
```

.

-
-
- ( )
-
-
- ( )
- utf8
- utf16
- utf32

"   " .

- () 1
- 1
- 2 8

: https://riptutorial.com/ko/elixir/topic/1774/

11:

Examples

```
iex(bob@127.0.0.1)> Node.list
[:"frank@127.0.0.1"]
```

.

```
>iex --name bob@127.0.0.1
iex(bob@127.0.0.1)>
>iex --name frank@127.0.0.1
iex(frank@127.0.0.1)>
```

.

```
iex(bob@127.0.0.1)> Node.connect :"frank@127.0.0.1"
true
```

.

```
iex(bob@127.0.0.1)> Node.list
[:"frank@127.0.0.1"]
iex(frank@127.0.0.1)> Node.list
[:"bob@127.0.0.1"]
```

.

```
iex(bob@127.0.0.1)> greet = fn() -> IO.puts("Hello from #{inspect(Node.self)}") end
iex(bob@127.0.0.1)> Node.spawn(:"frank@127.0.0.1", greet)
#PID<9007.74.0>
Hello from :"frank@127.0.0.1"
:ok
```

IP   :

```
$ iex --name foo@10.238.82.82 --cookie chocolate
iex(foo@10.238.82.82)> Node.ping :"bar@10.238.82.85"
:pong
iex(foo@10.238.82.82)> Node.list
[:"bar@10.238.82.85"]
```

IP   .

```
$ iex --name bar@10.238.82.85 --cookie chocolate
iex(bar@10.238.82.85)> Node.list
[:"foo@10.238.82.82"]
```

: https://riptutorial.com/ko/elixir/topic/2065/

12:

- [| tail] = [1, 2, 3, true] #   . head 1, tail [2, 3, true] .

- % {d : val} = % {d : 1, e : true} # val 1 . lhs d % {: d => _}   d   (   )

.

.   .

.

Examples

```
a = [1, 2, 3, true]
```

. Id est head (List.hd / 1)   tail (List.tail / 1)   cons .

```
List.hd(a) = 1
List.tl(a) = [2, 3, true]
```

```
b = {:ok, 1, 2}
```

.   .

13:

Examples

`IEx.pry/0` .

1. `require IEx` `require IEx` .
2.
3. `IEx.pry`

(`:iex -S mix`).

`IEx.pry/0` `IEx.pry/0` . .

`respawn` .

```
require IEx;

defmodule Example do
  def double_sum(x, y) do
    IEx.pry
    hard_work(x, y)
  end

  defp hard_work(x, y) do
    2 * (x + y)
  end
end
```

IO.inspect / 1

elixir `IO.inspect` / 1 .

```
defmodule MyModule do
  def myfunction(argument_1, argument_2) do
    IO.inspect(argument_1)
    IO.inspect(argument_2)
  end
end
```

argument_1 argument_2 . `IO.inspect/1` .

```
do_something(a, b)
|> do_something_else(c)

# can be adorned with IO.inspect, with no change in functionality:

do_something(IO.inspect(a), IO.inspect(b))
|> IO.inspect
```

```
do_something(IO.inspect(c))
```

```
defmodule Demo do
  def foo do
    1..10
    |> Enum.map(&(&1 * &1))            |> p
    |> Enum.filter(&rem(&1, 2) == 0)  |> p
    |> Enum.take(3)                   |> p
  end

  defp p(e) do
    require Logger
    Logger.debug inspect e, limit: :infinity
    e
  end
end
```

```
iex(1)> Demo.foo

23:23:55.171 [debug] [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

23:23:55.171 [debug] [4, 16, 36, 64, 100]

23:23:55.171 [debug] [4, 16, 36]

[4, 16, 36]
```

.

```
defmodule Demo do
  def foo do
    1..10
    |> Enum.map(&(&1 * &1))
    |> Enum.filter(&rem(&1, 2) == 0) |> pry
    |> Enum.take(3)
  end

  defp pry(e) do
    require IEx
    IEx.pry
    e
  end
end
```

```
iex(1)> Demo.foo
Request to pry #PID<0.117.0> at lib/demo.ex:11

      def pry(e) do
        require IEx
        IEx.pry
        e
      end

Allow? [Yn] Y
```

```
Interactive Elixir (1.3.2) – press Ctrl+C to exit (type h() ENTER for help)
pry(1)> e
[4, 16, 36, 64, 100]
pry(2)> respawn

Interactive Elixir (1.3.2) – press Ctrl+C to exit (type h() ENTER for help)
[4, 16, 36]
iex(1)>
```

: https://riptutorial.com/ko/elixir/topic/2719/-

14:

Examples

```
defmodule ATest do
  use ExUnit.Case

  [{1, 2, 3}, {10, 20, 40}, {100, 200, 300}]
  |> Enum.each(fn {a, b, c} ->
    test "#{a} + #{b} = #{c}" do
      assert unquote(a) + unquote(b) = unquote(c)
    end
  end)
end
```

:

```
.

  1) test 10 + 20 = 40 (Test.Test)
     test.exs:6
     match (=) failed
     code: 10 + 20 = 40
     rhs:  40
     stacktrace:
       test.exs:7


.

Finished in 0.1 seconds (0.1s on load, 0.00s on tests)
3 tests, 1 failure
```

: https://riptutorial.com/ko/elixir/topic/4069/-

15:

Elixir IO String   :"Elixir.ModuleName" :"Elixir.ModuleName" .

