Macros

In this Video, we are going to take a look at…

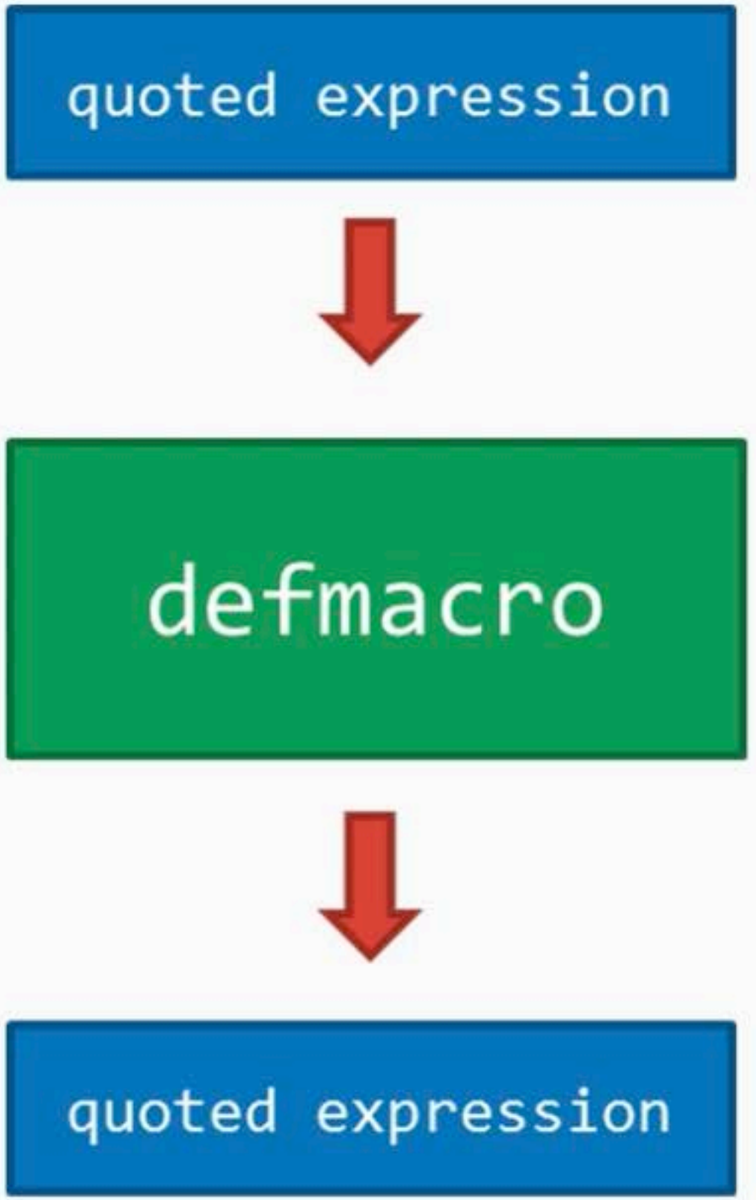
- Creating macros

Macros

The Elixir-provided mechanism to expand the language with new constructs.

Macros

The Elixir-provided mechanism to expand the language with new constructs.



task Macro

```
@async false
@timeout 10000
@timeout_reply “Hello, Unknown”
task say_hello(name) do
  “Hello, #{name}”
end

@async true
task do_something() do
  1 + 1
end
```

task Macro

```
@async false
@timeout 10000
@timeout_reply “Hello, Unknown”
task say_hello(name) do
  “Hello, #{name}”
end

@async true
task do_something() do
  1 + 1
end
```

