Universidad de los Andes

ISIS-4218 CPD
Elixir
Fecha: February 7, 2022

In this assignment we will practice some Elixir skills. Note that the programs will be evaluated using an automated tester, that means they have to be bulletproof (take into account border cases to test your solution. What happens when I use an empty list or nil, if the values are not in the data structures, etc).

Hand in an independent (`ex`/`exs`) file for each of the exercises (identified with your name and the task number)

Task 1. Implement a function `kind` to determine if a triangle is equilateral, isosceles, or scalene. An equilateral triangle has all three sides the same length. An isosceles triangle has at least two sides the same length. (It is sometimes specified as having exactly two sides the same length, but for the purposes of this exercise we'll say at least two.) A scalene triangle has all sides of different lengths.

Note that for a shape to be a triangle at all, all sides have to be of length ¿ 0, and the sum of the lengths of any two sides must be greater than or equal to the length of the third side. That is, let a, b, and c be sides of the triangle. Then all three of the following expressions must be true:

$$a + b \geq c \tag{1}$$

$$b + c \geq a \tag{2}$$

$$a + c \geq b \tag{3}$$

The function specification is given as

```
@type kind :: :equilateral | :isosceles | :scalene

@doc """
Return the kind of triangle of a triangle with 'a', 'b' and 'c' as
   lengths.
  """
@spec kind(number, number, number) :: {:ok, kind} | {:error, String.t()}
```

Task 2. A robot factory's test facility needs a program to verify robot movements. The robots have three possible movements: (1) `turn right`, (2) `turn left`, (3) `advance`

Robots are placed on a hypothetical infinite grid, facing a particular direction (north, east, south, or west) at a set of $\{x, y\}$ coordinates (*e.g.,* $\{3, 8\}$) with coordinates increasing to the north and east.

The robot then receives a number of instructions, at which point the testing facility verifies the robot's new position, and in which direction it is pointing.

The letter-string (*i.e.,* char sequence) "RAALAL" means: `turn right`, `advance` twice, `turn left`, `advance` once, `turn left` yet again. Say a robot starts at $\{7, 3\}$ facing north. Then running this stream of instructions should leave it at $\{9, 4\}$ facing west.

The complete robot simulator specification is as follows:

```
@doc """
Create a Robot Simulator given an initial direction and position.

Valid directions are: :north, :east, :south, :west
"""
@spec create(direction :: atom, position :: {integer, integer}) :: any

@doc """
Simulate the robot's movement given a string of instructions.

Valid instructions are: "R" (turn right), "L", (turn left), and "A" (
    advance)
"""
@spec simulate(robot :: any, instructions :: String.t()) :: any

@doc """
Return the robot's direction.

Valid directions are: :north, :east, :south, :west
"""
@spec direction(robot :: any) :: atom

@doc """
Return the robot's position.
"""
@spec position(robot :: any) :: {integer, integer}
```

Make sure to use multiple function clauses (if needed) to complete this task. You may add your own private functions to help with the simulation.

Task 3. Given a phrase, count the occurrences of each word in that phrase. For the purposes of this exercise you can expect that a word will always be one of: (1) a number composed of one or more ASCII digits (ie "0" or "1234"), (2) a simple word composed of one or more ASCII letters (ie "a" or "they"), or (3) A contraction of two simple words joined by a single apostrophe (*i.e.,* "it's" or "they're")

When counting words you can assume the following rules: The count is case insensitive (*i.e.,* "You", "you", and "YOU" are 3 uses of the same word) (you have to make sure to count the words correctly) The count is unordered; the tests will ignore how words and counts are ordered Other than the apostrophe in a contraction all forms of punctuation are ignored The words can be separated by any form of whitespace (*i.e.,* `"\t"`, `"\n"`, `" "`)

The function specification is given by:

```elixir
@doc """
Count the number of words in the sentence.

Words are compared case-insensitively.
"""
@spec count(String.t()) :: map
```

Make sure to use the pipe operator `|>` whenever needed

Task 4. Implement a recursive version (but in an iterative process) of the binary search algorithm.

If the remaining array to be searched is empty, then the key cannot be found in the array and a special ”not found” indication is returned.

