PRADITA
University

IF140303-Web Application Development

Session-01: Introduction to Elixir

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis

What is Elixir?

- Elixir is a dynamic, functional programming language designed for building scalable and maintainable applications.
- It runs on the Erlang VM (BEAM), known for its ability to handle large amounts of concurrent processes.
- Elixir leverages Erlang's capabilities for distributed systems, fault-tolerance, and low-latency messaging.
- Its syntax is inspired by Ruby, making it accessible for developers familiar with Ruby.

How to Install Elixir

- **On macOS:**
  - Use Homebrew: `brew install elixir`
- **On Ubuntu:**
  - Install dependencies: `sudo apt-get install wget gnupg`
  - Add the Elixir repository:

    ```
    wget -qO- https://deb.nodesource.com/setup_16.x | sudo -E bash -
    ```
  - Install Elixir: `sudo apt-get install elixir`
- **On Windows:**
  - Download the installer from the Elixir website:
    `https://elixir-lang.org/install.html`
- Verify the installation by running: `elixir --version`

Module Documentation

```elixir
defmodule Lottery do
@moduledoc """
This module provides functionalities for managing a lottery
    system.
It includes functions for creating, shuffling, checking for
    numbers, and distributing numbers within the lottery pool
    .
"""
```

- `@moduledoc` provides documentation for the `Lottery` module.
- Describes the module's purpose and the functionalities it offers.
- Includes functions for creating, shuffling, checking, and distributing numbers.

Elixir: Lottery Module

```
defmodule Lottery do
def greet do
"Good luck!"
end
```

- The `Lottery` module handles lottery operations such as creating pools, shuffling, and checking numbers.

Greeting Function

```
    def greet do
    "Good luck!"
    end
```

- The `greet/0` function returns a simple greeting message.

Generating Lottery Pool

```
@spec generate_pool() :: [<<_::24, _::_*16>>, ...]
def generate_pool do
numbers = ["Number 1", "Number 2", "Number 3", "Number 4", "
    Number 5", "Number 6"]
pots = ["Pot 1", "Pot 2", "Pot 3", "Pot 4"]

for pot <- pots, number <- numbers do
"#{number} in #{pot}"
end
end
```

- `generate_pool/0` creates a list of lottery numbers across multiple pots.
- The nested `for` comprehensions combine numbers with pot labels.

Shuffling the Pool

```
def randomize(pool) do
Enum.shuffle(pool)
end
```

- `randomize/1` shuffles the list of lottery numbers.
- Utilizes `Enum.shuffle/1` to randomize the order.

Checking Number Presence

```elixir
@spec contains?(any(), any()) :: boolean()
def contains?(pool, number) do
Enum.member?(pool, number)
end
```

- `contains?/2` checks if a number is present in the lottery pool.
- Uses `Enum.member?/2` to verify presence.

Distributing the Pool

```
def distribute(pool, draw_size) do
Enum.split(pool, draw_size)
end
end
```

- `distribute/2` splits the pool into two lists based on the draw size.
- Returns a tuple containing two lists.

```
@doc """
Splits the pool into two parts based on draw_size.

## Parameters

- pool: List of lottery numbers.
- draw_size: Number of items in the first part.

## Returns

- A tuple with two lists: the first with draw_size items,
   and the second with the rest.

## Example

iex> Lottery.distribute(["Number 1 in Pot 1", "Number 2 in
   Pot 2"], 1)
{["Number 1 in Pot 1"], ["Number 2 in Pot 2"]}
"""
```

Function Documentation (2)

- `@doc` provides documentation for the `distribute/2` function.
- Describes the parameters: `pool` and `draw_size`.
- Details the return value: a tuple with two lists.
- Provides an example usage.

Function Name/Number

- In Elixir, `function name/number` refers to the arity of a function.
- The number after the slash (/) indicates the number of arguments the function takes.
- For example:
  - `greet/0` has 0 arguments.
  - `generate_pool/0` has 0 arguments.
  - `contains?/2` has 2 arguments.
  - `distribute/2` has 2 arguments.
- This notation is useful for distinguishing between different functions with the same name but different arity.

What is Functional Programming?

- Functional programming is a paradigm that treats computation as the evaluation of mathematical functions.
- It avoids changing-state and mutable data.
- Functions are first-class citizens, meaning they can be passed as arguments, returned from other functions, and assigned to variables.
- It emphasizes the use of pure functions, which have no side effects and always produce the same output for the same input.

Key Concepts of Functional Programming (1)

- **Immutability:** Data cannot be modified after it is created. Instead, new data structures are created.
- **Pure Functions:** Functions that do not cause side effects and return the same result for the same inputs.
- **Higher-Order Functions:** Functions that can take other functions as arguments or return them as results.
- **First-Class Functions:** Functions are treated as first-class citizens, allowing them to be assigned to variables, passed as arguments, and returned from other functions.

Key Concepts of Functional Programming (2)

- **Declarative Style:** Focuses on what to compute rather than how to compute it, emphasizing expressions and declarations over statements.
- **Recursion:** Functional programming often uses recursion as the primary mechanism for iteration, avoiding traditional loops.