Run the code as a process

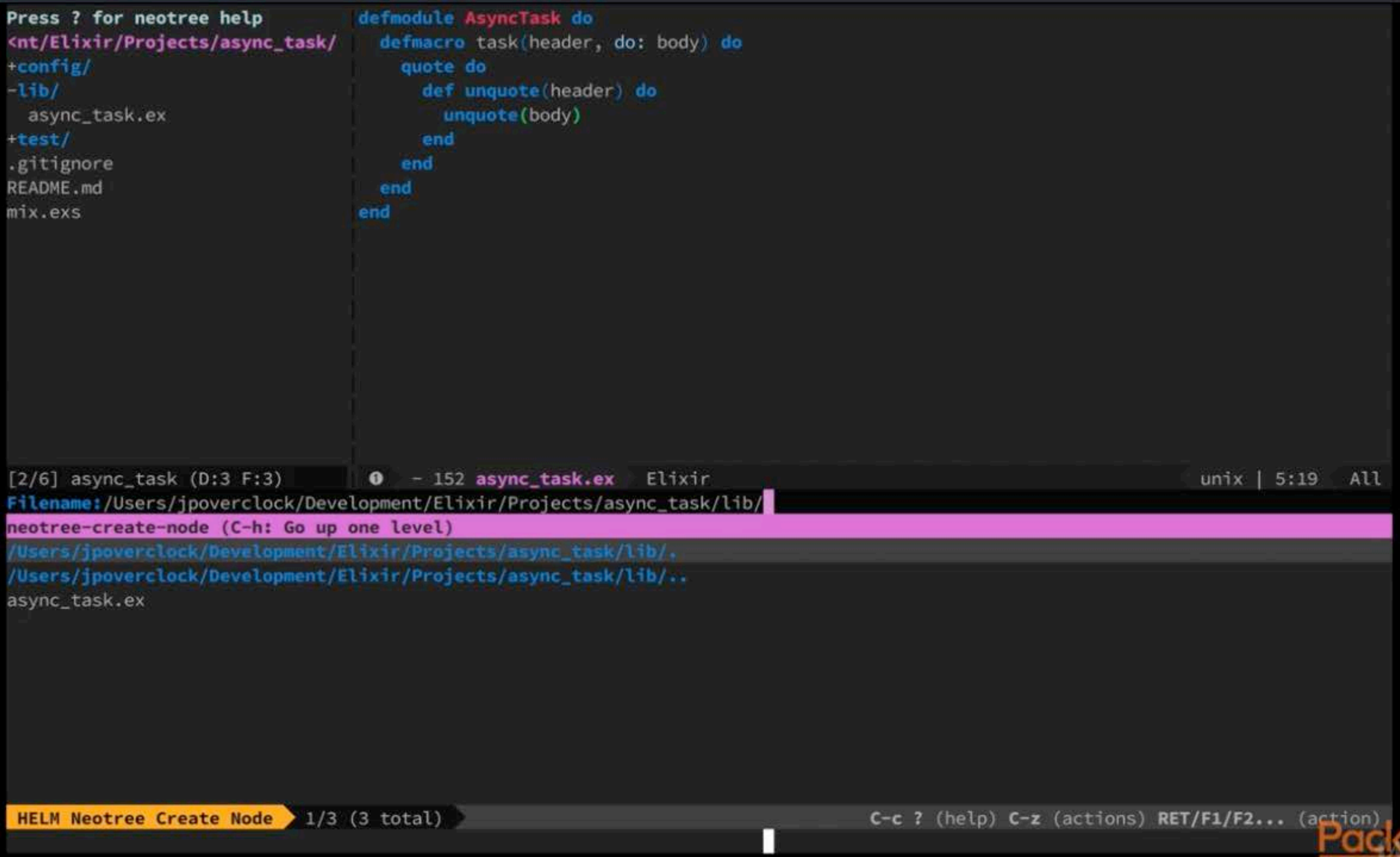
Wait for the result unless async

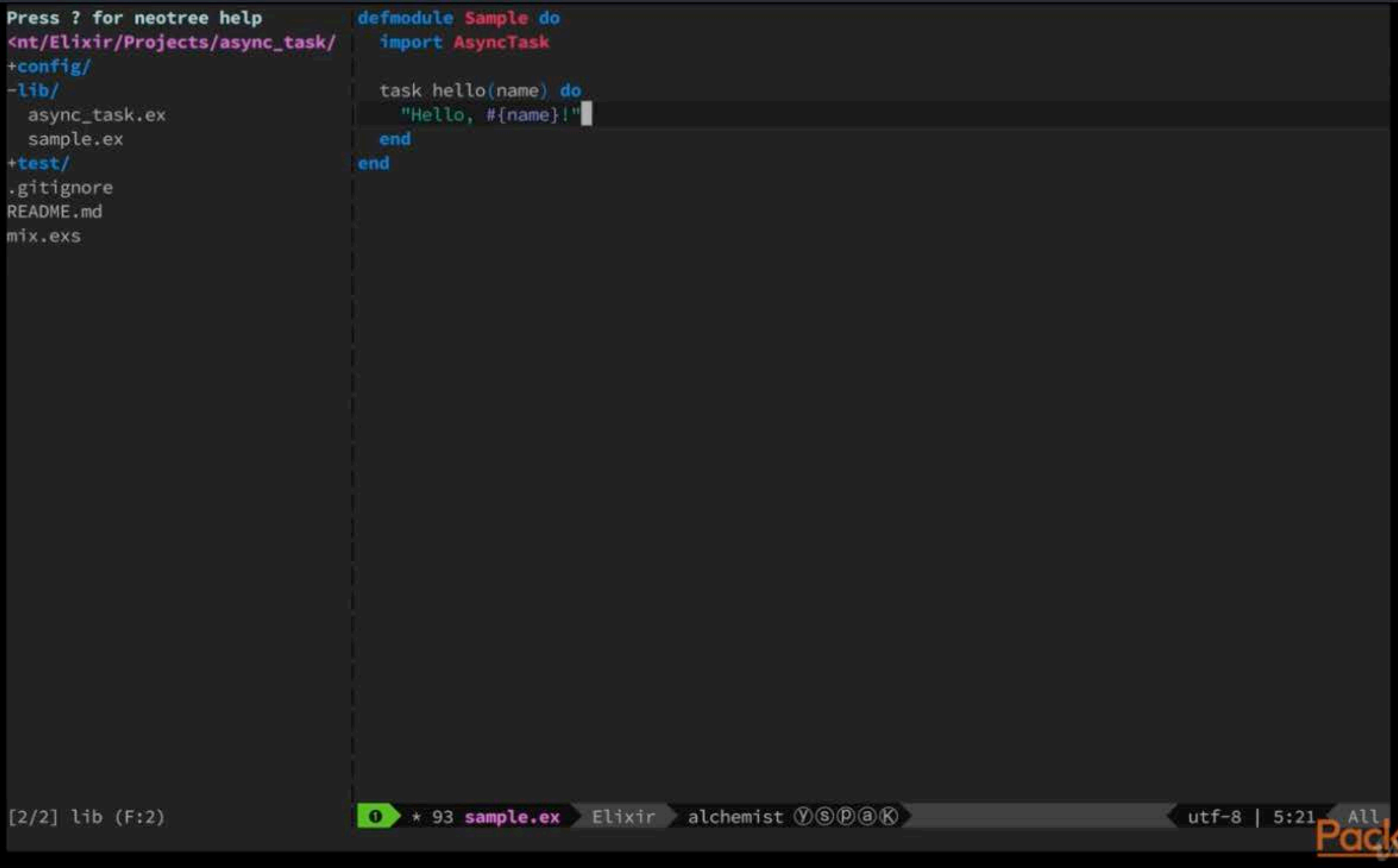
```
➜ Projects mix new async_task
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/async_task.ex
* creating test
* creating test/test_helper.exs
* creating test/async_task_test.exs

Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

    cd async_task
    mix test

Run "mix help" for more commands.
➜ Projects cd async_task
➜ async_task
```


Press ? for neotree help
<nt/Elixir/Projects/async_task/
+config/
-lib/
async_task.ex
+test/
.gitignore
README.md
mix.exs
defmodule AsyncTask do
defmacro task(header, do: body) do
quote do
def unquote(header) do
unquote(body)
end
end
end
end
[2/6] async_task (D:3 F:3)
- 152 async_task.ex Elixir
unix | 5:19 All
Filename:/Users/jpoverclock/Development/Elixir/Projects/async_task/lib/
neotree-create-node (C-h: Go up one level)
/Users/jpoverclock/Development/Elixir/Projects/async_task/lib/.
/Users/jpoverclock/Development/Elixir/Projects/async_task/lib/..
async_task.ex
HELM Neotree Create Node
1/3 (3 total)
C-c ? (help) C-z (actions) RET/F1/F2... (action)