```
iex(1)> is_atom(IO)
true
iex(2)> IO == :"Elixir.IO"
true
```

Examples

__info__/1   .

- :functions -   .
- :macros -   .

Kernel   .

```
iex> Kernel.__info__ :functions
[!=: 2, !==: 2, *: 2, +: 1, +: 2, ++: 2, -: 1, -: 2, --: 2, /: 2, <: 2, <=: 2,
 ==: 2, ===: 2, =~: 2, >: 2, >=: 2, abs: 1, apply: 2, apply: 3, binary_part: 3,
 bit_size: 1, byte_size: 1, div: 2, elem: 2, exit: 1, function_exported?: 3,
 get_and_update_in: 3, get_in: 2, hd: 1, inspect: 1, inspect: 2, is_atom: 1,
 is_binary: 1, is_bitstring: 1, is_boolean: 1, is_float: 1, is_function: 1,
 is_function: 2, is_integer: 1, is_list: 1, is_map: 1, is_number: 1, is_pid: 1,
 is_port: 1, is_reference: 1, is_tuple: 1, length: 1, macro_exported?: 3,
 make_ref: 0, ...]
```

Kernel   .

alias , import , use require   4   require .

alias   ( ).

```
defmodule MyModule do
  # Will make this module available as `CoolFunctions`
  alias MyOtherModule.CoolFunctions
  # Or you can specify the name to use
  alias MyOtherModule.CoolFunctions, as: CoolFuncs
end
```

import   :

```
defmodule MyModule do
  import Enum
  def do_things(some_list) do
    # No need for the `Enum.` prefix
    join(some_list, " ")
  end
end
```

use . .

require .

defdelegate .

```
defmodule Math do
  defdelegate pi, to: :math
end
```

```
iex> Math.pi
3.141592653589793
```

:

16:

Elixir `String` UTF-8 .

Examples

Kernel.inspect .

```
iex> Kernel.inspect(1)
"1"
iex> Kernel.inspect(4.2)
"4.2"
iex> Kernel.inspect %{pi: 3.14, name: "Yos"}
"%{pi: 3.14, name: \"Yos\"}"
```

```
iex> my_string = "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
iex> String.slice my_string, 6..10
"ipsum"
```

```
iex> String.split("Elixir, Antidote, Panacea", ",")
["Elixir", "Antidote", "Panacea"]
```

```
iex(1)> name = "John"
"John"
iex(2)> greeting = "Hello, #{name}"
"Hello, John"
iex(3)> num = 15
15
iex(4)> results = "#{num} item(s) found."
"15 item(s) found."
```

.

```
iex(1)> String.contains? "elixir of life", "of"
true
iex(2)> String.contains? "elixir of life", ["life", "death"]
true
iex(3)> String.contains? "elixir of life", ["venus", "mercury"]
false
```

<> Elixir .

```
"Hello" <> "World"                      # => "HelloWorld"
```

List Enum.join/2 .

```
Enum.join(["A", "few", "words"], " ")        # => "A few words"
```

: https://riptutorial.com/ko/elixir/topic/2618/

17:

Examples

```
iex(1)> [x, y] = ["String1", "String2"]
iex(2)> "#{x} #{y}"
# "String1 String2"
```

IO

```
["String1", " ", "String2"] |> IO.iodata_to_binary
# "String1 String2"
```

.
:
:

```
iex(1)> IO.puts(["String1", " ", "String2"])
# String1 String2
```

Enum.join

```
Enum.join(["String1", "String2"], " ")
# "String1 String2"
```

: https://riptutorial.com/ko/elixir/topic/9202/-

18:

Examples

Elixir   .

`when  def function_name(params)  do . :`

```
defmodule Math do

  def is_even(num) when num === 1 do
    false
  end
  def is_even(num) when num === 2 do
    true
  end

  def is_odd(num) when num === 1 do
    true
  end
  def is_odd(num) when num === 2 do
    false
  end

end
```

`Math.is_even(2)`   `.is_even` , **guard** .   **guard**   . `num === 1`   . `num === 2` ,   `true` .

`Math.is_odd(1)` **?**   `num  1`   **guard** .   `true`   .

. Elixir   ., , , , (`: is_atom`) (`: length`).   .

.   . **params**   `(FunctionClauseError) no function clause matching`   .

0   .

```
defmodule BadMath do
  def divide(a) when a / 0 === :foo do
    :bar
  end
end
```

`BadMath.divide("anything")`   `(FunctionClauseError) no function clause matching in`
`BadMath.divide/1 "anything" / 0`   . : `(ArithmeticError) bad argument in arithmetic expression` .

: https://riptutorial.com/ko/elixir/topic/6121/---

19:

Examples

```
iex> :observer.start
:ok
```

`:observer.start` CPU , GUI .

: https://riptutorial.com/ko/elixir/topic/3587/

Elixir ? .. .

- Elixir , .
- .
- SO . ;)

GitHub repo elixir-constants-concept .

Examples

```
defmodule MyModule do
  @my_favorite_number 13
  @use_snake_case "This is a string (use double-quotes)"
end
```

.

:

```
defmodule MyApp.ViaFunctions.Constants do
  def app_version, do: "0.0.1"
  def app_author, do: "Felix Orr"
  def app_info, do: [app_version, app_author]
  def bar, do: "barrific constant in function"
end
```

:

```
defmodule MyApp.ViaFunctions.ConsumeWithRequire do
  require MyApp.ViaFunctions.Constants

  def foo() do
    IO.puts MyApp.ViaFunctions.Constants.app_version
    IO.puts MyApp.ViaFunctions.Constants.app_author
    IO.puts inspect MyApp.ViaFunctions.Constants.app_info
  end

  # This generates a compiler error, cannot invoke `bar/0` inside a guard.
  # def foo(_bar) when is_bitstring(bar) do
  #   IO.puts "We just used bar in a guard: #{bar}"
  # end
end
```

:

```
defmodule MyApp.ViaFunctions.ConsumeWithImport do
  import MyApp.ViaFunctions.Constants

  def foo() do
```

```
    IO.puts app_version
    IO.puts app_author
    IO.puts inspect app_info
  end
end
```

.

:

```
defmodule MyApp.ViaMacros.Constants do
  @moduledoc """
  Apply with `use MyApp.ViaMacros.Constants, :app` or `import MyApp.ViaMacros.Constants,
:app`.

  Each constant is private to avoid ambiguity when importing multiple modules
  that each have their own copies of these constants.
  """

  def app do
    quote do
      # This method allows sharing module constants which can be used in guards.
      @bar "barrific module constant"
      defp app_version, do: "0.0.1"
      defp app_author, do: "Felix Orr"
      defp app_info, do: [app_version, app_author]
    end
  end

  defmacro __using__(which) when is_atom(which) do
    apply(__MODULE__, which, [])
  end
end
```

use :

```
defmodule MyApp.ViaMacros.ConsumeWithUse do
  use MyApp.ViaMacros.Constants, :app

  def foo() do
    IO.puts app_version
    IO.puts app_author
    IO.puts inspect app_info
  end

  def foo(_bar) when is_bitstring(@bar) do
    IO.puts "We just used bar in a guard: #{@bar}"
  end
end
```

@some_constant   .   .

21:

Examples

```
dnf install erlang elixir
```

OS X MacOS Elixir   .

```
$ brew update
$ brew install elixir
```

```
 $ sudo port install elixir
```

/

```
# Fetch and install package to setup access to the official APT repository
wget https://packages.erlang-solutions.com/erlang-solutions_1.0_all.deb
sudo dpkg -i erlang-solutions_1.0_all.deb

# Update package index
sudo apt-get update

# Install Erlang and Elixir
sudo apt-get install esl-erlang
sudo apt-get install elixir
```

Gentoo / Funtoo

Elixir   .
.

```
emerge --sync
```

.

```
emerge --ask dev-lang/elixir
```

: https://riptutorial.com/ko/elixir/topic/4208/

22:

Examples

```
iex> ~w(a b c)
["a", "b", "c"]
```

```
iex> ~w(a b c)a
[:a, :b, :c]
```

sigils

sigil_X sigil_X . X (   ).

```
defmodule Sigils do
  def sigil_j(string, options) do
    # Split on the letter p, or do something more useful
    String.split string, "p"
  end
  # Use this sigil in this module, or import it to use it elsewhere
end
```

options sigil   (:).

```
~j/foople/abc # string is "foople", options are 'abc'
# ["foo", "le"]
```

: https://riptutorial.com/ko/elixir/topic/2204/

23:

Examples

Erlang

Erlang . Erlang `:math` :

```
iex> :math.pi
3.141592653589793
```

Erlang

Erlang `module_info` :

```
iex> :math.module_info
 [module: :math,
  exports: [pi: 0, module_info: 0, module_info: 1, pow: 2, atan2: 2, sqrt: 1,
   log10: 1, log2: 1, log: 1, exp: 1, erfc: 1, erf: 1, atanh: 1, atan: 1,
   asinh: 1, asin: 1, acosh: 1, acos: 1, tanh: 1, tan: 1, sinh: 1, sin: 1,
   cosh: 1, cos: 1],
  attributes: [vsn: [113168357788724588783826225069997113388]],
  compile: [options: [{:outdir,
     '/private/tmp/erlang20160316-36404-xtp7cq/otp-OTP-18.3/lib/stdlib/src/../ebin'},
    {:i,
     '/private/tmp/erlang20160316-36404-xtp7cq/otp-OTP-18.3/lib/stdlib/src/../include'},
    {:i,
     '/private/tmp/erlang20160316-36404-xtp7cq/otp-OTP-
18.3/lib/stdlib/src/../../kernel/include'},
    :warnings_as_errors, :debug_info], version: '6.0.2',
   time: {2016, 3, 16, 16, 40, 35},
   source: '/private/tmp/erlang20160316-36404-xtp7cq/otp-OTP-18.3/lib/stdlib/src/math.erl'],
  native: false,
 md5: <<85, 35, 110, 210, 174, 113, 103, 228, 63, 252, 81, 27, 224, 15, 64,
   44>>]
```

: https://riptutorial.com/ko/elixir/topic/2716/

24: .gitignore

.gitignore : https://riptutorial.com/ko/elixir/topic/6493/----gitignore

25: .gitignore

.gitignore `/rel` `.exrm` .

Examples

.gitignore

```
/_build
/cover
/deps
erl_crash.dump
*.ez

# Common additions for various operating systems:
# MacOS
.DS_Store

# Common additions for various editors:
# JetBrains IDEA, IntelliJ, PyCharm, RubyMine etc.
.idea
```

```
### Elixir ###
/_build
/cover
/deps
erl_crash.dump
*.ez

### Erlang ###
.eunit
deps
*.beam
*.plt
ebin
rel/example_project
.concrete/DEV_MODE
.rebar
```

```
/_build
/cover
/deps
erl_crash.dump
*.ez
/rel
```

```
/_build
/db
/deps
/*.ez
erl_crash.dump
/node_modules
/priv/static/
```

```
/config/prod.secret.exs
/rel
```

.gitignore

, `mix new <projectname>` Elixir `.gitignore` .

