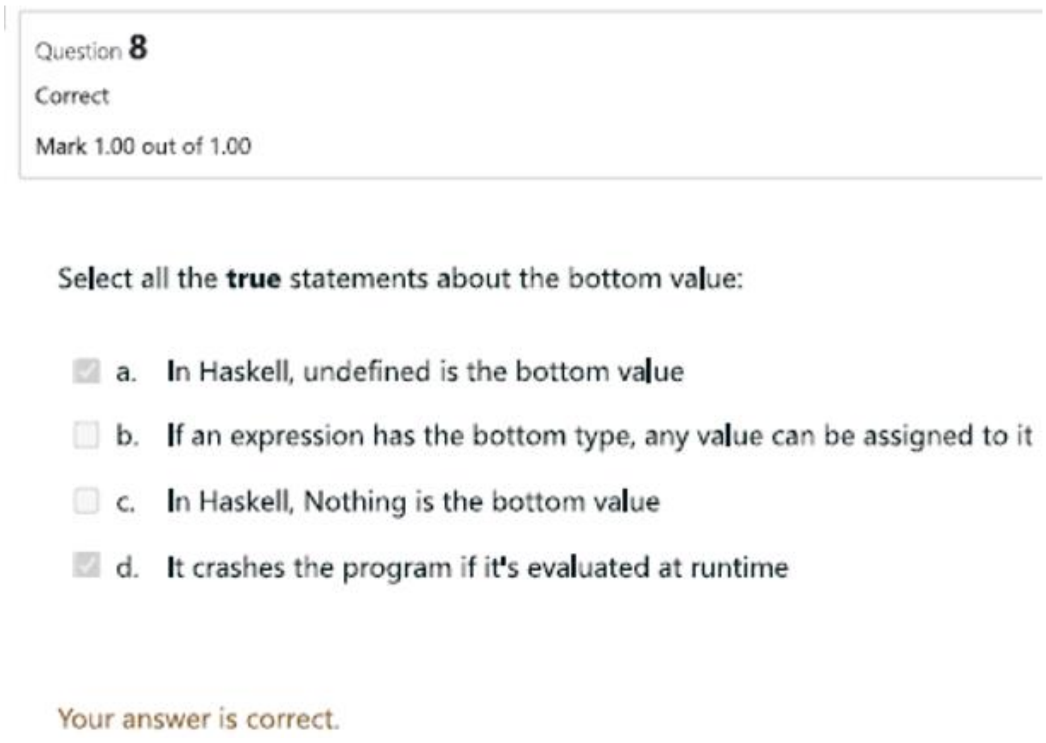
Select the **true** statements about the bottom value



Question **8**

Correct

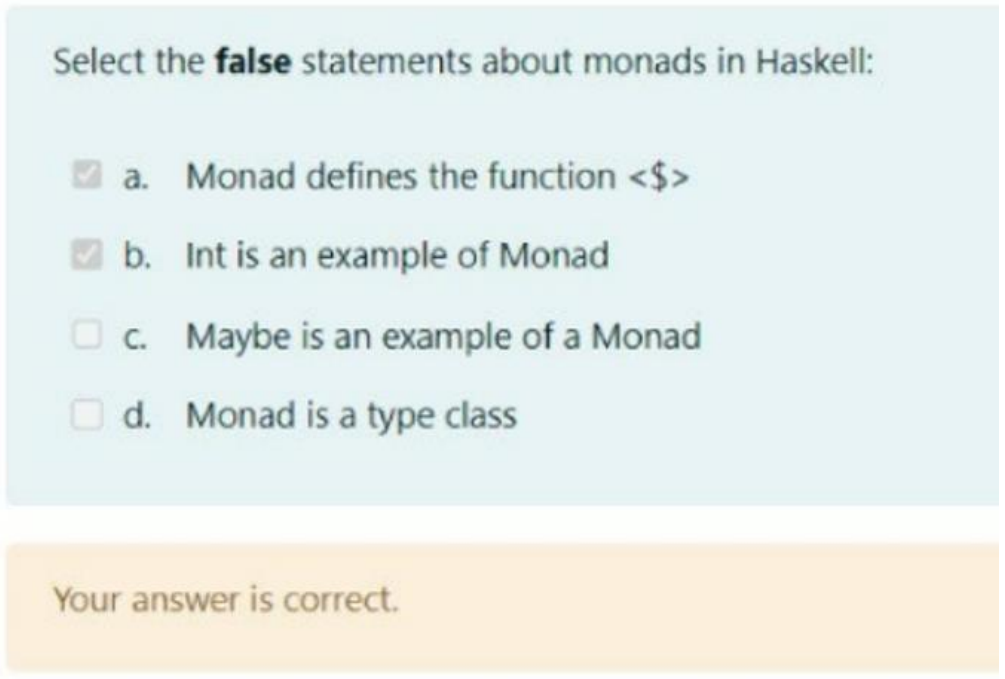
Mark 1.00 out of 1.00

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

- [✓] a. In Haskell, undefined is the bottom value
- [ ] b. If an expression has the bottom type, any value can be assigned to it
- [ ] c. In Haskell, Nothing is the bottom value
- [✓] d. It crashes the program if it's evaluated at runtime

Your answer is correct.

Select the **false** statement about monads in Haskell:

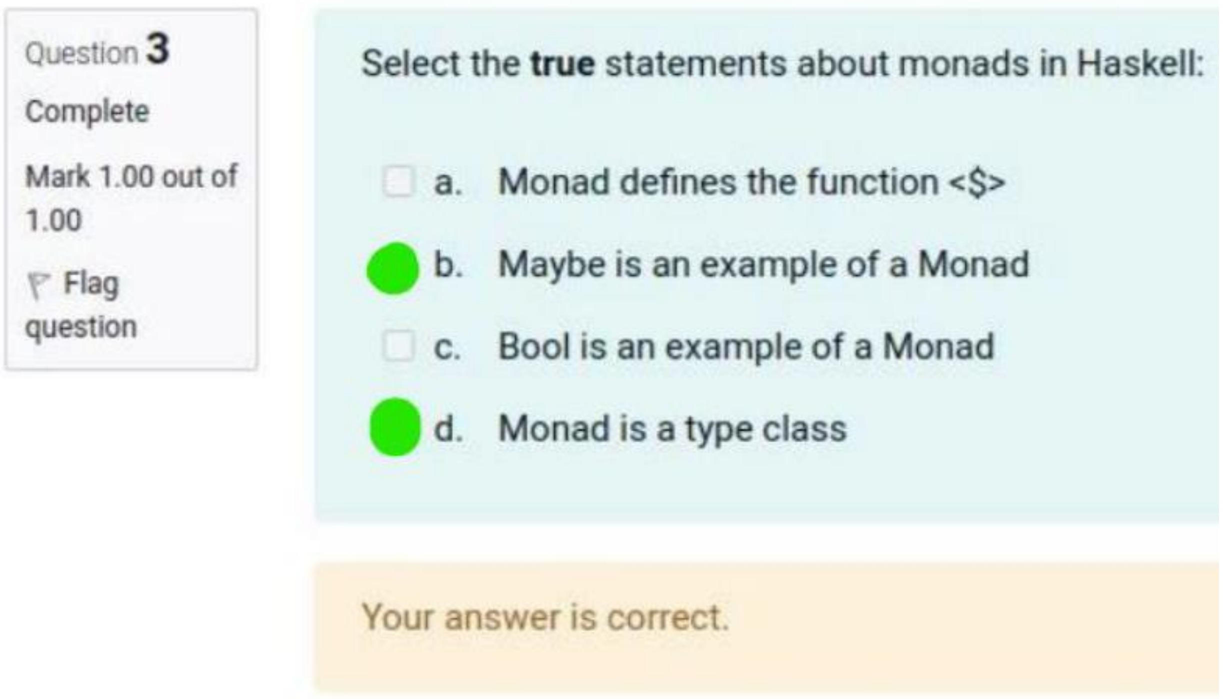


Select the **false** statements about monads in Haskell:

- [✓] a. Monad defines the function <$>
- [✓] b. Int is an example of Monad
- [ ] c. Maybe is an example of a Monad
- [ ] d. Monad is a type class

Your answer is correct.