Press ? for neotree help
<nt/Elixir/Projects/async_task/
+config/
-lib/
async_task.ex
sample.ex
+test/
.gitignore
README.md
mix.exs
defmodule Sample do
import AsyncTask
task hello(name) do
"Hello, #{name}!"
end
end
[2/2] lib (F:2)
1 * 93 sample.ex Elixir alchemist
utf-8 | 5:21 All

```
➜ async_task iex -S mix
Erlang/OTP 20 [erts-9.0.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Compiling 2 files (.ex)
Generated async_task app
Interactive Elixir (1.5.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> Sample.hello("Joao")
"Hello, Joao!"
iex(2)>
```

```elixir
defmodule AsyncTask do
  defmacro task(header, do: body) do
    quote do
      def unquote(header) do
        case @async do
          true -> spawn(fn -> unquote(body) end)
          _ ->
            caller = self()
            spawn(fn -> send(caller, unquote(body)) end)

            receive do
              message -> message
            after
              @timeout -> @timeout_response
            end
        end
      end
    end
  end
end
```


:w

```
Press ? for neotree help
<nt/Elixir/Projects/async_task/
+_build/
+config/
-lib/
  async_task.ex
  sample.ex
+test/
.gitignore
README.md
mix.exs
```

```elixir
defmodule Sample do
  import AsyncTask

  @async false
  @timeout 1000
  @timeout_response "Hello, Unknown!"
  task hello(name, timer \\ 10_000) do
    :timer.sleep(timer)
    "Hello, #{name}!"
  end
end
```

```
[2/2] lib (F:2)
1 * 204 sample.ex  Elixir  alchemist ⓎⓈⓅⓐⓀ  unix | 8:23  All
```

```
➜ async_task iex -S mix
Erlang/OTP 20 [erts-9.0.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Compiling 2 files (.ex)
Interactive Elixir (1.5.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> Sample.hello("Joao", 500)
"Hello, Joao!"
iex(2)> Sample.hello("Joao", 500)
"Hello, Joao!"
iex(3)> Sample.hello("Joao", 5000)
"Hello, Unknown!"
iex(4)>
```

```elixir
defmodule AsyncTask do
  defmacro __using__(_opts) do
    quote do
      import AsyncTask

      @async false
      @timeout 10_000
      @timeout_response nil
    end
  end

  defmacro task(header, do: body) do
    quote do
      def unquote(header) do
        case @async do
          true -> spawn(fn -> unquote(body) end)
          _ ->
            caller = self()
            spawn(fn -> send(caller, unquote(body)) end)

            receive do
              message -> message
            after
              @timeout -> @timeout_response
            end
        end
      end
    end
  end
end
```


Wrote /Users/jpoverclock/Development/Elixir/Projects/async_task/lib/async_task.ex

```elixir
defmodule Sample do
  import AsyncTask

  @async false
  @timeout 1000
  @timeout_response "Hello, Unknown!"
  task hello(name, timer \\ 10_000) do
    :timer.sleep(timer)
    "Hello, #{name}!"
  end
end
```

```elixir
defmodule Sample do
  use AsyncTask

  @timeout 1000
  @timeout_response "Hello, Unknown!"
  task hello(name, timer \\ 10_000) do
    :timer.sleep(timer)
    "Hello, #{name}!"
  end
end
```

```
➜ async_task iex -S mix
Erlang/OTP 20 [erts-9.0.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Compiling 2 files (.ex)
Interactive Elixir (1.5.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> Sample.hello("Joao", 6000)
"Hello, Unknown!"
iex(2)> Sample.hello("Joao", 500)
```

Use macros sparingly, but when you need them, USE THEM

Summary

- Explored quote and unquote
- Discussed macros