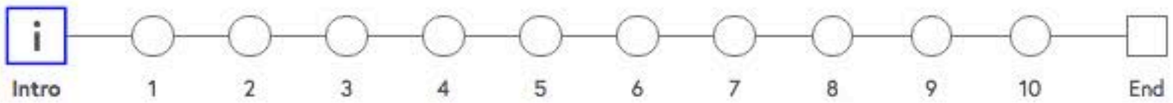
Erlang programs and lists



In this test you'll be asked questions that cover the work we've looked at so far: creating simple Erlang functions and modules over numbers, booleans, tuples and lists.

TEST RULES AND GRADING

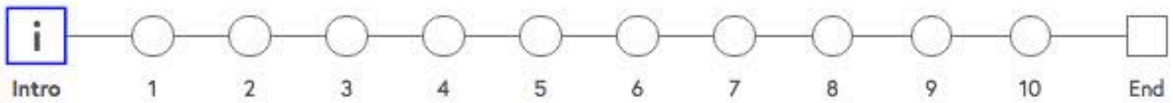
- You may take 3 attempts to answer each question
- Each question has 3 points available
- A point will be deducted for each incorrect attempt
- You can review your total score for the test at the end
- If you want to buy a Certificate of Achievement for this course, you will need to score an average of 70% or above on all tests
- You cannot repeat a test to improve your score
- You can check your average test score for this course so far on your Progress page

Begin test

HOW ARE YOU GETTING ON WITH ERLANG?
DISCUSSION

SKIP TEST
GO TO STEP 2.25

Erlang programs and lists



In this test you'll be asked questions that cover the work we've looked at so far: creating simple Erlang functions and modules over numbers, booleans, tuples and lists.

TEST RULES AND GRADING

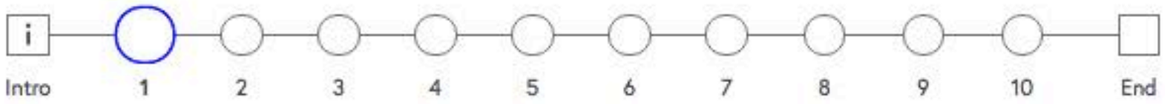
- You may take 3 attempts to answer each question
- Each question has 3 points available
- A point will be deducted for each incorrect attempt
- You can review your total score for the test at the end
- If you want to buy a Certificate of Achievement for this course, you will need to score an average of 70% or above on all tests
- You cannot repeat a test to improve your score
- You can check your average test score for this course so far on your Progress page

Begin test

HOW ARE YOU GETTING ON WITH ERLANG?
DISCUSSION

SKIP TEST
GO TO STEP 2.25

Erlang programs and lists



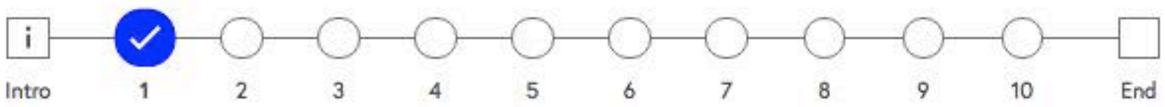
Question 1

Which of the following `fun` expressions does *not* implement the `nand` function on a pair of booleans?

- ○ `fun (A,B) -> (not(A) or not(B)) end`
- ○ `fun (A,B) -> not(A and B) end`
- ○ `fun (A,B) -> not(A andalso B) end`
- ○ `fun (A,B) -> (not(A) and B) or (not(B) and A) end`

Tries left: 3

Erlang programs and lists



Question 1

Which of the following `fun` expressions does *not* implement the `nand` function on a pair of booleans?

- `fun (A,B) -> (not(A) or not(B)) end`
- `fun (A,B) -> not(A and B) end`
- `fun (A,B) -> not(A andalso B) end`
- **`fun (A,B) -> (not(A) and B) or (not(B) and A) end`**

Correct

You scored: 2

Simon Thompson LEAD EDUCATOR

Question 2

Which of the following statements about this Erlang shell command is correct?

```
{a,A} = {B,A}
```

- ◯ This will fail because it is not possible to match atoms in a pattern match.
- ◯ This will fail if `A` is not already bound.
- ◯ This will fail because it is not possible to re-assign to the variable `A`.
- ◯ It is impossible to match an atom with a variable.

Tries left: 3

PREVIOUS QUESTION

SKIP QUESTION

Which of the following statements about this Erlang shell command is correct?

```
{a,A} = {B,A}
```

- This will fail because it is not possible to match atoms in a pattern match.
- **This will fail if `A` is not already bound.**
- This will fail because it is not possible to re-assign to the variable `A`.
- It is impossible to match an atom with a variable.