Select the **true** statement about monads in Haskell:



Question **3**

Complete

Mark 1.00 out of 1.00

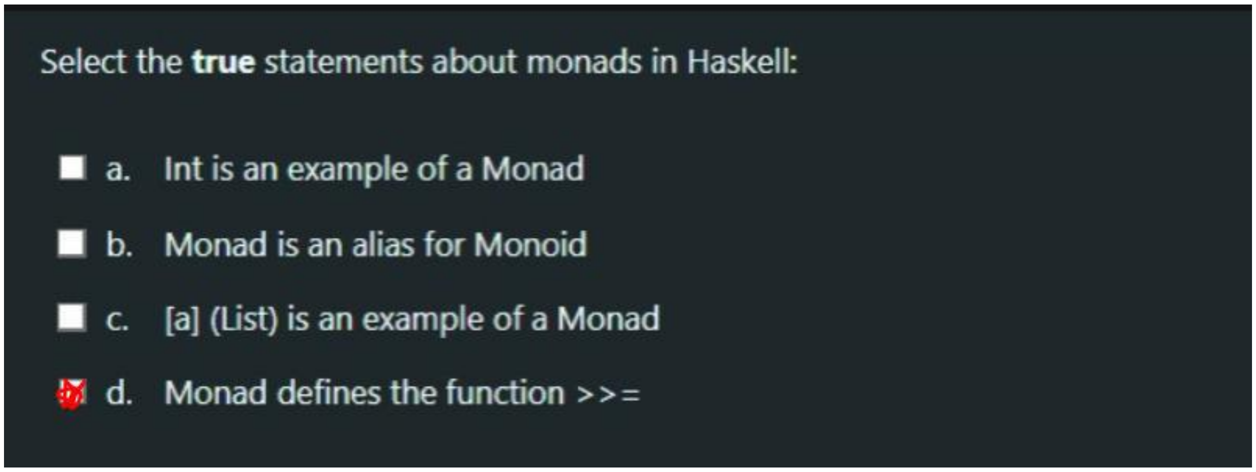
Flag question

Select the **true** statements about monads in Haskell:

- a. Monad defines the function <$>
- [✓ b. Maybe is an example of a Monad]
- c. Bool is an example of a Monad
- [✓ d. Monad is a type class]

Your answer is correct.

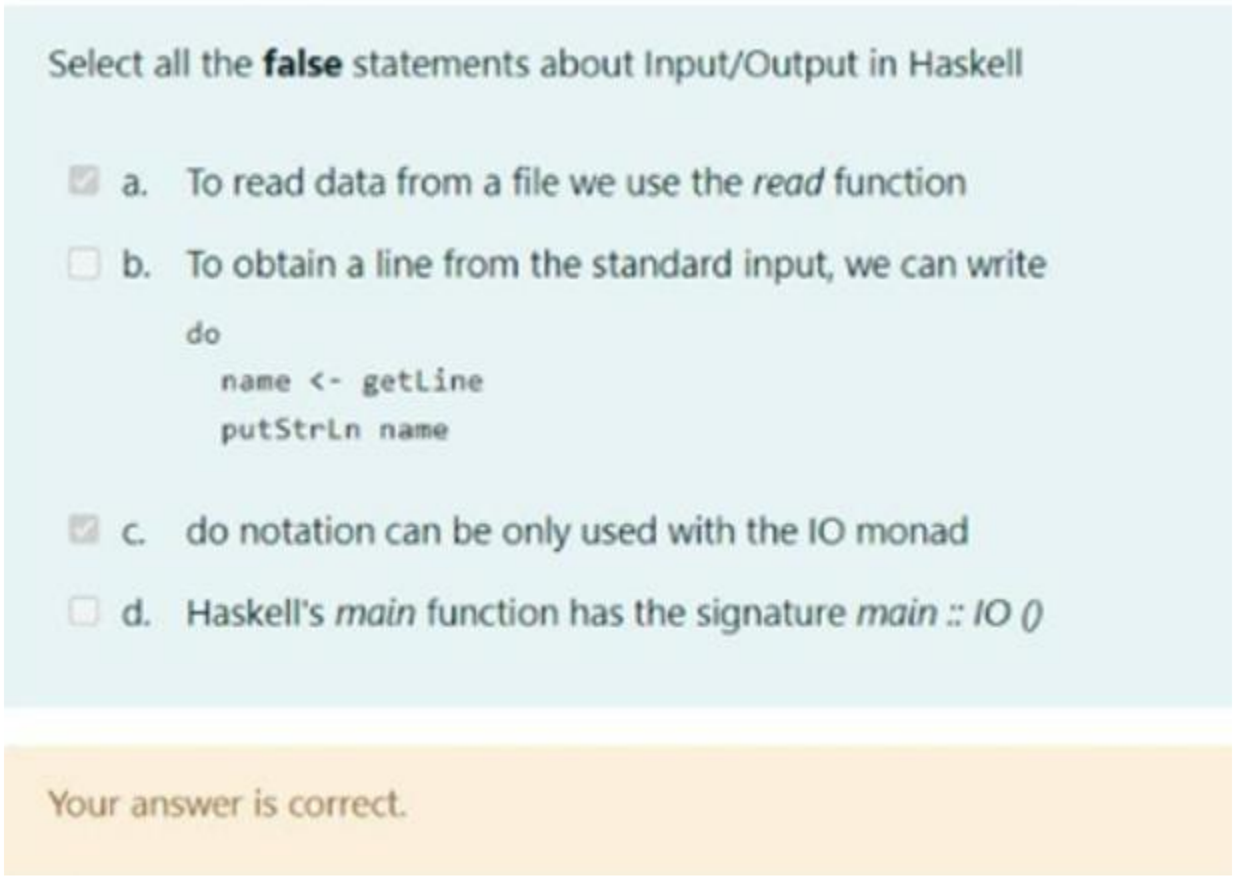
Select the **True** statements about monads in Haskell: (CHAT ZICE C SI D)



Select the **true** statements about monads in Haskell:

- a. Int is an example of a Monad
- b. Monad is an alias for Monoid
- c. [a] (List) is an example of a Monad
- [✓ d. Monad defines the function >>=]

Select all the **false** statements about Imput/Output in Haskell



Select all the **false** statements about Input/Output in Haskell

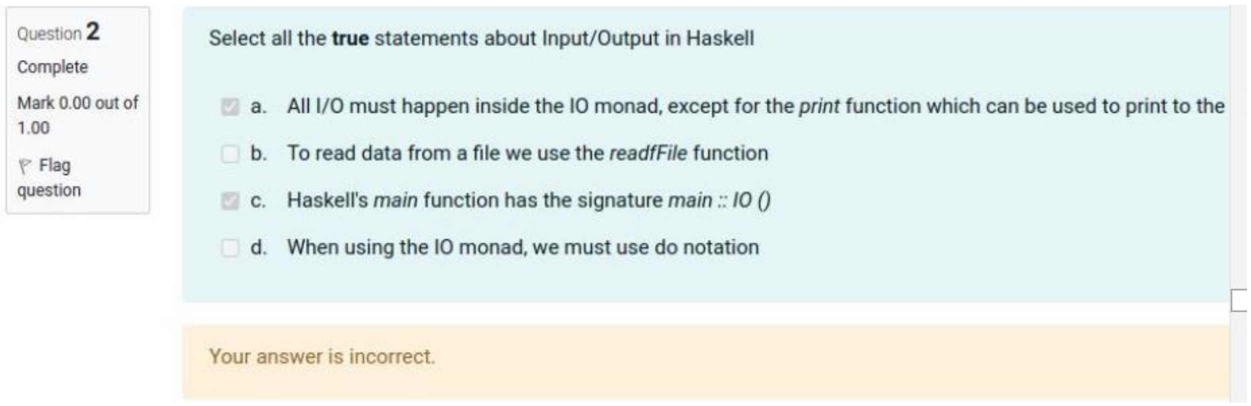
- [✓] a. To read data from a file we use the *read* function
- [ ] b. To obtain a line from the standard input, we can write

```
do
  name <- getLine
  putStrLn name
```

- [✓] c. do notation can be only used with the IO monad
- [ ] d. Haskell's *main* function has the signature *main :: IO ()*

Your answer is correct.

