Class Exercise 1: Elixir - Solution for Data Types and Pattern Matching

Instructions

To complete the problems, follow these steps:

- Enter your solution in the code cell below the problem statement, or add a code cell if none is present.
- Run the code to make sure it's correct.
- After finishing all the problems, use the export function to save your work as a .livemd file.
    - Be sure to select the option to include outputs when exporting.
- Upload the saved .livemd file to the assignment on Canvas.

In this exercise, you will need to call functions on Elixir modules. The syntax is straghtforward. Function calls on modules are simply the module name, a period, and the function name. For example, to duplicate a list using the `duplicate` function from the `List` module, you woud use `List.duplicate(list)`.

For some of the exercises below, you will need to use the `List` and `Enum` modules.

- List docs: https://hexdocs.pm/elixir/List.html
- Enum docs: https://hexdocs.pm/elixir/Enum.html
- Map docs: https://hexdocs.pm/elixir/Map.html

Lists

`Lists` are a fundamental data structure in Elixir. Using https://hexdocs.pm/elixir/List.html as a reference if needed, complete the following exercises.

1. Convert the following nested list to a list with no nested elements and then find the 5th element

```
m = [1, 2, 3, [4, 5, 6], [1, 1, 1], [1, 4, 5, [1, 2]]]
f = List.flatten(m)
Enum.at(f, 4)
```

```
5
```

```
# alternate using pipe operator
m =
  [1, 2, 3, [4, 5, 6], [1, 1, 1], [1, 4, 5, [1, 2]]]
  |> List.flatten()
  |> Enum.at(4)
```

```
5
```

2. Show four ways to retrieve the first value from the list `x = 5,6,"seven"` and assign it to the variable `first`.

```
x = [5, 6, "seven"]
first = Enum.at(x, 0)
first = hd(x)
first = List.first(x)
[first | _] = x
[first, _, _] = x
```

```
[5, 6, "seven"]
```

Maps

3. Create a map to hold the following data, assigning it to the variable `mp`:

- name equal to “Fred”
- age equal to 10
- occupation equal to “data scientist and chatGPT expert”

All of the keys in the map should be atoms.

```
# Create the map and assign it to the variable p3
mp = %{name: "Fred", age: 10, occupation: "data scientist and chatGPT expert"}
```

```
%{age: 10, name: "Fred", occupation: "data scientist and chatGPT expert"}
```

4. Use pattern matching to get the name from the map and assign it to the variable `new_name`

```
%{name: new_name} = mp
IO.puts(new_name)
```

```
Fred
```

```
:ok
```

5. Use an appropriate function from the `Map` module to change the name from “Fred” to “Eelke”

```
Map.merge(mp, %{name: "Eelke"})
```

```
%{age: 10, name: "Eelke", occupation: "data scientist and chatGPT expert"}
```

Complex Pattern Matching

The struct in the code block below is from a real Phoenix application with minor modifications to allow it to be used in Livebook. Complete the following problems related to this struct. Remember that in Livebook, the cells following a cell have access to the variables in prior cells so that you do not need to cut and paste the struct for each problem.

```
socket = %{
  id: "phx-F0JQW5smPUxCeAGJ",
  endpoint: MaterialUploadWeb.Endpoint,
  view: MaterialUploadWeb.CSVLive.Index,
  parent_pid: nil,
  root_pid: 111,
  router: "a router",
  assigns: %{
    __changed__: %{},
    action: :new,
    changeset: "changeset",
    data: "some data",
    csv: %{
      id: 100,
      rows: nil,
      inserted_at: nil,
      updated_at: nil
    },
    flash: %{},
    id: :new,
    myself: "me",
    navigate: "/csvs",
    title: "New Csv",
    uploads: %{
      name: "csv_file",
      max_entries: 1,
      max_file_size: 8_000_000,
      entries: [
        %{
          progress: 100,
          preflighted?: true,
          upload_config: :csv_file,
          upload_ref: "phx-F0JQW_ZlHRye0wGp",
          ref: "0",
          uuid: "ccfa14d4-09c3-4faf-84b7-37545a14274d",
          valid?: true,
          done?: true,
          cancelled?: false,
          client_name: "materials.csv",
          client_relative_path: "",
          client_size: 724,
          client_type: "text/csv",
          client_last_modified: 1_675_711_864_050
        }
      ],
      accept: ".csv",
      ref: "phx-F0JQW_ZlHRye0wGp",
      errors: [],
      auto_upload?: false,
      progress_event: nil
    }
  }
}
```

```
%{
  assigns: %{
    __changed__: %{},
    action: :new,
    changeset: "changeset",
    csv: %{id: 100, inserted_at: nil, rows: nil, updated_at: nil},
    data: "some data",
    flash: %{},
    id: :new,
    myself: "me",
    navigate: "/csvs",
    title: "New Csv",
    uploads: %{
      accept: ".csv",
      auto_upload?: false,
      entries: [
        %{
          cancelled?: false,
          client_last_modified: 1675711864050,
          client_name: "materials.csv",
          client_relative_path: "",
          client_size: 724,
          client_type: "text/csv",
          done?: true,
          preflighted?: true,
          progress: 100,
          ref: "0",
          upload_config: :csv_file,
          upload_ref: "phx-F0JQW_ZlHRye0wGp",
          uuid: "ccfa14d4-09c3-4faf-84b7-37545a14274d",
          valid?: true
        }
      ],
      errors: [],
      max_entries: 1,
      max_file_size: 8000000,
      name: "csv_file",
      progress_event: nil,
      ref: "phx-F0JQW_ZlHRye0wGp"
    }
  },
  endpoint: MaterialUploadWeb.Endpoint,
  id: "phx-F0JQW5smPUxCeAGJ",
  parent_pid: nil,
  root_pid: 111,
  router: "a router",
  view: MaterialUploadWeb.CSVLive.Index
}
```

6.a) Use pattern matching to extract and print the id of the csv in the assigns section.

```
# id
%{assigns: %{csv: %{id: id}}} = socket
id
```

```
100
```

6.b) Use pattern matching (and other techniques if needed) to extract the client name in the first entry under the entries key.

```
%{assigns: %{uploads: %{entries: [%{client_name: client_name} | _]}}} = socket
client_name
```

```
"materials.csv"
```