```
# The directory Mix will write compiled artifacts to.
/_build

# If you run "mix test --cover", coverage assets end up here.
/cover

# The directory Mix downloads your dependencies sources to.
/deps

# Where 3rd-party dependencies like ExDoc output generated docs.
/doc

# If the VM crashes, it generates a dump, let's ignore it too.
erl_crash.dump

# Also ignore archive artifacts (built via "mix archive.build").
*.ez
```

.gitignore : https://riptutorial.com/ko/elixir/topic/6526/----gitignore

26:

.

Examples

Map   .

```
defmodule MyList do
  def map([], _func) do
    []
  end

  def map([head | tail], func) do
    [func.(head) | map(tail, func)]
  end
end
```

iex   :

```
MyList.map [1,2,3], fn a -> a * 5 end
```

```
MyList.map [1,2,3], &(&1 * 5)
```

Reduce ,   *(accumulator)*   ( )

```
defmodule MyList do
  def reduce([], _func, acc) do
    acc
  end

  def reduce([head | tail], func, acc) do
    reduce(tail, func, func.(acc, head))
  end
end
```

iex .

1. : `MyList.reduce [1,2,3,4], fn acc, element -> acc + element end, 0`
2. mutliply : `MyList.reduce [1,2,3,4], fn acc, element -> acc * element end, 1`

1 :

```
Iteration 1 => acc = 0, element = 1 ==> 0 + 1 ===> 1 = next accumulator
Iteration 2 => acc = 1, element = 2 ==> 1 + 2 ===> 3 = next accumulator
Iteration 3 => acc = 3, element = 3 ==> 3 + 3 ===> 6 = next accumulator
Iteration 4 => acc = 6, element = 4 ==> 6 + 4 ===> 10 = next accumulator = result(as all
elements are done)
```

reduce

```
MyList.reduce [1,2,3,4], fn acc, element -> if rem(element,2) == 0 do acc else acc ++
[element] end end, []
```

:

27:

(Polymorphism) . . . . Elixir `protocols` .

, `Any` ., Enum String.Char .

Examples

.

```
defmodule Kelvin do
  defstruct name: "Kelvin", symbol: "K", degree: 0
end

defmodule Fahrenheit do
  defstruct name: "Fahrenheit", symbol: "°F", degree: 0
end

defmodule Celsius do
  defstruct name: "Celsius", symbol: "°C", degree: 0
end

defprotocol Temperature do
  @doc """
  Convert Kelvin and Fahrenheit to Celsius degree
  """
  def to_celsius(degree)
end

defimpl Temperature, for: Kelvin do
  @doc """
  Deduct 273.15
  """
  def to_celsius(kelvin) do
    celsius_degree = kelvin.degree - 273.15
    %Celsius{degree: celsius_degree}
  end
end

defimpl Temperature, for: Fahrenheit do
  @doc """
  Deduct 32, then multiply by 5, then divide by 9
  """
  def to_celsius(fahrenheit) do
    celsius_degree = (fahrenheit.degree - 32) * 5 / 9
    %Celsius{degree: celsius_degree}
  end
end
```

Kelvin Fahrenheit . .

```
iex> fahrenheit = %Fahrenheit{degree: 45}
%Fahrenheit{degree: 45, name: "Fahrenheit", symbol: "°F"}
iex> celsius = Temperature.to_celsius(fahrenheit)
%Celsius{degree: 7.22, name: "Celsius", symbol: "°C"}
```

```
iex> kelvin = %Kelvin{degree: 300}
%Kelvin{degree: 300, name: "Kelvin", symbol: "K"}
iex> celsius = Temperature.to_celsius(kelvin)
%Celsius{degree: 26.85, name: "Celsius", symbol: "°C"}
```

```
to_celsius        .
```

```
iex> Temperature.to_celsius(%{degree: 12})
** (Protocol.UndefinedError) protocol Temperature not implemented for %{degree: 12}
     iex:11: Temperature.impl_for!/1
     iex:15: Temperature.to_celsius/1
```

: https://riptutorial.com/ko/elixir/topic/9519/-

28:

Examples

Agent .   .   .

```
iex(1)> {:ok, pid} = Agent.start_link(fn -> :initial_value end)
{:ok, #PID<0.62.0>}
iex(2)> Agent.get(pid, &(&1))
:initial_value
iex(3)> Agent.update(pid, fn(value) -> {value, :more_data} end)
:ok
iex(4)> Agent.get(pid, &(&1))
{:initial_value, :more_data}
```

: https://riptutorial.com/ko/elixir/topic/6596/--

Examples

Pipe Operator `|>`   .