Select all the **true** statements about Imput/Output in Haskell (CHAT ZICE B SI C) C- ESTE SIGUR



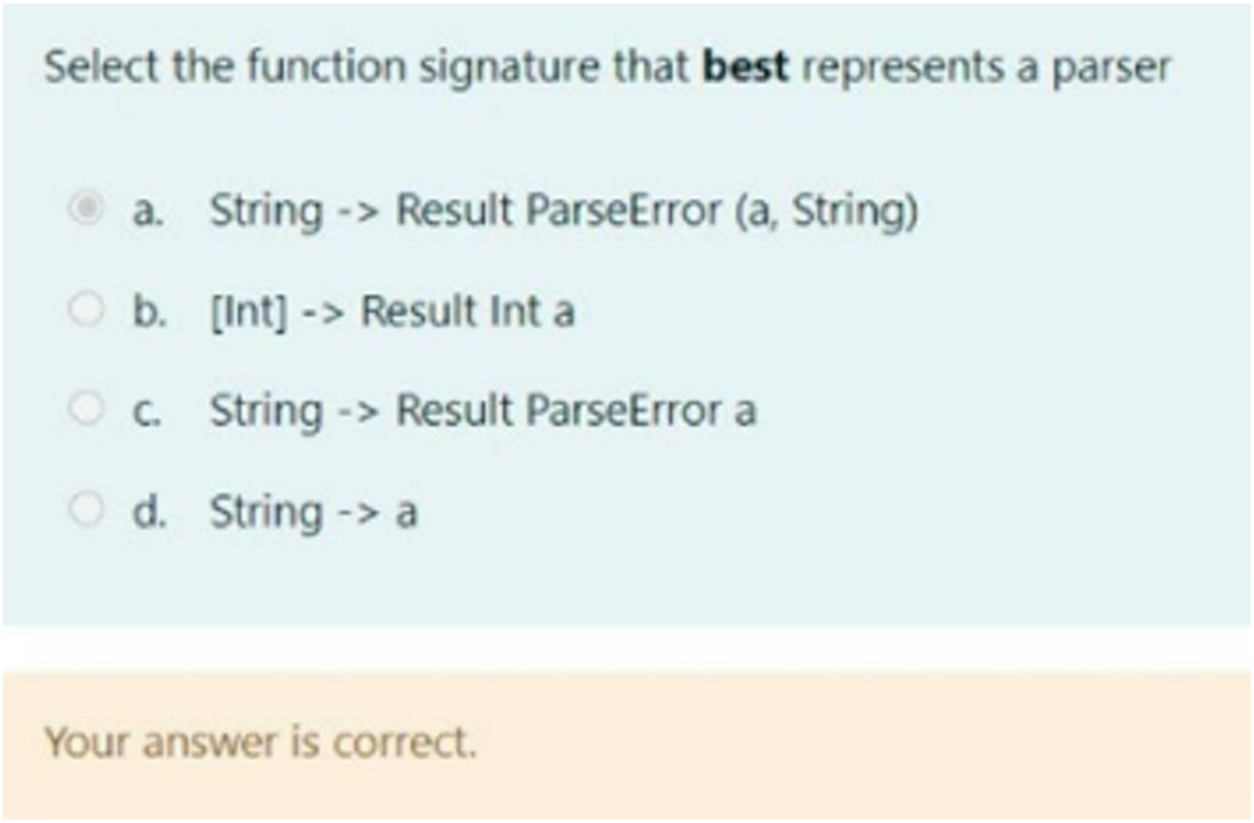
Question **2**
Complete
Mark 0.00 out of 1.00
Flag question

Select all the **true** statements about Input/Output in Haskell

- [✓] a. All I/O must happen inside the IO monad, except for the *print* function which can be used to print to the
- [ ] b. To read data from a file we use the *readfFile* function
- [✓] c. Haskell's *main* function has the signature *main :: IO ()*
- [ ] d. When using the IO monad, we must use do notation

Your answer is incorrect.

Select the function signature that **best** represents a parser:



Select the function signature that **best** represents a parser

[✓ a.] String -> Result ParseError (a, String)

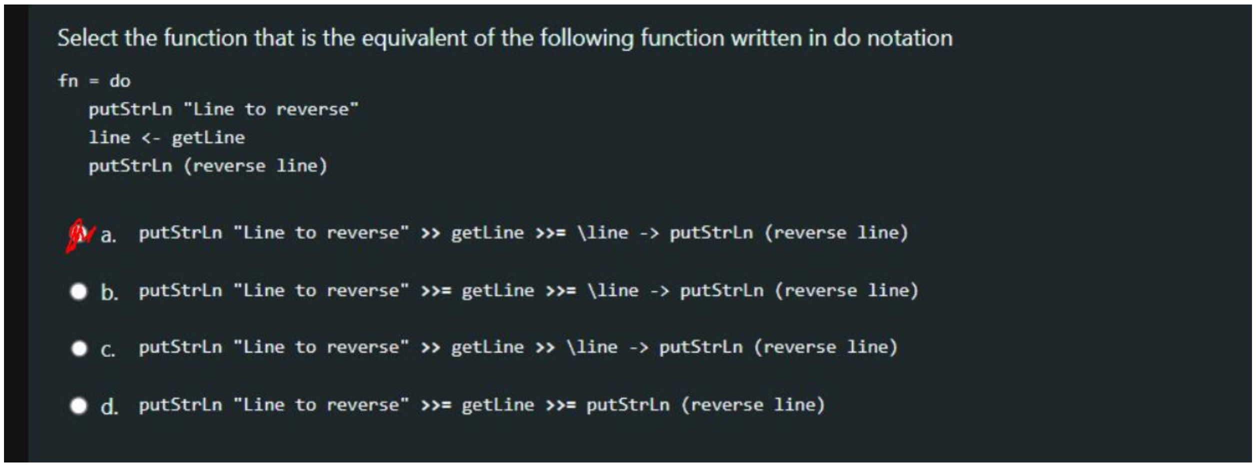
b. [Int] -> Result Int a

c. String -> Result ParseError a

d. String -> a

Your answer is correct.

Select the function that is the equivalent of the following function written in do notation (CHAT ZICE A)



Select the function that is the equivalent of the following function written in do notation

```
fn = do
   putStrLn "Line to reverse"
   line <- getLine
   putStrLn (reverse line)
```

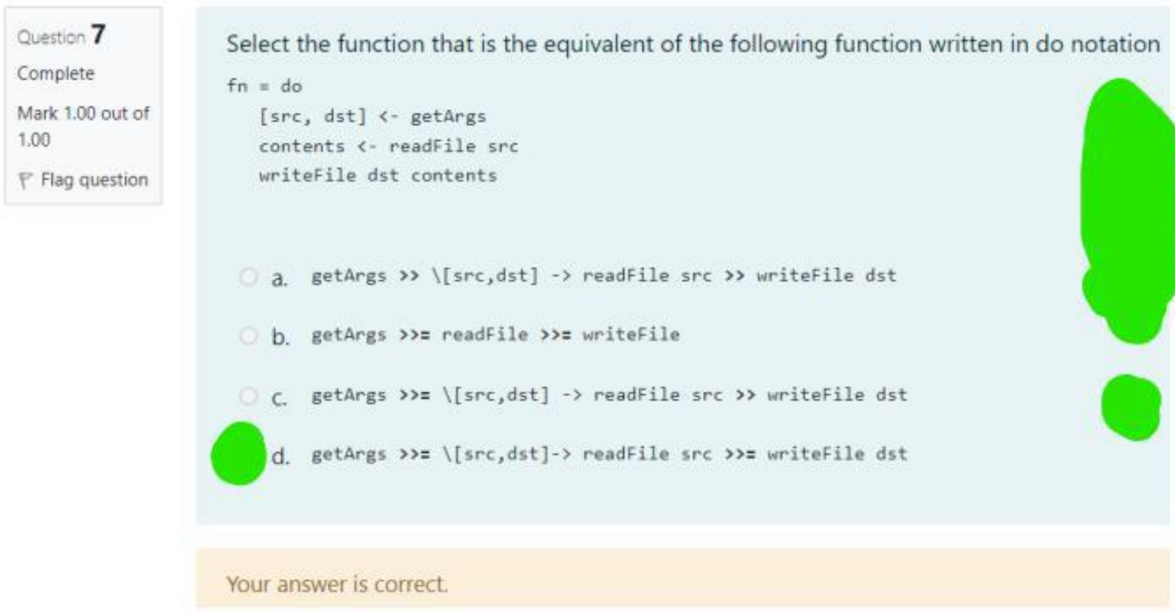
[✓ a.] `putStrLn "Line to reverse" >> getLine >>= \line -> putStrLn (reverse line)`

b. `putStrLn "Line to reverse" >>= getLine >>= \line -> putStrLn (reverse line)`

c. `putStrLn "Line to reverse" >> getLine >> \line -> putStrLn (reverse line)`

d. `putStrLn "Line to reverse" >>= getLine >>= putStrLn (reverse line)`

Select the function that is the equivalent of the following function written in do notation


