Dashboard / My courses / FP (2022/2023 - R.R. Slavescu) / 19 December - 25 December / Haskell test - Labs 8 - 10 (30434/2)

Started on	Tuesday, 20 December 2022, 12:11 PM
State	Finished
Completed on	Tuesday, 20 December 2022, 12:24 PM
Time taken	12 mins 31 secs
Grade	**5.00** out of 10.00 (**50**%)

Information

The next section contains basic questions.

Read each question carefully.

Question **1**

Correct

Mark 1.00 out of 1.00

Please select True to receive the default mark.

Select one:

● True ✓

○ False

Question **2**

Incorrect

Mark 0.00 out of 1.00

Which of the following are examples of **not valid** ways to create local definitions in Haskell?

□ a. x + 1 with x = 2

□ b. local x = 2 in x + 1

□ c. x + 1 where x = 2

☑ d. let x = 2 in x + 1 ✗

Your answer is incorrect.

Question **3**

Correct

Mark 1.00 out of 1.00

Select the function that uses pattern guards correctly to implement the filter function:

○ a.
```
filter _ [] = []
filter p (x:xs) =
  if p x then x:filter p xs
  otherwise filter p xs
```

○ b.
```
filter _ [] = []
filter p (x:xs)
  | p x -> x:filter p xs
  | else -> filter p xs
```

● c. ✓
```
filter _ [] = []
filter p (x:xs)
  | p x = x:filter p xs
  | otherwise = filter p xs
```

Your answer is correct.

Question **4**

Incorrect

Mark 0.00 out of 1.00

Given the following function definition:

```
f :: [String] -> Int
f ["a", "b"] = 1
f ["a", _] = 2
f ("a":_) = 3
f ["a", "b", "c"] = 4
```

the result of the following function call is:

```
f ["a", "b", "c"]
```

Answer: 2 ✗

Question **5**

Correct

Mark 1.00 out of 1.00

Which function describes best the each of the following list comprehensions?

[x * 3 | x <- xs]	map	✓
[x | x <- xs, x > 3]	filter	✓
[x + 2 | x <- xs]	map	✓

Your answer is correct.

Information

The next 3 questions are intermediate questions.

Read each question carefully.

Question **6**

Incorrect

Mark 0.00 out of 1.00

Select all the **false** statements about the bottom value:

☑ a. In Haskell, None is the bottom value ✓

☑ b. The bottom value can be assigned to any type ✗

□ c. In Haskell, Nothing is the bottom value

□ d. Evaluating the bottom value at runtime will crash the program

Your answer is incorrect.

Question **7**

Correct

Mark 1.00 out of 1.00

The following list comprehension:

```
[(x, y) |x <- [1, 2], y <- ['a', 'b']]
```

○ a. Fails to combile because the syntax is invalid

○ b. Returns [(1, 'a'), (2, 'b')]

○ c. Fails to combile because x and y have different types

● d. Returns: [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')] ✓

Your answer is correct.

Question **8**

Partially correct

Mark 0.50 out of 1.00

Select all the **true** statements about type classes

□ a. All type class implementations for a data type must be in the module where the data is defined

□ b. Type classes are used to define a common interface for a set of operations that can be performed on various types

☑ c. Any type class can be implemented for any type ✓

□ d. Type classes are used to organize related types in a file

Your answer is partially correct.

You have correctly selected 1.

Information

The next 2 questions are advanced questions.

Read each question carefully.

Question **9**

Incorrect

Mark 0.00 out of 1.00

Given the following code that generates the hamming numbers:

```
merge3 x y z = merge (merge x y) z where
    merge (u:us) (v:vs)
        | u < v = u:merge us (v:vs)
        | u > v = v:merge (u:us) vs
        | otherwise = u:merge us vs
```

```
ham :: [Integer]
ham = 1:merge3 ham2 ham3 ham5
```

```
ham2 = [ 2*i | i <- ham ]
```

```
ham3 = [ 3*i | i <- ham ]
```

```
ham5 = [ 5*i | i <- ham ]
```

```
hammingGen :: Int -> [Integer]
hammingGen n = take n ham
```

Select what will be printed for each of the following commands after evaluating:

```
hammingGen 3
```

```
> :sprint ham2
ham2 =
```

3 : _ ✗

```
> :sprint ham3
ham3 =
```

2 : 4 : _ ✗

Your answer is incorrect.

Question **10**

Partially correct

Mark 0.50 out of 1.00

Given the following code:

```
newtype All = All Bool
instance Semigroup All where
   (All a) <> (All b) = All (a && b)
instance Monoid All where
   mempty = All True
```

The result of the following expressions is:

`foldl (<>) mempty (map (\x -> All (x >= 2)) [2, 3, 4])`	All True	✓
``foldl (<>) mempty (map (\x -> All (x `mod` 2 == 0)) [1, 2, 3])``	True	✗

Your answer is partially correct.

You have correctly selected 1.

◄ Haskell test - Labs 8 - 10 (30433/2)

Jump to...

Test 1 Haskell - NR 1 - Gr. 30434/1 ►