```
expression |> function
```

.

:

```
Oven.bake(Ingredients.Mix([:flour, :cocoa, :sugar, :milk, :eggs, :butter]), :temperature)
```

Oven.bake Ingredients.mix Oven.bake .,   :temperature Oven.bake   Oven.bake

:

```
[:flour, :cocoa, :sugar, :milk, :eggs, :butter]
|> Ingredients.mix
|> Oven.bake(:temperature)
```

., :temperature Oven.bake .

. :

```
Enum.each([1, 2, 3], &(&1+1)) # produces [2, 3, 4]
```

:

```
[1, 2, 3]
|> Enum.each(&(&1+1))
```

.

```
foo 1 |> bar 2 |> baz 3
```

:

```
foo(1)  |> bar(2)  |> baz(3)
```

Elixir   .

- (`true false`)

```
x or y      # true if x is true, otherwise y
```

```
x and y     # false if x is false, otherwise y

not x       # false if x is true, otherwise true
```

ArgumentError . true false (nil).

```
iex(1)> false and 1 # return false
iex(2)> false or 1 # return 1
iex(3)> nil and 1 # raise (ArgumentError) argument error: nil
```

  - (, false nil   true)

```
x || y     # x if x is true, otherwise y

x && y     # y if x is true, otherwise false

!x         # false if x is true, otherwise true
```

|| (Elixir nil false   true).   .

```
iex(1)> 1 || 3 # return 1, because 1 is truthy
iex(2)> false || 3 # return 3
iex(3)> 3 || false # return 3
iex(4)> false || nil # return nil
iex(5)> nil || false # return false
```

&&   . , false nil .

```
iex(1)> 1 && 3 # return 3, first argument is truthy
iex(2)> false && 3 # return false
iex(3)> 3 && false # return false
iex(4)> 3 && nil # return nil
iex(5)> false && nil # return false
iex(6)> nil && false # return nil
```

&& || .   .

!   .

```
iex(1)> !2 # return false
iex(2)> !false # return true
iex(3)> !"Test" # return false
iex(4)> !nil # return true
```

.

```
iex(1)> !!true # return true
iex(2)> !!"Test" # return true
iex(3)> !!nil # return false
iex(4)> !!false # return false
```

:

- `x == y` (`1 == 1.0 # true`)
- `x == y` (`1 != 1.0 # false`)
- `x === y` (`1 === 1.0 # false`)
- `x === y` (`1 !== 1.0 # true`)

:

- `x > y`
- `x >= y`
- `x < y`
- `x <= y`

.   .

```
number < atom < reference < function < port < pid < tuple < map < list < binary
```

( ) () .

```
iex(1)> [1, 2, 3] ++ [4, 5]
[1, 2, 3, 4, 5]

iex(2)> [1, 2, 3, 4, 5] -- [1, 3]
[2, 4, 5]

iex(3)> "qwe" <> "rty"
"qwerty"
```

"

in   .

```
iex(4)> 1 in [1, 2, 3, 4]
true

iex(5)> 0 in (1..5)
false
```

: https://riptutorial.com/ko/elixir/topic/1161/

30:

Examples

Ecto.Repo

3   .

1. Ecto.Repo elixir   app otp_app .

```
defmodule Repo do
  use Ecto.Repo, otp_app: :custom_app
end
```

2. Repo   . .

```
config :custom_app, Repo,
    adapter: Ecto.Adapters.Postgres,
    database: "ecto_custom_dev",
    username: "postgres_dev",
    password: "postgres_dev",
    hostname: "localhost",
   # OR use a URL to connect instead
   url: "postgres://postgres_dev:postgres_dev@localhost/ecto_custom_dev"
```

3. Ecto   Ecto . lib / custom_app.ex Ecto .

```
   def start(_type, _args) do
     import Supervisor.Spec

     children = [
      supervisor(Repo, [])
     ]

     opts = [strategy: :one_for_one, name: MyApp.Supervisor]
     Supervisor.start_link(children, opts)
   end
```

Repo.get_by / 3 ""

Ecto.Queryable   Post .

"hello" "world"   .

```
 MyRepo.get_by(Post, [title: "hello", description: "world"])
```

Repo.get_by   .

.

```
some_field = :id
some_value = 10

from p in Post, where: field(p, ^some_field) == ^some_value
```

( )

postgres .

, (`mix ecto.gen.migration`) .

```
def up do
  # creating the enumerated type
  execute("CREATE TYPE post_status AS ENUM ('published', 'editing')")

  # creating a table with the column
  create table(:posts) do
    add :post_status, :post_status, null: false
  end
end

def down do
  drop table(:posts)
  execute("DROP TYPE post_status")
end
```

, Elixir .

```
schema "posts" do
  field :post_status, :string
end
```

`Ecto.Type` .

`ecto_enum` . **github** .

Phoenix enum_ecto .

: https://riptutorial.com/ko/elixir/topic/6524/

31:

do...end   .

```
unless false, do: IO.puts("Condition is false")
# Outputs "Condition is false"

# With an `else`:
if false, do: IO.puts("Condition is true"), else: IO.puts("Condition is false")
# Outputs "Condition is false"
```

Examples

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

case   . {1,2}   .

