Elixir Lesson 2018/7/3

アジェンダ

- iexとmixの復習
- モジュールと関数
  - 関数を書いてみる
  - 関数を複数書いてみる
  - 関数から他の関数を呼び出してみる
  - 異なるモジュールの中に定義された関数を呼び出す
  - パイプライン演算子を使う
- 練習問題回答(動物あてクイズ)

このレッスンの所要時間

- 1時間

* iexとmixの復習

- 前回のソースコードをcloneする

```
$ git clone -b lesson-20180626-1 https://github.com/Eigo-Mt-Fuji/elixir-ex-cli-samp
$ cd elixir-ex-cli-sample
$ mix deps.get # パッケージのインストール
$ mix compile # コンパイル
$ mix run sample_cli.ex -vv hello 中尾さん --from 藤川 # 実行
```

モジュールと関数

- 関数を書いてみる
  - https://elixir-lang.org/getting-started/modules-and-functions.html

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

```
end
```

- 関数を複数書いてみる

```
defmodule Math do
...(1)> def sum(a, b)
...(1)> do
...(1)> a + b
...(1)> end
...(1)> def add(a, b) do
...(1)> a + b + 1
...(1)> end
...(1)> end
{:module, Math,
 <<70, 79, 82, 49, 0, 0, 4, 116, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 0, 128,
   0, 0, 0, 15, 11, 69, 108, 105, 120, 105, 114, 46, 77, 97, 116, 104, 8, 95,
   95, 105, 110, 102, 111, 95, 95, 9, 102, ...>>, {:add, 2}}
iex(2)> Math.sum(1,2)
3
iex(3)> Math.add(1,2)
4
```

- 関数から他の関数を呼び出してみる
    - `a + b + 1` -> `sum(a, b) + 1`

```
def sum(a, b) do
  a + b
end

def add(a, b) do
  sum(a, b) + 1
end
```

- `異なるモジュール` の中に定義された関数を呼び出す
    - `モジュール名.関数名` と書くことで呼び出すことができます
    - 例1 (MathUseモジュールのadd関数からMathモジュールのsum関数を呼び出す)

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

defmodule MathUse do
```

```
  def add(a, b) do
    Math.sum(a + b) + 1 # Mathモジュールのsum関数を呼
  end
end

MathUse.add(1, 2)
```

- 例2 (Math, Sub, Calcの３つのモジュールを使った例)

```
iex(1)> defmodule Math do
...(1)> def sum(a, b) do
...(1)> a + b
...(1)> end
...(1)> def add(a, b) do
...(1)> sum(a, b) + 1
...(1)> end
...(1)> end
{:module, Math,
 <<70, 79, 82, 49, 0, 0, 4, 124, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 0, 128,
   0, 0, 0, 15, 11, 69, 108, 105, 120, 105, 114, 46, 77, 97, 116, 104, 8, 95,
   95, 105, 110, 102, 111, 95, 95, 9, 102, ...>>, {:add, 2}}
iex(2)> Math.sum(1,2)
3
iex(3)> Math.add(1,2)
4
iex(4)> defmodule Sub do
...(4)> def sub(a, b) do
...(4)> a - b
...(4)> end
...(4)> def add(a, b) do
...(4)> sub(a, b) -1
...(4)> end
...(4)> end
{:module, Sub,
 <<70, 79, 82, 49, 0, 0, 4, 124, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 0, 127,
   0, 0, 0, 15, 10, 69, 108, 105, 120, 105, 114, 46, 83, 117, 98, 8, 95, 95,
   105, 110, 102, 111, 95, 95, 9, 102, 117, ...>>, {:add, 2}}
iex(5)> Sub.sub(1,2)
-1
iex(6)> Sub.add(1,2)
-2
iex(7)> Math.add(1,2)
4
iex(8)> defmodule Calc do
...(8)> def cal(a, b) do
...(8)> Math.add(a, b) + Sub.add(a ,b)
...(8)> end
...(8)> end
{:module, Calc,
 <<70, 79, 82, 49, 0, 0, 4, 132, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 0, 151,
   0, 0, 0, 17, 11, 69, 108, 105, 120, 105, 114, 46, 67, 97, 108, 99, 8, 95, 95,
   105, 110, 102, 111, 95, 95, 9, 102, ...>>, {:cal, 2}}
```

```
iex(9)> Calc.cal(1,2)
```

- `パイプライン演算子` を使って、関数同士をつなげて実行する
    - https://elixirschool.com/ja/lessons/basics/pipe-operator/#%E4%BE%8B
    - `"Elixir rocks" |> String.split` は `String.split("Elixir rocks")` と同じ
    - より関数を組み合わせて大きなシステムを作る考え方をしたときに、パイプライン演算子を活用することで処理・データが左から順に流れる姿をそのままコードにできるため、より自然な記述が可能

練習プログラム

- 動物あてクイズ