Correct

You scored: 3

Simon Thompson LEAD EDUCATOR

That's right. It will also fail if B is not already bound, but it will succeed if both A and B are bound.

PREVIOUS QUESTION

NEXT QUESTION

Question 3

Which of the following statements is false?

- ◯ It is not possible to match functions to variables.
- ◯ It is possible to use floating point numbers on the left hand side of pattern matches.
- ◯ It is not possible to use `fun` expressions on the left hand side of pattern matches.
- ◯ Strings can be used on the left hand sides of patterns.

Tries left: 3

PREVIOUS QUESTION

SKIP QUESTION

Question 3

Which of the following statements is false?

- [x] **It is not possible to match functions to variables.**
- [ ] It is possible to use floating point numbers on the left hand side of pattern matches.
- [ ] It is not possible to use `fun` expressions on the left hand side of pattern matches.
- [ ] Strings can be used on the left hand sides of patterns.

Correct

You scored: 2

Simon Thompson LEAD EDUCATOR

That's right. Functions *can* be matched with variables, but functions can't occur on the left hand side of patterns, unlike floating point numbers, strings and other (non-function) data.

Question 4

What is wrong with the following definition of an Erlang function, intended to check when its two arguments are equal?

```
square(X,Y) ->
    false;
square(X,X) ->
    true.
```

- ◯ The first clause should end with a comma.
- ◯ The first clause will match every case.
- ◯ Function definitions cannot contain repeated variables.
- ◯ `square` is a built-in function.

Tries left: 3

PREVIOUS QUESTION

SKIP QUESTION

Question 4

What is wrong with the following definition of an Erlang function, intended to check when its two arguments are equal?

```
square(X,Y) ->
    false;
square(X,X) ->
    true.
```

- The first clause should end with a comma.
- **The first clause will match every case.**
- Function definitions cannot contain repeated variables.
- `square` is a built-in function.

Correct

You scored: 3

Simon Thompson LEAD EDUCATOR

That's right. And this is a problem: for the `X,X` case to have an effect it needs to be the first clause, as otherwise the `X,Y` case will match all arguments.

Question 5

Which of the following statements about the `merge` function is true?

```
merge([],Ys) -> Ys;
merge(Xs,[]) -> Xs;
merge([X|Xs],[Y|Ys]) when X<Y ->
    [ X | merge(Xs,[Y|Ys]) ];
merge([X|Xs],[Y|Ys]) when X>Y ->
    [ Y | merge([X|Xs],Ys) ];
merge([X|Xs],[Y|Ys]) ->
    [ X | merge(Xs,Ys) ].
```

- ◯ The length of `merge(Xs,Ys)` is the sum of the lengths of `Xs` and `Ys`.
- ◯ If `Xs` and `Ys` are sorted into ascending order, then so is `merge(Xs,Ys)`.
- ◯ If neither of the lists `Xs` and `Ys` contains duplicates, then neither does `merge(Xs,Ys)`.

Tries left: 3

PREVIOUS QUESTION

SKIP QUESTION

Support

Question 5

Which of the following statements about the `merge` function is true?

```
merge([],Ys) -> Ys;
merge(Xs,[]) -> Xs;
merge([X|Xs],[Y|Ys]) when X<Y ->
    [ X | merge(Xs,[Y|Ys]) ];
merge([X|Xs],[Y|Ys]) when X>Y ->
    [ Y | merge([X|Xs],Ys) ];
merge([X|Xs],[Y|Ys]) ->
    [ X | merge(Xs,Ys) ].
```

- The length of `merge(Xs,Ys)` is the sum of the lengths of `Xs` and `Ys`.
- **If `Xs` and `Ys` are sorted into ascending order, then so is `merge(Xs,Ys)`.**
- If neither of the lists `Xs` and `Ys` contains duplicates, then neither does `merge(Xs,Ys)`.

Correct

You scored: 3

Simon Thompson LEAD EDUCATOR

This is true: ordered lists are merged by this program.

Question 6

What is the result of evaluating `foo(0,[4,0,1])` when `foo` is defined like this?

```
foo(_,[])              -> [];
foo(Y,[X|_]) when X==Y -> [X];
foo(Y,[X|Xs])          -> [X | foo(Y,Xs) ].
```

- ○ `[4]`
- ○ `[4,1,0]`
- ○ `[4,0]`
- ○ `[4,1]`

Tries left: 3

PREVIOUS QUESTION

SKIP QUESTION

Question 6

What is the result of evaluating `foo(0,[4,0,1])` when `foo` is defined like this?

```
foo(_,[])              -> [];
foo(Y,[X|_]) when X==Y -> [X];
foo(Y,[X|Xs])          -> [X | foo(Y,Xs) ].
```

- [ ] `[4]`
- [ ] `[4,1,0]`
- [x] `[4,0]`
- [ ] `[4,1]`

Correct

You scored: 3

Simon Thompson LEAD EDUCATOR

Yes, well done! In general the function cuts off the list after the first occurrence of the first argument, in this case 0.

