1. Select the function that uses pattern guards correctly to implement the dropWhile function:

a) dropWhile \_ [] = []

dropWhile p (x:xs)

| p x = dropWhile p xs

| otherwise = x : xs

A screenshot of a computer program

Description automatically generated

2) Select the function that uses pattern guards correctly to implement the difference function"

c) difference (a : as) b

| a 'elem' b = difference as b

| otherwhise = a : difference as b

A screenshot of a computer program

Description automatically generated

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

filter \_ [] = []

filter p (x:xs)

| p x = x : filter p xs

| otherwise = filter p xs

4) Given the following code:

newtype Any = Any Bool

instance Semigroup Any where

(Any a) <> (Any b) = Any (a || b)

instance Monoid Any where

mempty = Any False

The result of the following expression is:

\* foldl (<>) mempty (map (\x -> Any (x `mod` 2 == 0)) [1, 2, 3])

\* foldl (<>) mempty (map (\x -> Any (x >= 2)) [2, 3, 4])

R: Any True (both)

Explanation:

map (\x -> Any (x `mod` 2 == 0)) [1, 2, 3] will create a list of 'Any' values based on whether

each element in the list is even (True if even, False if odd)

Applying the foldl (<>) mempty to this list with the <> operation defined for 'Any' will

perform a left-fold using the || operation

The result is 'Any True', because at least one elem in the list os even

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

[x + 1 | x <- xs] R: map

[x \* 2 | x <- xs] R: map

[x | x <- xs, x `mod` 2 == 0] R: filter

6) The following list comprehension:

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

R: fails to combine because the syntax is invalid

(in loc de ; e , ca sa mearga, si asa ar fi [('a',1), ('a',2), ('b',1), ('b', 2)]

7) 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 `mod` 2 == 0)) [1, 2, 3]) R: All False

\* foldl (<>) mempty (map(\x -> All (x >= 2)) [2, 3, 4]) R: All True

8) The following list comprehension:

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

R: d) Returns [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]

9) Select all the true statements about type classes:

a) We can implement type classes defined by the standard library for our own types

b) Type classes are used to abstract common behavior for various types (like Java interfaces)

Incorecte!!:

c) Type classes are used to define classes, types that also have methods and private fields

d) All type class implementations for a data type must be in the module where the data is defined

10) Select the snippets that are NOT VALID Haskell code (i.e. will fail to compile)

R: a) dec : Num a => a -> a

dec a = a - 1

d) len l = case l of

[] -> 0

(\_::xs) -> 1 + len xs

Variantele VALIDE:

b) dec :: Num a => a -> a

dec a = a - 1

c) len l = case l of

[] -> 0

(\_:xs) -> 1 + len xs

11) Which of the following are examples of VALID ways to create local definitions in Haskell?

R: c) let y = 5 in y \* 1

d) y \* 2 where y = 5

INCORECTE:

a) a\_ local y = 5 in y \* 2

b) y \* 2 with y = 5

12) Given the following function definition:

f :: [Int] -> Int

f[1,2] = 1

f[\_,\_] = 2

f[3,4] = 3

the result of the following function call is:

f[3, 4] => R: 2 (intra pe al doilea branch)

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

\* [x | x <- xs, x `elem` ['a'..'z']] R: filter

\* [Char.toUpper x | x <- xs] R: map

\* [take 2 x | x <- xs] ?? take( cred ca map)

14) Select all the FALSE statements about the bottom value:

a) In Haskell, None is the bottom value

d) In Haskell, Nothing is the bottom value

15) Select all the TRUE statements about the bottom value:

b) In Haskell, undefined is the bottom value

d) It crashes the program if it's evaluated at runtime

\* The bottom value can be assigned to any type

16) Given the following code:

newtype Any = Any Bool

instance Semigroup Any where

(Any a) <> (Any b) = Any (a || b)

instance Monoid Any where

mempty = Any False

The result of the following expressions is:

\* foldl (<>) mempty (map (\x -> Any (Char.isLower x)) "Hello") R: Any True

\* foldl (<>) mempty (map Any []) R: Any True

17)

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

hammingGen 4

> :sprint ham2

ham2 = 2 : 4 : \_

>:sprint ham5

ham5 = 5 : \_

18) Select all the TRUE statements about type classes:

b) Any type class can be implemented for any type

19) Select all the TRUE statements about the bottom value:

a) In Haskell, undefined is the bottom value

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

[x | x <- xs, length x > 2] R: filter

[drop 2 x | x <- xs] R: take

[not x | x <- xs] R: filter

21) Select all the FALSE statements about type classes:

a) Type classes are used to define classes, a special kind of data definition thet includes methods

and private fields

c) All instance implementations for a type class must be in the module where the type class is

22) 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 ham3

ham3 = 3 : \_

>: sprint ham2

ham2 = 2 : 4 : \_

23) Select the FALSE statements about monads in Haskell:

a) Monad defines the function <$>

b) Int is an example of Monad

24) Which of the following names would best describe the following parser:

satisfies (==c)

R: d) char

25) Select the function signature that best represents a parser

R: a) String -> Result ParseError (a, String)

26) Complete the parser below such that it parses a C\C++ array indexing expression (i.e [1][2]):

cArrayIndex = some $ between char `[` char `]` number

27) Match the concepts:

Mappable types that can also unpack nested structures in results - MONAD

Mappable types - FUNCTOR

Generalized mappable types - APPLICATIVE

28) Select all the FALSE statements about Input/Output in Haskell

a) To read data from a file we use the read function

c) do notation can be only used with the IO monad

CORECTE SUNT:

b) To obtain a line from the standard input, we can write

do

name <- getLine

putStrLn name

d) Haskell's main function has signature main :: IO ()

29) Select the FALSE statements about monads in Haskell:

a) Monad defined the function <$>

b) Int is an example of Monad

CORECTE: b) Maybe is an example of a Monad, d) Monad is a type class

30) Given the following function definition:

f :: [String] -> Int

f["x", "y"] = 1

f["x", "y", "z"] = 2

f["x", \_] = 3

f("x":\_) = 4

f \_ = 5

f["x", "z", "y"] -> R: 4

31) Select all the FALSE statements about the bottom value:

a) In Haskell, None is the bottom value

b) If an expression has the bottom type, any value can be assigned to it

32) Select the snippets that are valid Haskell code:

a) len [] = 0

len (x:xs) = 1 + len xs

c) allSame :: Eq a -> a -> a -> a -> Bool

allSame a b c = (a = b) & (b==c)

33) Which of the following are examples of VALID ways to create local definitions in Haskell?

a) x + 1 where x = 2

b) let x = 2 in x + 1

34) Select all the TRUE statements about the bottom value:

b) In Haskell, undefined is the bottom value

d) It crashes the program if it's evaluated at runtime

35) Select all the FALSE statements about the bottom value:

a) In Haskell, None is the bottom value

d) In Haskell, Nothing is the bottom value

alta: If an expression has the bottom type, any value can be assigned to it

36) Select the correct functions such that the definition of m3 below multiplies 3 numbers wrapped in

Maybe

mult3 a b c = a \* b \* c

mul3 a b c = `fmap` mul3 a <\*> b <\*>

37) 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 iss:

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

R: 3

38) Select all the TRUE statements about type classes

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

39) 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 R: 2 : 4 : \_

> :sprint ham3 R: 3 : \_

40) Select the snippets that are VALID Haskell code

a) len [] = 0

len(x:xs) = 1 + len xs

c)allSame::Eq a -> a -> a -> a -> Bool

allSame a b c = (a == b) && ( b == c )

41) Select the snippets that are valid Haskell code

a) inc :: Num a = a -> a

inc a = a + 1

c) len l = case l of

[] -> 0

(\_:xs) -> 1 + len xs