Question 7
Complete
Mark 1.00 out of 1.00
Flag question

Select the function that is the equivalent of the following function written in do notation

```
fn = do
   [src, dst] <- getArgs
   contents <- readFile src
   writeFile dst contents
```

a. `getArgs >> \[src,dst] -> readFile src >> writeFile dst`

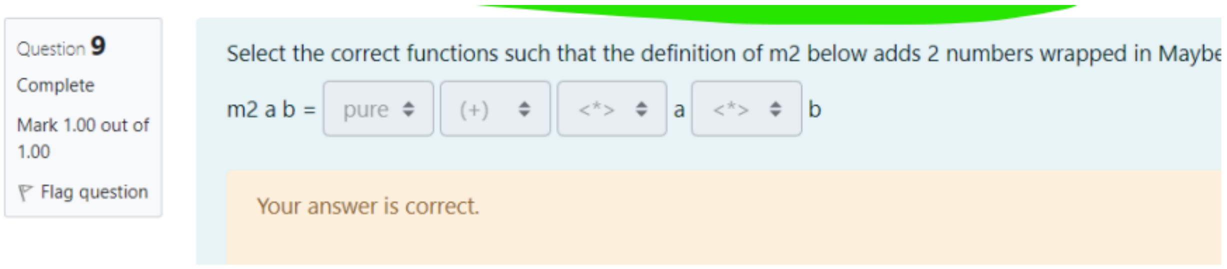
b. `getArgs >>= readFile >>= writeFile`

c. `getArgs >>= \[src,dst] -> readFile src >> writeFile dst`

[✓ d.] `getArgs >>= \[src,dst]-> readFile src >>= writeFile dst`

Your answer is correct.

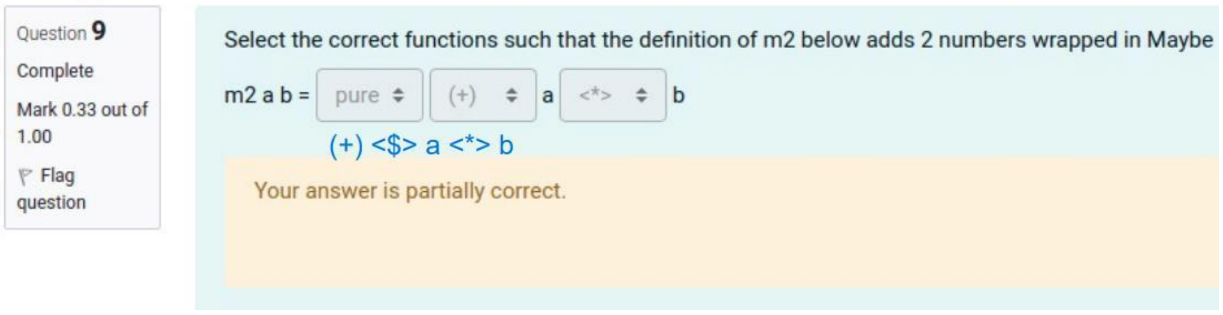
Select the correct function such that the definition of m2 below adds 2 numbers wrapped in Maybe


Question 9
Complete
Mark 1.00 out of 1.00
Flag question

Select the correct functions such that the definition of m2 below adds 2 numbers wrapped in Maybe

m2 a b = pure (+) <*> a <*> b

Your answer is correct.


Question 9
Complete
Mark 0.33 out of 1.00
Flag question

Select the correct functions such that the definition of m2 below adds 2 numbers wrapped in Maybe

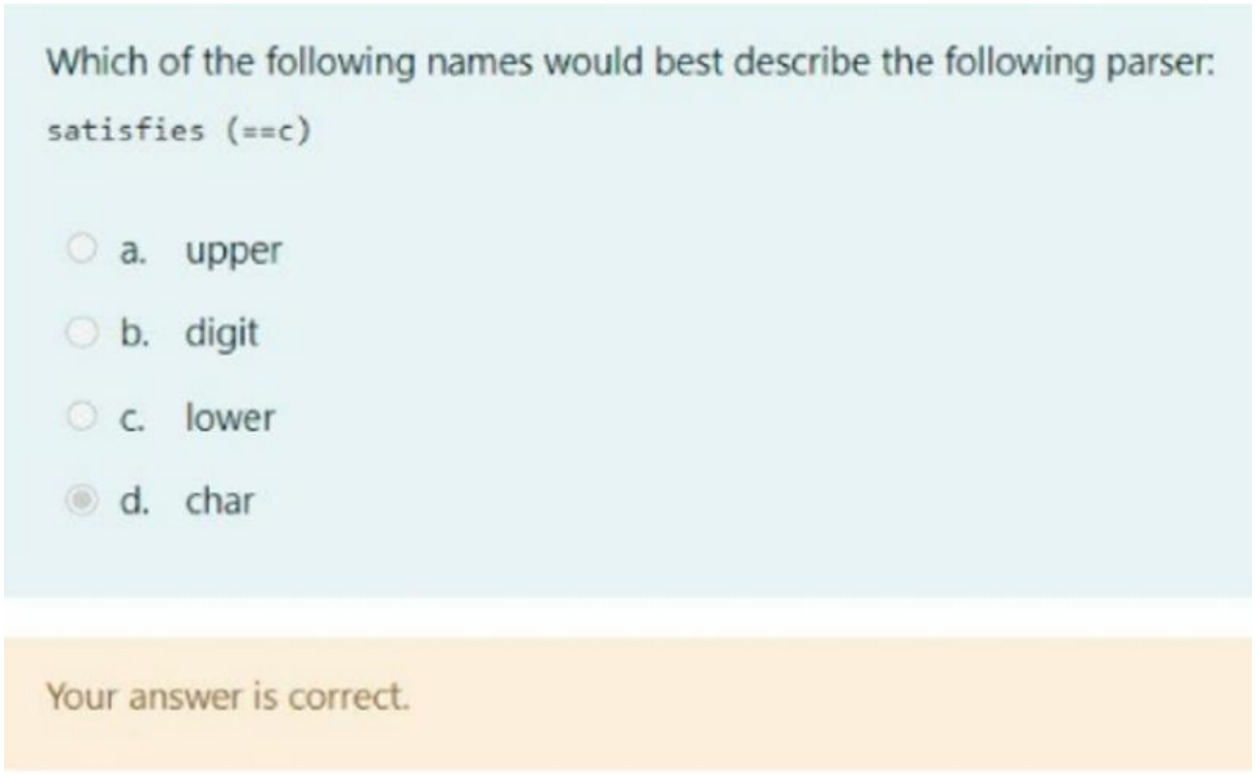
m2 a b = pure (+) a <*> b

(+) <$> a <*> b

Your answer is partially correct.

