

3. Lexical scoping is a crucial part of code execution in many programming languages, including ML. For each statement below, check the box if and only if the statement is true regarding this ML code. Consider each statement after the identified line is executed.

```
1
    1-
        val x = 50
2
    2- val y = 3
 3
    3- val z = 10
   4- val f = fn z \Rightarrow z
4
    5- val a =
6 6- let
7
   7-
          va1 x = 3*x
8 8-
         val z = y*z
9
   9-
        in
10
   10-
          x*z
11
   11- end
   12- fun f x z = x + y + z
12
13
   13 -
```

3. Lexical scoping is a crucial part of code execution in many programming languages, including ML.

For each statement below, check the box if and only if the statement is true regarding this ML code.

Consider each statement after the identified line is executed.

```
val x = 50
    1-
2
    2-
        val y = 3
3
    3-
        val z = 10
    4-
        val f = fn z \Rightarrow z
    5-
        val a =
    6- let
6
         val x = 3*x
    7-
8
    8-
          val z = y*z
   9- in
10
    10-
         x*z
   11- end
11
12 12- fun f x z = x + y + z
13
    13 -
```

- $\hfill \square$ On line 4, the variable z inside the function body is bound to 10.
- On line 7, x is bound to 150.

```
✓ 正确
```

On line 8, z is bound to 30.

```
✓ 正确
```

- On line 10, z is bound to 10.
- \square On line 12, the variable x inside the function body is bound to 50.
- ightharpoonup On line 12, the variable m y inside the function body is bound to 3.

```
✓ 正确
```

On line 13, x is bound to 50.

```
✓ 正确
```

4. For each type below, check the box if and only if the type is a valid type for the function foo. Do not only select the most general type, also select less general types.

10/10分

```
1 fun foo f x y z =
2 if x >= y
3 then (f z)
4 else foo f y x (tl z)
```

- [(int -> real) -> int -> int -> real
- ✓ (string list -> bool list) -> int -> int -> string list -> bool list

```
✓ 正确
```

✓ ('a list -> 'b list) -> int -> int -> 'a list -> 'b list

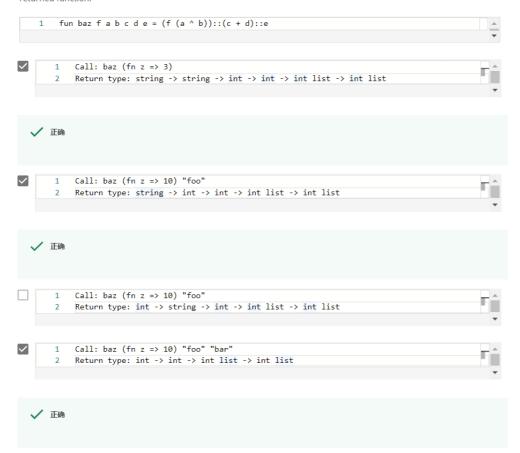
```
✓ 正确
```

- $\hfill \Box$ (int list -> 'b list) -> int -> int -> 'b list -> int list
- $\hfill\Box$ ('a list -> string list) -> int -> int -> 'a list -> 'a option list

5. Several correct implementations of the factorial function appear below. Check the box next to a definition if and only if all recursive functions calls (possibly including recursive helper functions) are tail calls.

```
fun factorial i =
  if i = 0
  then 1
              else i * factorial (i - 1)
~
            fun factorial i =
       3
              fun factorialhelper (accum,i) =
        4
              if i = 0
        5
                then accum
               else factorialhelper (accum*i, i-1)
        6
            in
            factorialhelper (1,i)
            end
  ✓ 正确
            fun factorial i =
        1
        2
            let
             fun factorialhelper (start,i) =
              if start <> i
then start * factorialhelper (start+1, i)
                else start
            if i=0
        8
       9
             then 1
            else factorialhelper (1,i)
      10
       11
            end
            fun factorial i =
       2
             case i of
              0 => 1
| x => x * factorial (i-1)
```

Given the curried function below, check the box if and only if the given function call is paired with a correct type for the returned function.



Consider the two functions maybeEven and maybeOdd below, which are mutually recursive. For each statement below,
check the box if and only if the statement is true regarding this ML code. Notice that these functions have some
unconventional behaviour.

18/18分

	1 ft	un maybeEven x =	
	2	if $x = 0$	
	3	then true	
	4	else	
	5	if $x = 50$	
	6	then false	
	7	else maybeOdd (x-1)	
	8		
		nd maybeOdd y =	
	10	if $y = 0$	
	11	then false	
	12	else	
	13	if $y = 99$	
	14	then true	
	15	else maybeEven (y-1)	ш
✓		ution of the call maybeEven 50 requires 25 calls to maybeOdd.	
	✓ 正確	·	
	The ca	Il maybeEven 1 does not terminate.	
~	Evalua	ation of the call maybeOdd 6 requires 3 calls to maybeEven.	
	✓ 正確	·····································	
~	Every o	call from maybeEven to maybeOdd or from maybeOdd to maybeEven is a tail call.	
	✓ 正確	·····································	
	Evalua	sting any call to maybeEven will always involve a call to maybeOdd.	
~	The fur	nctions maybeEven and maybeOdd have the same type.	
	✓ 正確	·····································	
	For inp	put x > 50, maybeEven always returns false.	