Question 7

The following function is supposed to remove all occurrences of the first argument in the second argument, which is a list. But there are a few errors in the definition.

```
bar (N, [Y]) ->
        [Y];

bar (N, [N]) ->
        [];

bar (N, [Y|Ys]) when N =/= Y ->
        [Y|bar (N, Ys)];

bar (N, [Y|Ys]) ->
        Ys.
```

Three of the following corrections need to be made. Which one of the possible corrections does **not** need to be made?

- ◯ The second clause should precede the first clause.
- ◯ A case for the empty list needs to be added.
- ◯ There needs to be a recursive call to `bar` in the fourth clause.
- ◯ The third clause should test for == rather than =/= in the guard

Tries left: 3

Question 7

The following function is supposed to remove all occurrences of the first argument in the second argument, which is a list. But there are a few errors in the definition.

```
bar (N, [Y]) ->
        [Y];

bar (N, [N]) ->
        [];

bar (N, [Y|Ys]) when N =/= Y ->
        [Y|bar (N, Ys)];

bar (N, [Y|Ys]) ->
        Ys.
```

Three of the following corrections need to be made. Which one of the possible corrections does **not** need to be made?

- The second clause should precede the first clause.
- A case for the empty list needs to be added.
- There needs to be a recursive call to `bar` in the fourth clause.
- **The third clause should test for == rather than =/= in the guard**

Correct

You scored: 1

Question 8

Which of the following statements is true of the baz function?

```
baz([])     -> [];
baz([X|Xs]) -> [X | baz(zab(X,Xs))].

zab(N,[])     -> [];
zab(N,[N|Xs]) -> zab(N,Xs);
zab(N,[X|Xs]) -> [X | zab(N,Xs)].
```

- ◯ The length of the output list is the same as the length of the input list.
- ◯ The output list contains the last occurrence of each element in the input list.
- ◯ The output list contains the first occurrence of each element in the input list.
- ◯ The result list is sorted.

Tries left: 3

PREVIOUS QUESTION

SKIP QUESTION

Question 8

Which of the following statements is true of the `baz` function?

```
baz([])     -> [];
baz([X|Xs]) -> [X | baz(zab(X,Xs))].

zab(N,[])     -> [];
zab(N,[N|Xs]) -> zab(N,Xs);
zab(N,[X|Xs]) -> [X | zab(N,Xs)].
```

- The length of the output list is the same as the length of the input list.
- The output list contains the last occurrence of each element in the input list.
- **The output list contains the first occurrence of each element in the input list.**
- The result list is sorted.

Correct

You scored: 3

Simon Thompson LEAD EDUCATOR

Yes, that' right!

Question 9

Which of the following statements is true?

- Every element of a list needs to be of the same type, as in the list of numbers, `[1,2,3]`.
- **To pattern match a tuple, the match will either match the tuple to a single variable, or will have to match a pattern to every element of the tuple.**
- The `length` function can be used to find the size of a tuple.
- Strings in Erlang are simply a way of viewing tuples of numbers.

Correct

You scored: 3

Simon Thompson LEAD EDUCATOR

Yes, that's true, in contrast to a (non-empty) list, which be matched to `[X|Xs]`, in which `Xs` is matched to the 'tail' of the list, that is the list except for the first element (which is matched in this pattern to `X`).

Question 10

Which of the following statements is true?

- [x] **Patterns in the different clauses of a function definition can overlap.**
- [ ] Patterns in the different clauses of a function must be *exhaustive*, and cover every possible input.
- [ ] Different clauses of a function definition can take different numbers of parameters.
- [ ] Functions cannot be called in guards.

Correct answer

You scored: 0

Simon Thompson LEAD EDUCATOR

Yes, that's right, and the first clause that matches the given parameters is the one that is used.

Back to question

2.24

2 MORE STEPS TO GO

Erlang programs and lists

Summary

You've scored 23 out of 30

Moving on from the test we'll round off Week 2 with a summary of what we've learned so far, together with a couple more programming challenges that you might like to try for more practice.

PREVIOUS QUESTION

WEEK TWO SUMMARY VIDEO