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

Satisfies (==c)



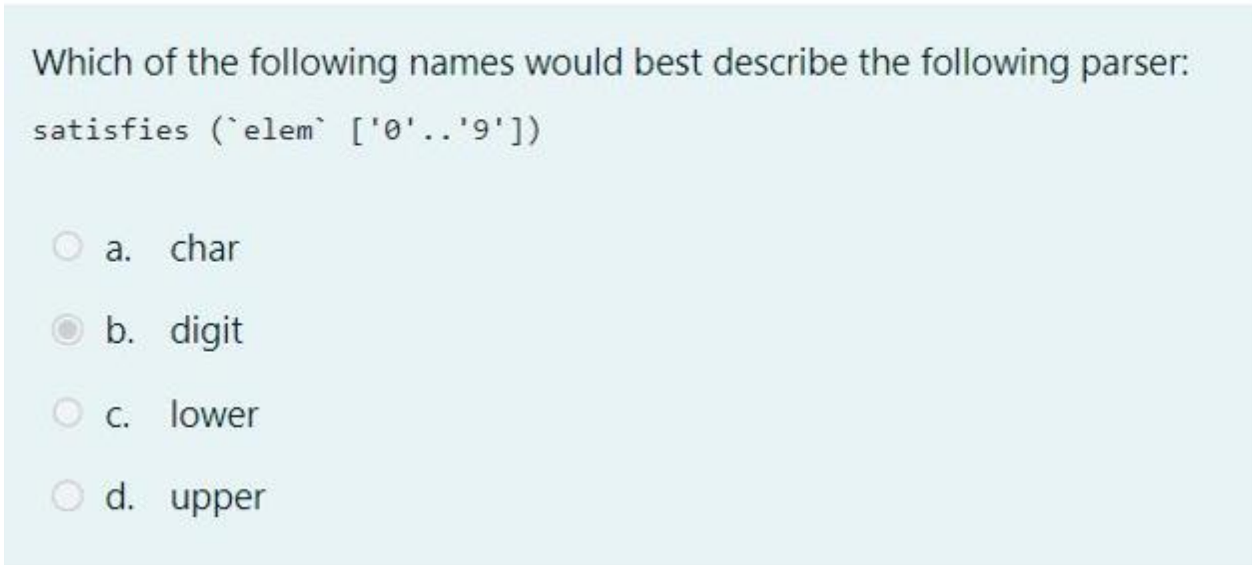
Which of the following names would best describe the following parser:

```
satisfies (==c)
```

a. upper
b. digit
c. lower
[✓ d. char]

Your answer is correct.

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

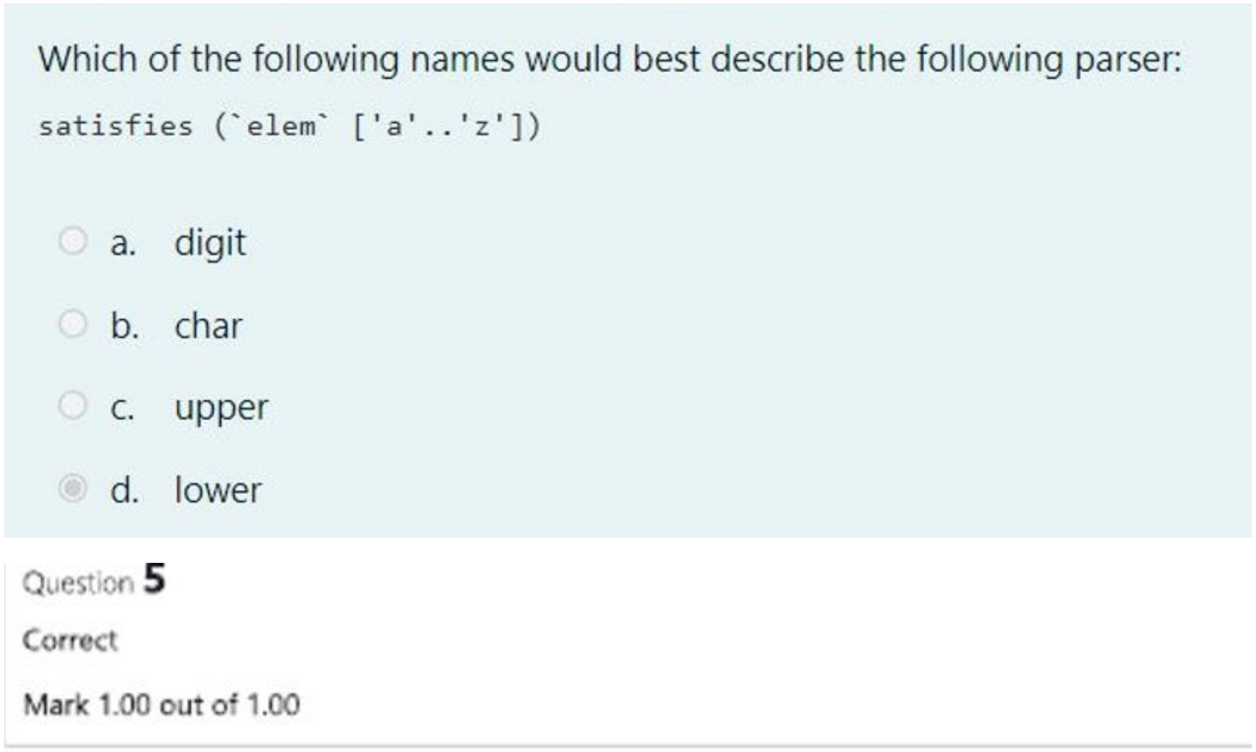


Which of the following names would best describe the following parser:

```
satisfies (`elem` ['0'..'9'])
```

a. char
[✓ b. digit]
c. lower
d. upper

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



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

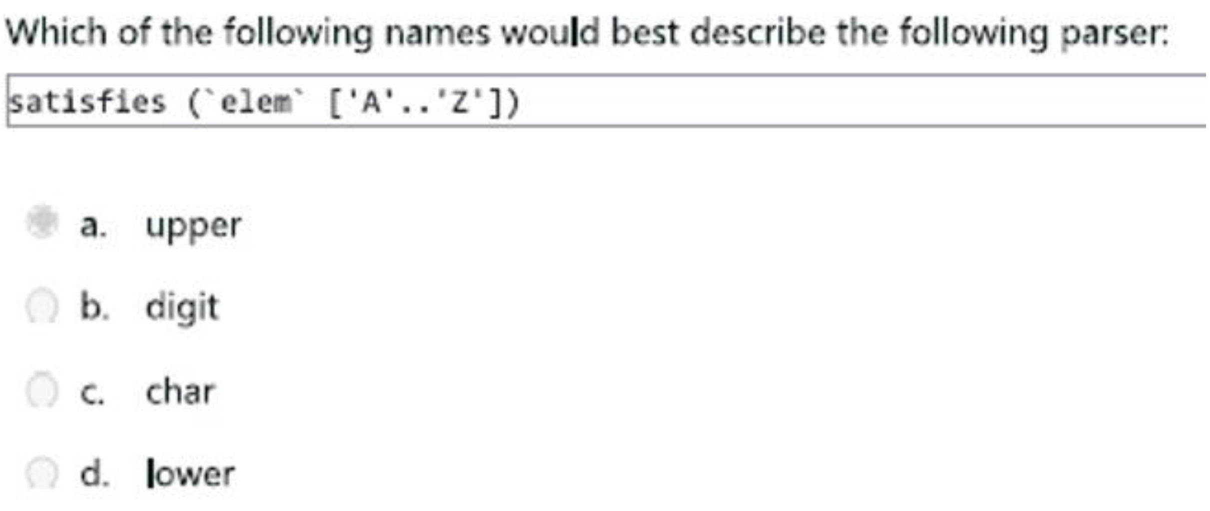
```
satisfies (`elem` ['a'..'z'])
```

a. digit
b. char
c. upper
[✓ d. lower]

Question **5**

Correct

Mark 1.00 out of 1.00

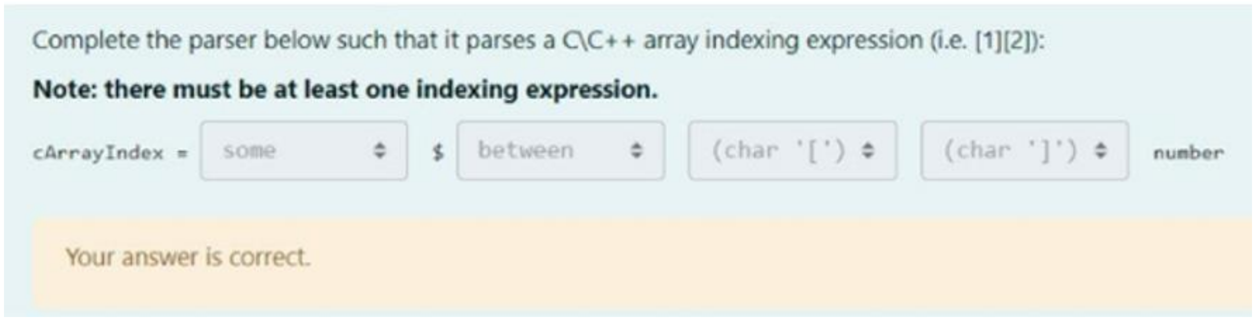


Which of the following names would best describe the following parser:

```
satisfies (`elem` ['A'..'Z'])
```