```
if true do
    "Will be seen since condition is true."
end

if false do
    "Won't be seen since condition is false."
else
    "Will be seen.
end

unless false do
    "Will be seen."
end

unless true do
    "Won't be seen."
else
    "Will be seen."
end
```

cond

```
cond do
   0 == 1 -> IO.puts "0 = 1"
   2 == 1 + 1 -> IO.puts "1 + 1 = 2"
   3 == 1 + 2 -> IO.puts "1 + 2 = 3"
end

# Outputs "1 + 1 = 2" (first condition evaluating to true)
```

true `CondClauseError` cond `CondClauseError` `CondClauseError` .

```
cond do
  1 == 2 -> "Hmmm"
  "foo" == "bar" -> "What?"
end
# Error
```

.

```
cond do
  ... other conditions
  true -> "Default value"
end
```

.

with

`with` . `()` . : DB .

`with` `with` `(` `)`.

```
case create_user(user_params) do
  {:ok, user} ->
    case Mailer.compose_email(user) do
      {:ok, email} ->
        Mailer.send_email(email)
      {:error, reason} ->
        handle_error
    end
  {:error, changeset} ->
    handle_error
end
```

`case`   `case` `(` cond if `)`.   ' (pyramid of doom)' .   . `with` **statement**   :

```
with {:ok, user} <- create_user(user_params),
     {:ok, email} <- Mailer.compose_email(user) do
  {:ok, Mailer.send_email}
else
  {:error, _reason} ->
    handle_error
end
```

`case` **with** `with` `with` . `with` `(` `)` .   `with` `do` , `else` .

`else` , `with` `do`   .

`with` `do` .

32:

- map = % {} // .
- map = % {: a => 1, : b => 2} //   .
- list = [] // .
- list = [{: a, 1}, {: b, 2}] //   .

:   .

- (   ) .

/ .   .

Examples

- (   ) . `%w{}` .

```
%{} // creates an empty map
%{:a => 1, :b => 2} // creates a non-empty map
```

.

```
%{"a" => 1, "b" => 2}
%{1 => "a", 2 => "b"}
```

. "

```
// keys are integer or strings
%{1 => "a", "b" => :foo}
// values are string or nil
%{1 => "a", 2 => nil}
```

.

```
%{a: 1, b: 2}
```

/ .

```
[{:a, 1}, {:b, 2}] // creates a non-empty keyword list
```

.

```
[{:a, 1}, {:a, 2}, {:b, 2}]
[{:a, 1}, {:b, 2}, {:a, 2}]
```

.

```
[{"a", 1}, {:a, 2}, {2, "b"}]
```

.   .   .

.

:

- .
- .

.

- /   .
- .
-

: https://riptutorial.com/ko/elixir/topic/2706/---

33:

Examples

!

.   .   .   1 %   .   .

1   .   10 %   10   .

ExProf   .

: https://riptutorial.com/ko/elixir/topic/6062/

34:

- Task.async ()
- Task.await ()

	.
	Task.async .
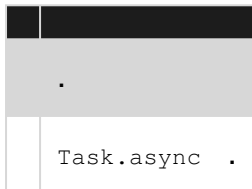
Examples

.

```
task = Task.async(fn -> expensive_computation end)
do_something_else
result = Task.await(task)
```

```
crawled_site = ["http://www.google.com", "http://www.stackoverflow.com"]
|> Enum.map(fn site -> Task.async(fn -> crawl(site) end) end)
|> Enum.map(&Task.await/1)
```

: https://riptutorial.com/ko/elixir/topic/7588/

35:

Elixir .

Examples

Sigils

x sigil sigil_x .

```
defmodule MySigils do
  #returns the downcasing string if option l is given then returns the list of downcase
letters
  def sigil_l(string,[]), do: String.Casing.downcase(string)
  def sigil_l(string,[?l]), do: String.Casing.downcase(string) |> String.graphemes

  #returns the upcasing string if option l is given then returns the list of downcase letters
  def sigil_u(string,[]), do: String.Casing.upcase(string)
  def sigil_u(string,[?l]), do: String.Casing.upcase(string) |> String.graphemes
end
```

[OR]

OR . true . .

```
# Regular Approach
find = fn(x) when x>10 or x<5 or x==7 -> x end

# Our Hack
hell = fn(x) when true in [x>10,x<5,x==7] -> x end
```

iex - iex

.iex.exs ~ home . .

```
# IEx.configure colors: [enabled: true]
# IEx.configure colors: [ eval_result: [ :cyan, :bright ] ]
IO.puts IO.ANSI.red_background() <> IO.ANSI.white() <> " ❄❄❄ Good Luck with Elixir ❄❄❄ " <> IO.ANSI.reset
Application.put_env(:elixir, :ansi_enabled, true)
IEx.configure(
 colors: [
  eval_result: [:green, :bright] ,
  eval_error: [[:red,:bright,"Bug Bug ..!!"]],
  eval_info: [:yellow, :bright ],
 ],
 default_prompt: [
  "\e[G",   # ANSI CHA, move cursor to column 1
  :white,
  "I",
  :red,
  "❤" ,     # plain string
```

```
  :green,
  "%prefix",:white,"|",
   :blue,
   "%counter",
   :white,
   "|",
  :red,
  "▶" ,        # plain string
  :white,
  "▶▶" ,       # plain string
   # ❤ ❤-»" ,  # plain string
  :reset
 ] |> IO.ANSI.format |> IO.chardata_to_string

