Código:

```elixir
# Actividad 2.2
# Programacion funcional parte 2
# Ricardo Calvo - A01028889
# 02/04/2024

defmodule Hw.Ariel2 do
  def invert(list), do: do_invert(list, [])

  defp do_invert([], res), do: res
  defp do_invert([head | tail], res), do: do_invert(tail, [head | res])

  # 3. The rotate-left function takes two inputs: an integer n and a
list list. It returns the list
  # that results from rotating list a total of n elements to the left.
If n is negative,
  # it rotates to the right.

  def rotate_left(list, n) do
    do_rotate_left(list, n)
  end

  defp do_rotate_left([], _), do: []
  defp(do_rotate_left(list, 0), do: list)

  defp do_rotate_left([head | tail], n) when n > 0 do
    do_rotate_left(tail ++ [head], n - 1)
  end

  defp do_rotate_left(list, n) when n < 0 do
    head = Enum.take(list, -1)
    tail = Enum.drop(list, -1)
    do_rotate_left(head ++ tail, n + 1)
  end

  # 4. The prime-factors function takes an integer n as input (n > 0)
and returns a list containing
  # the prime factors of n in ascending order. Prime factors are prime
numbers that divide a number
  # exactly. If all the prime factors are multiplied, the original
number is obtained.
```

```
def prime_factors(n),
  do:
    do_prime_factors(n, [], [
      2,
      3,
      5,
      7,
      11,
      13,
      17,
      19,
      23,
      29,
      31,
      37,
      41,
      43,
      47,
      53,
      59,
      61,
      67,
      71,
      73,
      79,
      83,
      89,
      97
    ])

defp do_prime_factors(1, res, _), do: invert(res)

defp do_prime_factors(n, res, primes) do
  [p_head | p_tail] = primes

  cond do
    rem(n, p_head) == 0 ->
      do_prime_factors(div(n, p_head), [p_head | res], primes)

    true ->
      do_prime_factors(n, res, p_tail)
  end
end
```

```
  # 8. The pack function takes a list list as input. It returns a list
of lists that group consecutive
  # equal elements.

  def pack(list), do: do_pack(list, [])

  defp do_pack([], res), do: invert(res)

  defp do_pack([head | tail], [group | grouped]) when hd(group) == head
do
    #  IO.inspect(binding())
    do_pack(tail, [[head | group] | grouped])
  end

  defp do_pack([head | tail], grouped) do
    #  IO.inspect(binding())
    do_pack(tail, [[head] | grouped])
  end

  # 10. The encode function takes a list list as input. The consecutive
elements in list are encoded into
  # lists of the form: (n e), where n is the number of occurrences of
the element e.

  def encode(list), do: do_encode(list, [])

  defp do_encode([], res), do: invert(res)

  defp do_encode([head | tail], [{count, n} | rest]) when head == n do
    do_encode(tail, [{count + 1, head} | rest])
  end

  defp do_encode([head | tail], res) do
    do_encode(tail, [{1, head} | res])
  end

  # 12. The decode function takes as input an encoded list list that
has the same structure as the resulting
  # list from the previous problem. It returns the decoded version of
list.

  def decode(list), do: do_decode(list, [])
```

```elixir
  defp do_decode([], res), do: invert(res)

  defp do_decode([{count, value} | rest], res) when count > 1,
    do: do_decode([{count - 1, value} | rest], [value | res])

  defp do_decode([{_, value} | rest], res),
    do: do_decode(rest, [value | res])

  defp do_decode([head | tail], res),
    do: do_decode(tail, [head | res])
end
```

Screenshot

```
calvo@Rick MINGW64 ~/OneDrive/Documentos/TC2037/Homeworks/hw (main)
$ mix test test/ariel_set_2_test.exs
.....
Finished in 0.03 seconds (0.00s async, 0.03s sync)
5 tests, 0 failures

Randomized with seed 184589
```