[✓ a. upper]
b. digit
c. char
d. lower

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



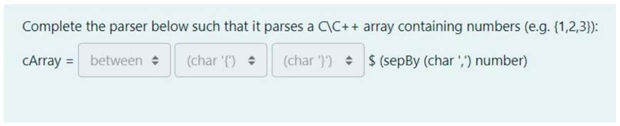
Complete the parser below such that it parses a C\C++ array indexing expression (i.e. [1][2]):

Note: there must be at least one indexing expression.

```
cArrayIndex = some $ between (char '[') (char ']') number
```

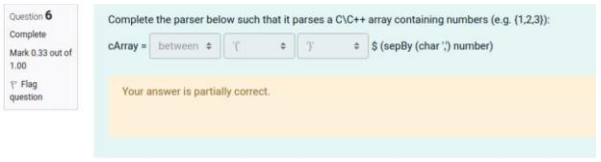
Your answer is correct.

cArray = between (char '[') (char ']') (sepBy number (char ','))



Complete the parser below such that it parses a C\C++ array containing numbers (e.g. {1,2,3}):

cArray = between (char '{') (char '}') $ (sepBy (char ',') number)



Question 6

Complete

Mark 0.33 out of 1.00

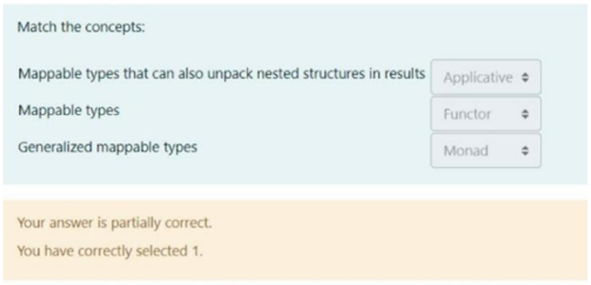
Flag question

Complete the parser below such that it parses a C\C++ array containing numbers (e.g. {1,2,3}):

cArray = between '(' ')' $ (sepBy (char ',') number)

Your answer is partially correct.

Match the concepts: -orice e posibil

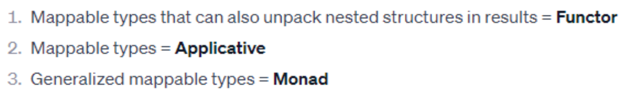


Match the concepts:

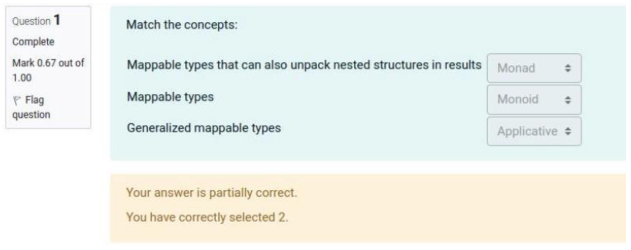
Mappable types that can also unpack nested structures in results	Applicative
Mappable types	Functor
Generalized mappable types	Monad

Your answer is partially correct.

You have correctly selected 1.



1. Mappable types that can also unpack nested structures in results = **Functor**
2. Mappable types = **Applicative**
3. Generalized mappable types = **Monad**



Question **1**

Complete

Mark 0.67 out of 1.00

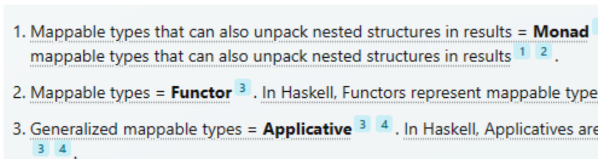
Flag question

Match the concepts:

Mappable types that can also unpack nested structures in results	Monad
Mappable types	Monoid
Generalized mappable types	Applicative

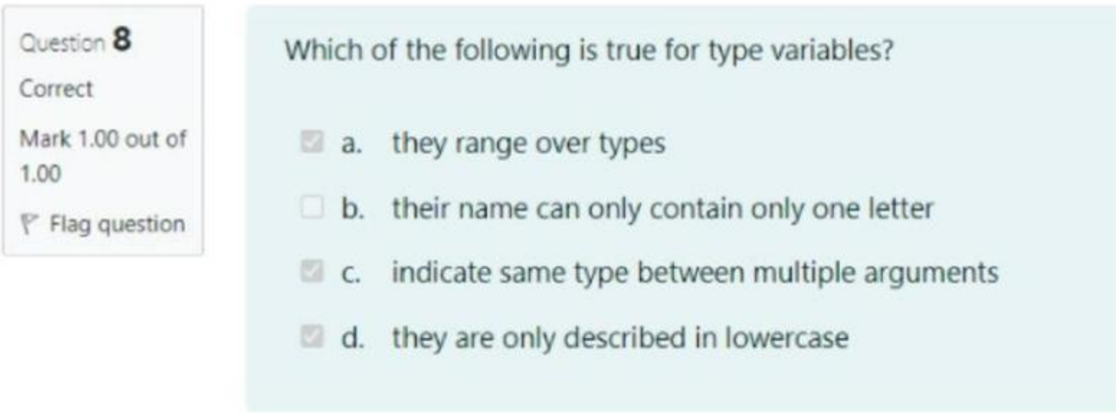
Your answer is partially correct.

You have correctly selected 2.



1. Mappable types that can also unpack nested structures in results = **Monad** mappable types that can also unpack nested structures in results [1] [2] .
2. Mappable types = **Functor** [3] . In Haskell, Functors represent mappable type
3. Generalized mappable types = **Applicative** [3] [4] . In Haskell, Applicatives are [3] [4] .

Given of the following is **true** for type variables



Question **8**

Correct

Mark 1.00 out of 1.00

Flag question

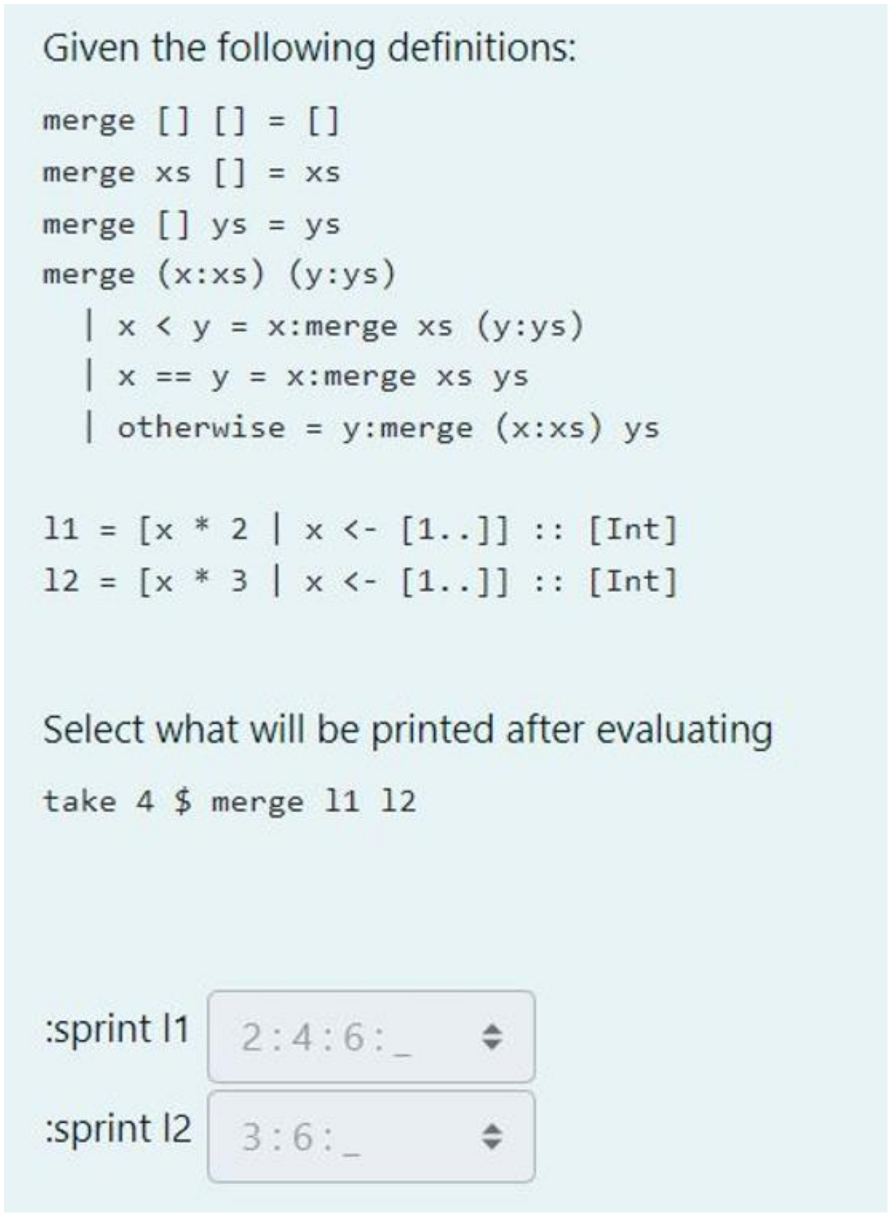
Which of the following is true for type variables?