)
```

:

36:

Examples

```
#You can use pattern matching to run different
#functions based on which parameters you pass

#This example uses pattern matching to start,
#run, and end a recursive function

defmodule Counter do
    def count_to do
        count_to(100, 0) #No argument, init with 100
    end

    def count_to(counter) do
        count_to(counter, 0) #Initialize the recursive function
    end

    def count_to(counter, value) when value == counter do
        #This guard clause allows me to check my arguments against
        #expressions. This ends the recursion when the value matches
        #the number I am counting to.
        :ok
    end

    def count_to(counter, value) do
        #Actually do the counting
        IO.puts value
        count_to(counter, value + 1)
    end
end
```

```
%{username: username} = %{username: "John Doe", id: 1}
# username == "John Doe"
```

```
%{username: username, id: 2} = %{username: "John Doe", id: 1}
** (MatchError) no match of right hand side value: %{id: 1, username: "John Doe"}
```

.

.

```
[head | tail] = [1,2,3,4,5]
# head == 1
# tail == [2,3,4,5]
```

( ) ᛁ () ᛁ .

.

```
[1,2 | tail] = [1,2,3,4,5]
```

```
# tail = [3,4,5]

[4 | tail] = [1,2,3,4,5]
** (MatchError) no match of right hand side value: [1, 2, 3, 4, 5]
```

| . .

```
[a, b | tail] = [1,2,3,4,5]
# a == 1
# b == 2
# tail = [3,4,5]
```

-       :

```
iex(11)> [a = 1 | tail] = [1,2,3,4,5]
# a == 1
```

```
defmodule Math do
  # We start of by passing the sum/1 function a list of numbers.
  def sum(numbers) do
    do_sum(numbers, 0)
  end

  # Recurse over the list when it contains at least one element.
  # We break the list up into two parts:
  #   head: the first element of the list
  #   tail: a list of all elements except the head
  # Every time this function is executed it makes the list of numbers
  # one element smaller until it is empty.
  defp do_sum([head|tail], acc) do
    do_sum(tail, head + acc)
  end

 # When we have reached the end of the list, return the accumulated sum
  defp do_sum([], acc), do: acc
end
```

```
f = fn
  {:a, :b} -> IO.puts "Tuple {:a, :b}"
  [] -> IO.puts "Empty list"
end

f.({:a, :b}) # Tuple {:a, :b}
f.([])       # Empty list
```

```
{ a, b, c } = { "Hello", "World", "!" }

IO.puts a # Hello
IO.puts b # World
IO.puts c # !

# Tuples of different size won't match:

{ a, b, c } = { "Hello", "World" } # (MatchError) no match of right hand side value: {
"Hello", "World" }
```

.

sample.txt This is a sample text .

```
{ :ok, file } = File.read("sample.txt")
# => {:ok, "This is a sample text"}

file
# => "This is a sample text"
```

, :

```
{ :ok, file } = File.read("sample.txt")
# => ** (MatchError) no match of right hand side value: {:error, :enoent}

{ :error, msg } = File.read("sample.txt")
# => {:error, :enoent}
```

```
fizzbuzz = fn
  (0, 0, _) -> "FizzBuzz"
  (0, _, _) -> "Fizz"
  (_, 0, _) -> "Buzz"
  (_, _, x) -> x
end

my_function = fn(n) ->
  fizzbuzz.(rem(n, 3), rem(n, 5), n)
end
```

: https://riptutorial.com/ko/elixir/topic/1602/-

37:

Examples

Greeter greet   .

```
defmodule Greeter do
    def greet do
        IO.puts "Hello programmer!"
    end
end

iex> spawn(Greeter, :greet, [])
Hello
#PID<0.122.0>
```

#PID<0.122.0>   .

```
defmodule Processes do
    def receiver do
        receive do
            {:ok, val} ->
                IO.puts "Received Value: #{val}"
            _ ->
                IO.puts "Received something else"
        end
    end
end
```

```
iex(1)> pid = spawn(Processes, :receiver, [])
#PID<0.84.0>
iex(2)> send pid, {:ok, 10}
Received Value: 10
{:ok, 10}
```

.

```
defmodule Processes do
    def receiver do
        receive do
            {:ok, val} ->
                IO.puts "Received Value: #{val}"
            _ ->
                IO.puts "Received something else"
        end
        receiver
    end
end
```

```
iex(1)> pid = spawn Processes, :receiver, []
#PID<0.95.0>
iex(2)> send pid, {:ok, 10}
Received Value: 10
```

```
{:ok, 10}
iex(3)> send pid, {:ok, 42}
{:ok, 42}
Received Value: 42
iex(4)> send pid, :random
:random
Received something else
```

Elixir   .

: https://riptutorial.com/ko/elixir/topic/3173/

38:

.

Examples

Elixir . `defprotocol` :

```
defprotocol Log do
  def log(value, opts)
end
```

`defimpl` :

```
require Logger
# User and Post are custom structs

defimpl Log, for: User do
  def log(user, _opts) do
    Logger.info "User: #{user.name}, #{user.age}"
  end
end

defimpl Log, for: Post do
  def log(user, _opts) do
    Logger.info "Post: #{post.title}, #{post.category}"
  end
end
```

.

