How is *Erlang* pronounced?

AIR-lang

air rhymes with *fair*.
lang rhymes with *sang*.

Why is Erlang growing in popularity?

Concurrency - Erlang eliminates buggy threads, but tries to make processes as lightweight as possible.

Reliability - Erlang embraces a non-defensive "let it crash" attitude with regard to errors andl offers zero-downtime hot-swapping of code.

What does an actor represent in Erlang?

Contrast this with Scala.

- In Scala in actor represents an object. A thread pool backs the running actors, but actors may have to share the same thread.
- In Erlang an actor represents a lightweight process. Each actor gets its own process.

How do you make comments?

How do you end a statement?

Inline comments begin with a percent sign.

Statements end with a period.

Lists are formed with ... and tuples with ...

What's the difference between them?

... square brackets ... curly brackets.

Tuples are fixed-length.

How are Erlang variables different from variables in imperative languages?

Erlang variables can only be assigned once.

How are mappings usually expressed in Erlang?

Using tuples.

```
> Capital = {capital, {paris, france}}
> {capital, {What, france}} = Capital.
> What.
paris
```

Why do "map" tuples often begin with an initial atom to identify the relationship?

To allow pattern matching to collect all the mappings of a given type.

What is = in Erlang?

Pattern matching, **not** assignment.

What is a key difference between tuples and lists when it comes to pattern matching?

Lists can pattern match against lists of different lengths (using cons). Tuples never can.

How do you prepend to a list?

How do you pattern match on a list?

Using the same syntax:

```
1> A = [1,2].
[1,2]
2> B = [0 | A].
[0,1,2]
3> [First | Rest] = B.
[0,1,2]
4> First.
0
5> Rest.
[1,2]
```

What are Erlang binaries?

A collection-like Erlang feature for specifying the number of bits each variable gets.

```
Bytes = <<A:2, B:6, C:4, D:4>>
```

Means `A` gets 2 bits, `B` gets 6 bits, etc.
(Those variables are already bound).

To extract, use the same syntax:

```
<<ABits:2, BBits:6, CBits:4, DBits:4>> = Bytes
```

What's the conceptual difference between `case` and `if` expressions?

In `case` expressions control flows by pattern matching. In `if` expressions it flows by evaluating Boolean guards.

What function iterates over a list?

What's its downside?

`lists:foreach(fun(El) -> ... end, List)

It results in `ok`, not a useful value.

Why don't `loop()` functions blow up the stack?

They are tail recursive. The last thing any `receive` case should do is call `loop()` afresh. Erlang supports tail-call optimization.

What's the difference between sending and receiving synchronously?

How is each conventionally done?

Receiving synchronously involves `receive` clauses that reply to the sender. By convention the PID of the sender should be the first element of a message tuple.

Send synchronously involves blocking the requester until the reply is ready. By convention the actor module will have a function that takes as its first argument the PID of the instance to send the message to, and the request as the second argument. This function now behaves like a normal, non-blocking function, even though its actually executes in a different process by sending an asynchronous message to the provided PID.

How can you check if a process is alive?

```
erlang:is_process_alive(Pid).
```

How can you create a monitoring process that restarts dead processes?

Create an actor such that:
1) The first thing its `loop` function does is trap exit.
2) Write a `receive` case for creating a process, and registering it with a meaningful atom name.
3) Write a `receive` case for exits which sends a signal to the other case.