The purple side has a question or a code snippet which returns a value

The white side has the answer or the return value

F1

```
cond do
  2 + 2 == 5 -> "boo"
  2 * 2 == 3 -> "foo"
  1 + 1 == 2 -> "hey"
end
```

```
=> "hey"
```

F1

```
f2 = fn -> 99 end
f2.()
```

=> 99

```
div(10, 2)
```

=> 5

F1

```
length([1, 2, 3])
```

=> 3

F1

```
rem(10, 3)
```

=> 1

```
tuple_size({:ok, "boo"})
```

=> 2

F1

round 3.58

=> 4

F1

trunc 3.58

=> 3

```
foo = fn a, b ->
        a + b
      end

is_function(foo)
```

```
=> true
```

F1

```
foo = fn a, b ->
        a + b
      end

is_function(foo, 2)
```

```
=> true
```

2 is the arity of the function

F1

```
0.1 + 0.2
```

```
=> 0.30000000000000004
```

Computers can only store integers, so they need a way of representing decimal numbers, which comes with some inaccuracy

F1

```
[m | _g] = [1, 2, 3]

m
```

=> 1

```
[_m | g] = [1, 2, 3]

g
```

```
=> [2, 3]
```

F1

```
hd([1, 2, 3])
```

=> 1

```
tl([1, 2, 3])
```

```
=> [2, 3]
```

F1

```
byte_size("José")
```

=> 5

special characters
like "é" weigh
more than 1 byte

F1

```
"Hello" <> "!"
```

```
=> "Hello!"
```

```
true == :true
```

```
=> true
```

`true` is an atom. Atoms hold their name and their value. `false` and `nil` are atoms.

F1

```
elem({:ok, "hello"}, 1)
```

```
=> "hello"
```

F1

```
x = fn a, b -> a + b end
y = fn a -> x.(a, 2) end
y.(2)
```

=> 4

x is bound to a function,

which is called later, within the function which is bound to variable y

F1

```
'hello' == "hello"
```

```
=> false
```

'hello' is a char list

"hello" is a string

F1

```
[1, 2] ++ [4, 5]
```

```
=> [1, 2, 4, 5]
```

F1

```
String.length("hello")
```

=> 5

F1

```
[1,12,32,12] -- [12,32]
```

```
=> [1, 12]
```

F1

```
String.upcase("Таня")
```

=> "ТАНЯ"

F1

```
if nil, do: "hey", else: "boo"
```

=> "boo"

F1

```
Regex.run ~r{[aeiou]},
  "catepillar"
```

```
=> ["a"]
```

F1

```
byte_size << 1, 2 >>
```

=> 2

F1

```
foo = Stream.unfold(
  "hełło",
  &String.next_codepoint/1
)

Enum.take(foo, 3)
```

```
=> ["h", "e", "ł"]
```

F1

```
f = fn(x, a) -> a <> x end

["alice", "bob"]
|> Enum.reduce(f)
```

```
=> "alicebob"
```

F1

Given the function:

```
foo = fn(a, b) ->
  a + b
end
```

How would you call this function?

```
foo.(1, 2)
```

What is arity?

`arity` is the number of arguments a function accepts.

Example:

```
foo = fn a, b -> a + b end
```

We say function `foo` with arity 2

`foo/2`

F1

How would you run this file?

`sample.exs`

from Terminal:

```
$ elixir sample.exs
```

from iex:

```
iex> c "sample.exs"
```

```
map = %{name: "Mike"}
a = Map.get(map, :name)
b = Map.fetch(map, :name)
a == b
```

```
=> false
```

```
iex> Map.get(map, :name)
"Mike"
```

```
iex> Map.fetch(map, :name)
{:ok, "Mike"}
```

What does `make_ref` return?

An almost unique reference.

Each scheduler thread has its own set of references, and all other threads have a shared set of references. Each set of references consist of $2^{64} - 1$ unique references.

F1

```
"exciting"
|> String.ends_with?("ing")
```

```
=> true
```

F1

```
for n <- 1..3 do
  n + n
end
```

```
=> [2, 4, 6]
```

F1

What module should you include when writing unit tests?

`ExUnit.Case`

Example:

```
defmodule MyTest do
  use ExUnit.Case
...
```

```
for <> do
  <> |> String.upcase
end
```

```
=> ["H", "I"]
```

```
str = "Hello world!"
```

How would you print `str`?

```
IO.puts str
```

How would you create a new regular expression?

```
~r{regexp}
```

F1

What is tail call optimization?

TCO (Tail Call Optimization) is the process by which the compiler can make a call to a function without taking additional stack space.

F1

```
for n <- [1, 2],
    acc <- 1..2 do
  n + acc
end
```

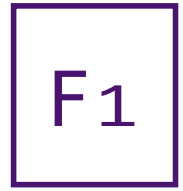
```
=> [2, 3, 3, 4]
```

F1

How would you import the `.flatten` function from the `List` module into your own module?

```
defmodule My do
  import List, only: [flatten: 1]
end
```

`1` is the arity of the function



How can you alias
`Mix.Test.Exercise`
to `Task` within your module?

```
defmodule My do
  alias Mix.Test.Exercise, as Task
end
```

```
an = %{type: "bear", name: "John"}

with type <- Map.get(an, :type),
     name <- Map.get(an, :name),
do: "Hello #{type} #{name}!"
```

```
=> "Hello bear John!"
```

F1

How can you find the current process id?

```
self()
```

F1

Given the function:

```
f = fn -> IO.puts "b" end
```

How would you run this function on a different node?

```
Node.spawn(:"node_name", f)
```

`#PID<7393.48.0>`

What does the `7393` mean?

It indicates that the process is running on a remote node.

`0` - for running on local node

F1

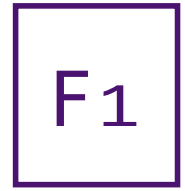
```
[5, 8, 1, -3]
|> Enum.sort
```

```
=> [-3, 1, 5, 8]
```

F1

What are the 4 levels of logging?

debug
info
warn
error



```
"Hello. I am cool."
|> String.replace(".", "!")
```

```
=> "Hello! I am cool!"
```

Which mix task will run ecto migrations?

```
$ mix ecto.migrate
```

Which mix task will list all project dependencies?

```
$ mix deps
```

F1