```
iex> Log.log(%User{name: "Yos", age: 23})
22:53:11.604 [info]  User: Yos, 23
iex> Log.log(%Post{title: "Protocols", category: "Protocols"})
22:53:43.604 [info]  Post: Protocols, Protocols
```

. `Atom` , `BitString` , `Tuples` .

: https://riptutorial.com/ko/elixir/topic/3487/

39:

Examples

. .

.

```
defmodule Parser do
  @callback parse(String.t) :: any
  @callback extensions() :: [String.t]
end
```

:

```
defmodule JSONParser do
  @behaviour Parser

  def parse(str), do: # ... parse JSON
  def extensions, do: ["json"]
end
```

`@behaviour`   Parser   .   .

`@behaviour`   .

Examples

```
# lib/mix/tasks/mytask.ex
defmodule Mix.Tasks.MyTask do
  use Mix.Task

  @shortdoc "A simple mix task"
  def run(_) do
    IO.puts "YO!"
  end
end
```

:

```
$ mix compile
$ mix my_task
"YO!"
```

run/1 . : def run(args) do ... end

```
defmodule Mix.Tasks.Example_Task do
  use Mix.Task

  @shortdoc "Example_Task prints hello + its arguments"
  def run(args) do
    IO.puts "Hello #{args}"
  end
end
```

:

```
$ mix example_task world
"hello world"
```

Elixir   .   .

Elixir mix.exs .

project   aliases/0 .   ()   .

```
  def project do
    [app: :my_app,
     ...
     aliases: aliases()]
  end
```

aliases/0 (: mix.exs).

___

```
  ...

  defp aliases do
    [go: "phoenix.server",
     trident: "do deps.get, compile, go"]
  end
```

$ mix go Phoenix (Phoenix). $ mix trident mix ,, .

.

```
 mix help
```

mix help task :

```
 mix help cmd
```

:

41:

.

( ) . `Enum`   `Enum` `Stream`   .

Examples

`Stream`   . `Stream`   (, `Enum`   ).

```
numbers = 1..100
|> Stream.map(fn(x) -> x * 2 end)
|> Stream.filter(fn(x) -> rem(x, 2) == 0 end)
|> Stream.take_every(3)
|> Enum.to_list

[2, 8, 14, 20, 26, 32, 38, 44, 50, 56, 62, 68, 74, 80, 86, 92, 98, 104, 110,
 116, 122, 128, 134, 140, 146, 152, 158, 164, 170, 176, 182, 188, 194, 200]
```

3 (`map` , `filter` , `take_every`) `take_every` `Enum.to_list`   .

`Stream`   .   . `Enum` . 3 .

S. No		Contributors
1	Elixir	alejosocorro, Andrey Chernykh, Ben Bals, Community, cwc, Delameko, Douglas Correa, helcim, I Am Batman, JAlberto, koolkat, leifg, MattW., rap-2-h, Simone Carletti, Stephan Rodemeier, Vinicius Quaiato, Yedhu Krishnan, Zimm i48
2	Doctests	aholt, milmazz, Philippe-Arnaud de MANGOU, Yos Riady
3	ExDoc	milmazz, Yos Riady
4	ExUnit	Yos Riady
5	IEx	alxndr, Cifer, fahrradflucht, legoscia, mudasobwa, muttonlamb, PatNowak, Paweł Obrok, sbs, Sheharyar, Simone Carletti, Stephan Rodemeier, Uniaika, Vincent, Yos Riady
6	IEx	helcim
7	IO.inspect	leifg
8		Andrey Chernykh, cwc, Dair, Eiji, Filip Haglund, PatNowak, rainteller, Simone Carletti, Stephan Rodemeier, Yedhu Krishnan, Yos Riady
9		Ben Bals, Candy Gumdrop, emoragaf, PatNowak, Sheharyar, Yos Riady
10		Andrey Chernykh, Arithmeticbird, Oskar, TreyE, Vinicius Quaiato
11		Yos Riady
12		Sam Mercier, Simone Carletti, Stephan Rodemeier, Yos Riady
13		javanut13, Paweł Obrok, Pfitz, Philippe-Arnaud de MANGOU, sbs
14		4444, Paweł Obrok
15		Alex G, javanut13, Yos Riady
16		Alex G, Sheharyar, Yos Riady
17		Agung Santoso
18		alxndr

19		Yos Riady
20		ibgib
21		cwc, Douglas Correa, Eiji, JAlberto, MattW.
22		javanut13, Yos Riady
23		4444, Yos Riady
24	.gitignore	Yos Riady
25		Dinesh Balasubramanian
26		mustafaturan
27		Paweł Obrok
28		alxndr, Andrey Chernykh, Dair, Gazler, Mitkins, nirev, PatNowak
29		fgutierr, Philippe-Arnaud de MANGOU, toraritte
30		Andrey Chernykh, evuez, javanut13, Musfiqur Rahman, Paweł Obrok
31		Sam Mercier, Simone Carletti, Yos Riady
32		Filip Haglund, legoscia
33		mario
34		Ankanna
35		Alex Anderson, Dair, Danny Rosenblatt, evuez, Gabriel C, gmile, Harrison Lucas, javanut13, Oskar, PatNowak, theIV, Thomas, Yedhu Krishnan
36		Alex G, Yedhu Krishnan
37		Yos Riady
38		Yos Riady
39		4444, helcim, rainteller, Slava.K, Yos Riady
40		Oskar