Sequential Erlang

1. Write three functions. Each function should satisfy one of the following criteria:

    a. Adds two even numbers together
    b. Adds two odd numbers together
    c. Adds two numbers together

2. Write a program which implements FizzBuzz up to 100. A correct program should emit the word 'fizz' if a number is divisible by 3, 'buzz' if the number is divisible by 5, 'fizzbuzz' if the number is divisible by both, or just the number if it is neither divisible by 3 or 5.

3. Implement map() using only Erlang's BIFs. (*Bonus: Implement this in, at most, two lines of code).*

4. Implement fold() using only Erlang BIFs.

5. Write a function which removes the duplicate values from the list [1, 1, 2, 3, 4, 5, 6, 6, a, a, b, c, d, a, 3, 5]. Do not use the lists:filter() function.

Parallel Erlang

1. Re-implement map() as a set of parallel operations. Each function application should occur in a separate process. The result set should be in the same order as the starting set.

2. Write a program which creates three processes. The processes will pass a token around, starting with the first process created and proceeding in creation order. The program should print a '+' each time the token makes a complete circuit. After ten circuits all processes should be cleanly terminated.

3. Write a function start(AnAtom, Fun) to register AnAtom as spawn(Fun). Make sure your program works correctly in the case when two parallel processes simultaneously evaluate start/2. In this case, you must guarantee that one of these processes succeeds and the other fails.

4. Write a program which sings the song "99 bottles of beer". Seek to parallelize as much of the program as possible.

Distributed Erlang

1. Write a program which registers a process using a global name. The process should accept two messages: {add, X, Y} or {subtract, X, Y}. The process should return the resulting value in the message {ok, Value}. If the process receives an unknown message, it should return the message {error, ignored}.

2. Port the token passing example (#2 above) and make it work in a multi-node environment.

3. Port parallel map (#1 above) to work in a multi-node environment. One node should be the master and other nodes should register themselves as workers or slaves. The master node should distribute individual operations in a round-robin basis to each worker node.

"Real World" Erlang Step By Step

1. Write a module which publishes itself using a global name. The program should offer two features: user registration and user authentication. User registration should accept three values: userid, password, and the user's real name. Registrations should be persistent, ie, recorded permanently on disk. The feature should return the atom 'ok' on success or 'error' on failure.

User authentication accepts two values: userid and password. The feature should return the message {ok, RealName} or {error, noauth}.

Bonus: Use mnesia as the persistent data store

Hint: Using gen_server will make this easier

2. Write a module using mochiweb which responds to the URL '/registration'. It should display a web form when it is invoked without any parameters. The web form should contain three fields: userid, password, and real_name. When the registration form is submitted -- when the registration URL is invoked with parameters -- the program should invoke the program created in step #1 to store the registration data.

3. Modify the module in step #2 to respond to a second URL '/authentication'. Like the '/registration' URL, it should display a form when invoked without any parameters. The form should include two fields, userid and password. When the URL is invoked with parameters it should invoke the program created in step #1 to perform the authentication.

If authentication is successful the program should display a HTML page with the phrase "Welcome <real_name>". If authentication fails the program should display the login page again.

4. Write a supervisor process to monitor the authentication and mochiweb modules to restart them when they fail.

Bonus Brain Teaser

Write a counter function that counts from 1 to max but only returns numbers whose digits don't repeat.

For example, part of the output would be:

- 8, 9, 10, 12 (11 is not valid)
- 98, 102, 103 (99, 100 and 101 are not valid)
- 5432, 5436, 5437 (5433, 5434 and 5435 are not valid)

Also:

- Display the biggest jump (in the sequences above, it's 4: 98 -> 102)
- Display the total count of numbers
- Give these two values for max=10000

(Thanks to Cedric Beust)