- [✓] a. they range over types
- [ ] b. their name can only contain only one letter
- [✓] c. indicate same type between multiple arguments
- [✓] d. they are only described in lowercase

Given the following function:

l1 = [2, 4, 6, 8, ...]

l2 = [3, 6, 9, 12, ...]



Given the following definitions:

```
merge [] [] = []
merge xs [] = xs
merge [] ys = ys
merge (x:xs) (y:ys)
  | x < y = x:merge xs (y:ys)
  | x == y = x:merge xs ys
  | otherwise = y:merge (x:xs) ys

l1 = [x * 2 | x <- [1..]] :: [Int]
l2 = [x * 3 | x <- [1..]] :: [Int]
```

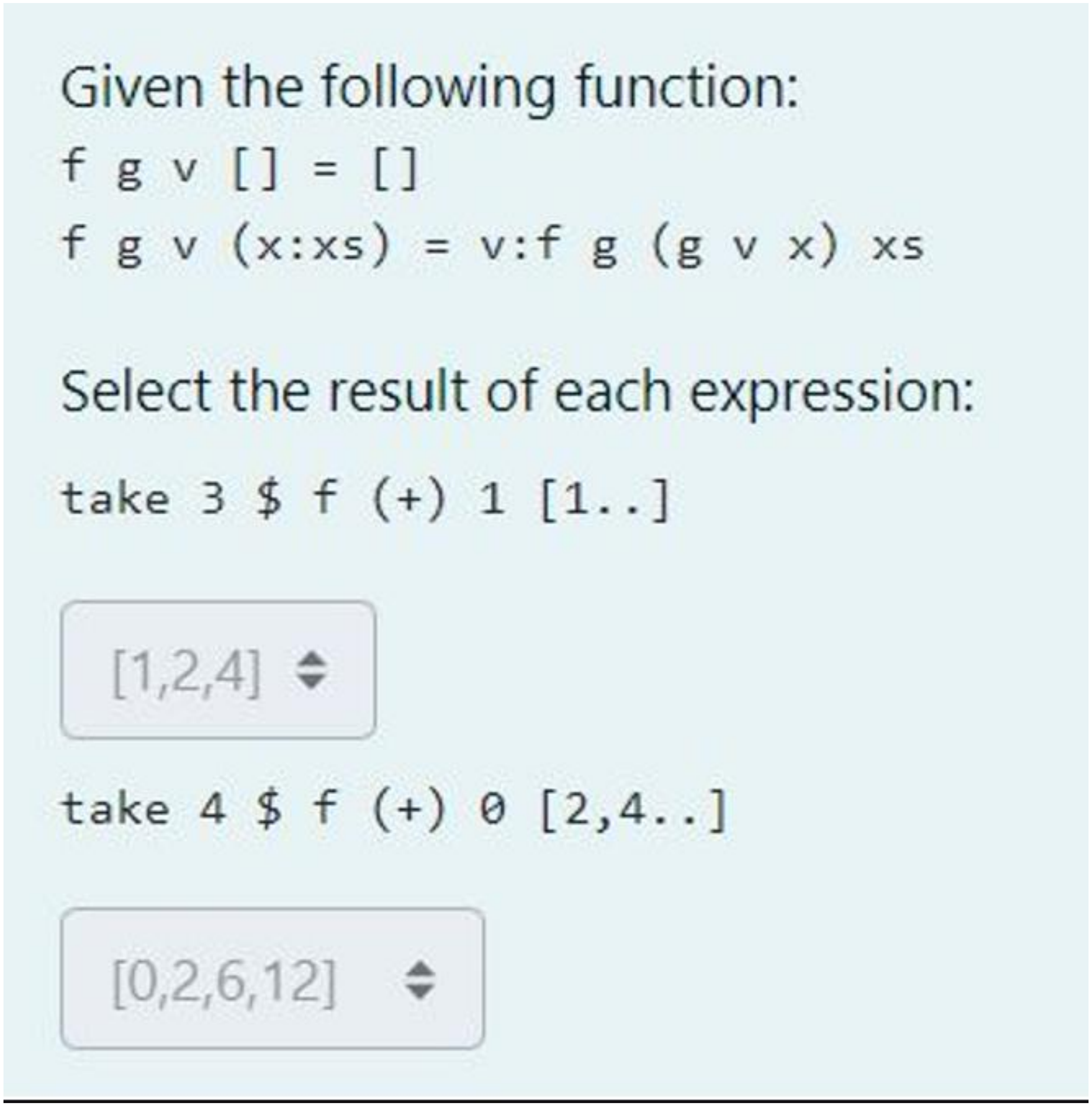
Select what will be printed after evaluating

```
take 4 $ merge l1 l2
```

:sprint l1 2:4:6:_

:sprint l2 3:6:_

Given the following function: E OK ASA



Given the following function:

```
f g v [] = []
f g v (x:xs) = v:f g (g v x) xs
```

Select the result of each expression:

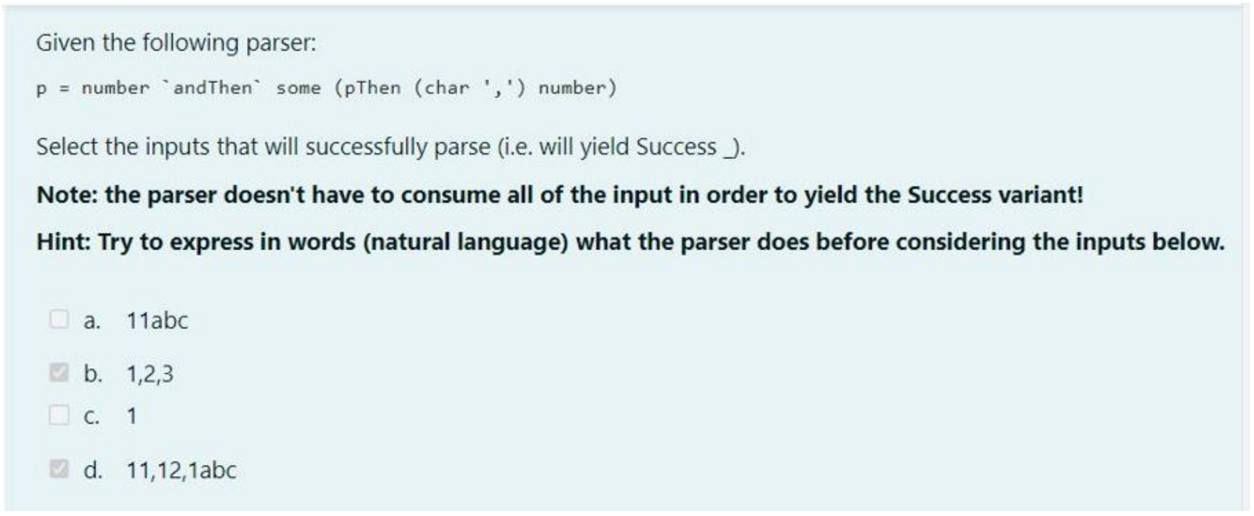
```
take 3 $ f (+) 1 [1..]
```

[1,2,4]

```
take 4 $ f (+) 0 [2,4..]
```

[0,2,6,12]