The algorithm’s specification is given as:

```
@doc """
    Searches for a key in the tuple using the binary search algorithm.
    It returns :not_found if the key is not in the tuple.
    Otherwise returns {:ok, index}.
"""
@spec search(tuple, integer) :: {:ok, integer} | :not_found
```

Task 5. You are running an online fashion boutique. Your big annual sale is coming up, so you need to take stock of your inventory to make sure you're ready. A single item in the inventory is represented by a map, and the whole inventory is a list of such maps in the `BoutiqueInventory` module.

```
%{
  name: "White Shirt",
  price: 40,
  quantity_by_size: %{s: 3, m: 7, l: 8, xl: 4}
}
```

1. Create a new inventory. Implement the `BoutiqueInventory.new/0` function. It should take no arguments and return a new inventory list

2. Add inventory item. Implement the `BoutiqueInventory.add_item/2` function. It receives an inventory and a new item

3. Add and remove sizes item. Implement the `BoutiqueInventory.add_size/4` function. It receives an inventory, an item name, a new size, and a quantity of products for that size. You should return a new inventory with the updated item. Implement the `BoutiqueInventory.remove_size/3` function. It receives an inventory, an item name, and a size. You should return a new inventory with the updated item (without the size).

4. Sort items by price. Implement the `BoutiqueInventory.sort_by_price/1` function. It should take the inventory and return it sorted by item price, ascending.

5. Find all items with missing prices After sorting your inventory by price, you noticed that you must have made a mistake when you were taking stock and forgot to fill out prices for a few items.

   Implement the `BoutiqueInventory.with_missing_price/1` function. It should take the inventory and return a list of items that do not have prices.

6. Increment the item's quantity Some items were selling especially well, so you ordered more, in all sizes.

   Implement the `BoutiqueInventory.increase_quantity/2` function. It should take a single item and a number n, and return that item with the quantity for each size increased by n.

7. Calculate the item's total quantity To know how much space you need in your storage, you need to know how many of each item you have in total.

   Implement the `BoutiqueInventory.total_quantity/1` function. It should take a single item and return how many pieces you have in total, in any size.

Task 6. You're part of a task force fighting against corporate espionage. You have a secret informer at Shady Company X, which you suspect of stealing secrets from its competitors.

Your informer, Agent Ex, is an Elixir developer. She is encoding secret messages in her code.

To decode her secret messages: Take all functions (public and private) in the order they're defined in. For each function, take the first n characters from its name, where n is the function's arity.

1. Turn code into data. Implement the `TopSecret.to_ast/1` function. It should take a string with Elixir code and return its AST.

```
TopSecret.to_ast("div(4, 3)")
# => {:div, [line: 1], [4, 3]}
```

2. Parse a single AST node Implement the `TopSecret.decode_secret_message_part/2` function. It should take an AST node and an accumulator for the secret message (a list). It should return a tuple with the AST node unchanged as the first element, and the accumulator as the second element.

   If the operation of the AST node is defining a function (def or defp), prepend the function name (changed to a string) to the accumulator. If the operation is something else, return the accumulator unchanged.

```
ast_node = TopSecret.to_ast("defp cat(a, b, c), do: nil")
TopSecret.decode_secret_message_part(ast_node, ["day"])
# => {ast_node, ["cat", "day"]}

ast_node = TopSecret.to_ast("10 + 3")
TopSecret.decode_secret_message_part(ast_node, ["day"])
# => {ast_node, ["day"]}
This function doesn't need to do any recursive calls to check the whole
   AST, only the given node. We will traverse the whole AST with built-
   in tools in the last step.
```

3. Decode the secret message part from function definition Extend the `TopSecret.decode_secret_message_part/2` function. If the operation in the AST node is defining a function, don't return the whole function name. Instead, check the function's arity. Then, return only first n character from the name, where n is the arity.

4. Fix the decoding for functions with guards Extend the `TopSecret.decode_secret_message_part/2` function. Make sure the function's name and arity is correctly detected for function definitions that use guards.

```
ast_node = TopSecret.to_ast("defp cat(a, b), do: nil")
TopSecret.decode_secret_message_part(ast_node, ["day"])
# => {ast_node, ["ca", "day"]}

ast_node = TopSecret.to_ast("defp cat(), do: nil")
TopSecret.decode_secret_message_part(ast_node, ["day"])
# => {ast_node, ["", "day"]}
```

```
ast_node = TopSecret.to_ast("defp cat(a, b) when is_nil(a), do: nil")
TopSecret.decode_secret_message_part(ast_node, ["day"])
# => {ast_node, ["ca", "day"]}
```

5. Decode the full secret message Implement the `TopSecret.decode_secret_message/1` function. It should take a string with Elixir code and return the secret message as a string decoded from all function definitions found in the code. Make sure to reuse functions defined in previous steps.

```
code = """
defmodule MyCalendar do
  def busy?(date, time) do
    Date.day_of_week(date) != 7 and
      time.hour in 10..16
  end

  def yesterday?(date) do
    Date.diff(Date.utc_today, date)
  end
end
"""

TopSecret.decode_secret_message(code)
# => "buy"
```