 $\begin{tabular}{ll} \hline & The return types of {\tt maybeEven} \ and {\tt maybeOdd} \ are \ different. \\ \hline \end{tabular}$

8.	The next three questions, including this one, relate to this situation: Types are often abstract representations for real world values. For each problem below, decide which type is the best choice to represent the given data.	3/3 分
	This problem: Values of the type will represent multiple country names.	
) int	
	○ string	
	O int list	
	<pre>string list</pre>	
	(string * int) list	
	✓ 正确	
9.	This problem: Values of the type will hold a person's last name.	3/3 分
) int	
	<pre> string </pre>	
	int list	
	O string list	
	○ (string * int) list	
	✓ 正确	
10.	This problem: Values of the type will hold a collection of student names and their grades on an assignment.	3/3 分
	○ int	
	○ string	
	int list	
	O string list	
	⑥ (string * int) list	
	✓ 正确	

11. The next 5 questions, including this one, are similar. Each question uses a slightly different definition of an ML signature DIGIT with the same structure definition Digit below. The Digit structure implements one-digit numbers that wrap around when you increment or decrement them.

```
1  structure Digit :> DIGIT =
2  struct
3  type digit = int
4  exception BadDigit
5  exception FailTest
6  fun make_digit i = if i < 0 orelse i > 9 then raise BadDigit else i
7  fun increment d = if d=9 then 0 else d-1
8  fun decrement d = if d=0 then 9 else d-1
9  val down_and_up = increment o decrement (* recall o is composition *)
10  fun test d = if down_and_up d = d then () else raise FailTest
11  end
12
```

In each problem, the definition of DIGIT matches the structure definition Digit, but different signatures let clients use the structure in different ways. You will answer the same question for each DIGIT definition by choosing the best description of what it lets clients do.In this question, the definition of DIGIT is:

```
signature DIGIT =
sig
type digit = int
val make_digit : int -> digit
val increment : digit -> digit
val decrement : digit -> digit
val down_and_up : digit -> digit
val test : digit -> unit
end
```

- The type-checker prevents the client from calling <code>Digit.test</code> with the expression <code>Digit.test</code> e, for any expression e that evaluates to a value v.
- There are calls by clients to Digit.test that can type-check, but Digit.test 10 does not type-check.
- The client call Digit.test 10 type-checks and causes the Digit. FailTest exception to be raised.
- The client call Digit.test 10 type-checks and evaluates without raising an exception.



```
1 signature DIGIT =
2 sig
3 type digit = int
4 val make_digit : int -> digit
5 val increment : digit -> digit
6 val decrement : digit -> digit
7 val down_and_up : digit -> digit
8 end
9
```

- (a) The type-checker prevents the client from calling Digit.test with the expression Digit.test e, for any expression e that evaluates to a value v.
- $\bigcirc \ \, \text{There are calls by clients to } \text{Digit.test that can type-check, but } \text{Digit.test } \ 10 \ \text{does not type-check.}$
- O The client call Digit.test 10 type-checks and causes the Digit.FailTest exception to be raised.
- The client call Digit.test 10 type-checks and evaluates without raising an exception.



13. In this question, the definition of <code>DIGIT</code> is:

3/3 分

```
1 signature DIGIT =
2 sig
3 type digit = int
4 val make_digit : int -> digit
5 val increment : digit -> digit
6 val decrement : digit -> digit
7 val test : digit -> unit
8 end
```

- O The type-checker prevents the client from calling Digit.test with the expression Digit.test e, for any expression e that evaluates to a value v.
- There are calls by clients to Digit.test that can type-check, but Digit.test 10 does not type-check.
- The client call Digit.test 10 type-checks and causes the Digit.FailTest exception to be raised.
- The client call Digit.test 10 type-checks and evaluates without raising an exception.

✓ 正确

14. In this question, the definition of DIGIT is:

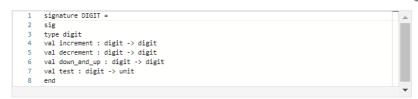
1 signature DIGIT =
2 sig
3 type digit
4 val make_digit : int -> digit
5 val increment : digit -> digit
6 val decrement : digit -> digit
7 val down_and_up : digit -> digit
8 val test : digit -> unit
9 end

- $\bigcirc \ \, \text{The type-checker prevents the client from calling Digit.test with the expression Digit.test} \ \, \text{e, for any expression e that evaluates to a value } v.$
- ① There are calls by clients to Digit.test that can type-check, but Digit.test 10 does not type-check.
- $\begin{tabular}{ll} \hline \end{tabular} \begin{tabular}{ll} The client call Digit.test 10 & type-checks and causes the Digit.FailTest exception to be raised. \\ \hline \end{tabular}$
- O The client call Digit.test 10 type-checks and evaluates without raising an exception.

✓ 正确

15. In this question, the definition of <code>DIGIT</code> is:

3/3分



- $\textcircled{\textbf{0}} \quad \text{The type-checker prevents the client from calling Digit.test with the expression Digit.test} \quad \text{e, for any expression e that evaluates to a value } \textbf{v}.$
- There are calls by clients to Digit.test that can type-check, but Digit.test 10 does not type-check.
- The client call Digit.test 10 type-checks and causes the Digit.FailTest exception to be raised.
- The client call Digit.test 10 type-checks and evaluates without raising an exception.

✓ 正确

3/3 分