Given the following parser ? D SIGUR



Given the following parser:

```
p = number `andThen` some (pThen (char ',') number)
```

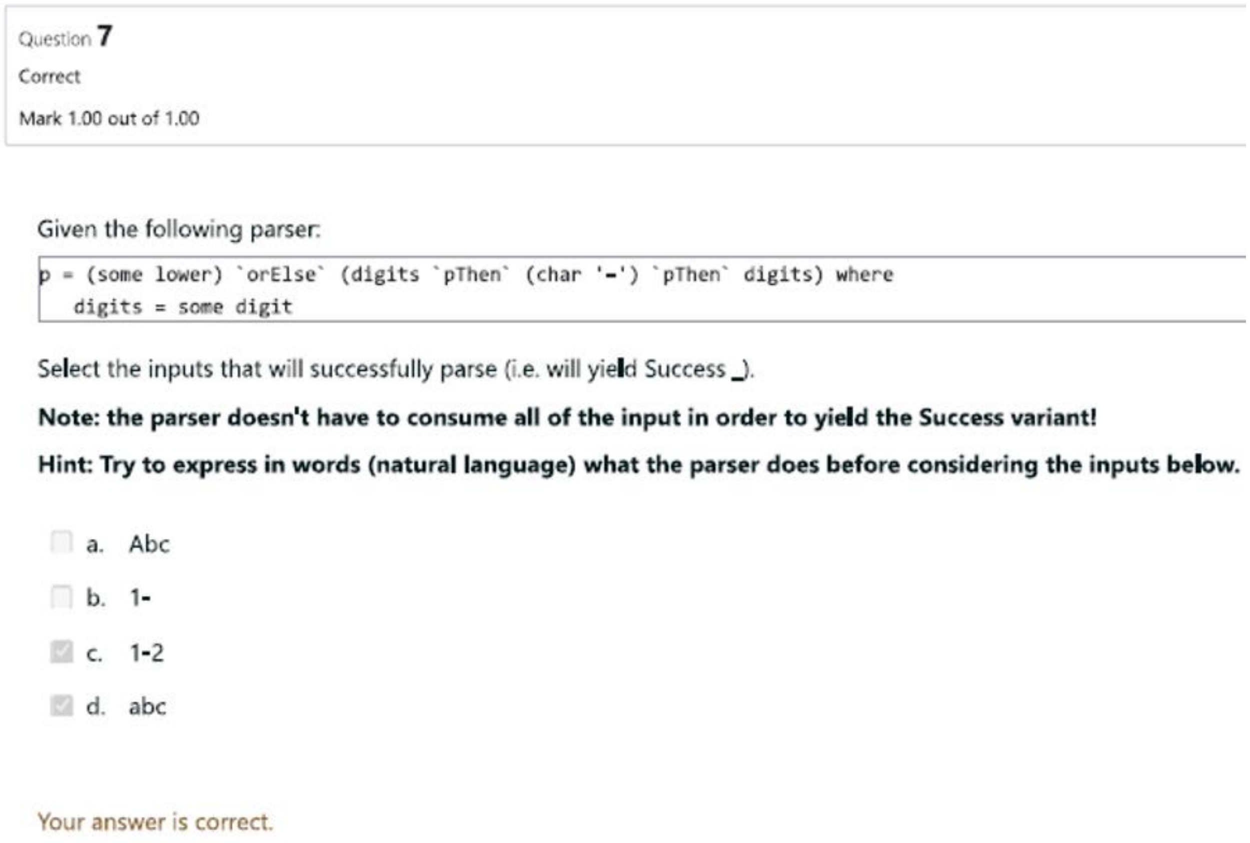
Select the inputs that will successfully parse (i.e. will yield Success _).

Note: the parser doesn't have to consume all of the input in order to yield the Success variant!

Hint: Try to express in words (natural language) what the parser does before considering the inputs below.

- ☐ a. 11abc
- ✓ b. 1,2,3
- ☐ c. 1
- ✓ d. 11,12,1abc

Given the following parser



Question **7**

Correct

Mark 1.00 out of 1.00

Given the following parser:

```
p = (some lower) `orElse` (digits `pThen` (char '-') `pThen` digits) where
   digits = some digit
```

Select the inputs that will successfully parse (i.e. will yield Success _).

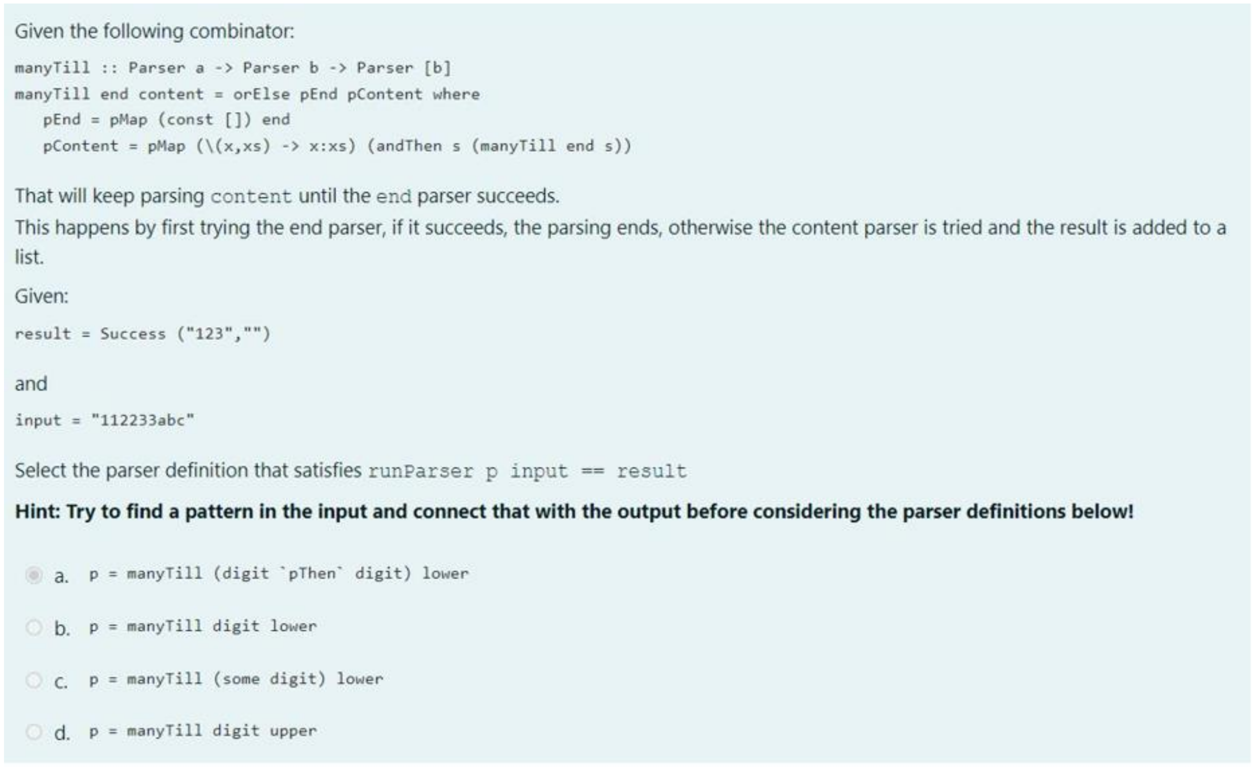
Note: the parser doesn't have to consume all of the input in order to yield the Success variant!

Hint: Try to express in words (natural language) what the parser does before considering the inputs below.

- ☐ a. Abc
- ☐ b. 1-
- ✓ c. 1-2
- ✓ d. abc

Your answer is correct.

Given the following combinator(CHAT ZICE C)



Given the following combinator:

```
manyTill :: Parser a -> Parser b -> Parser [b]
manyTill end content = orElse pEnd pContent where
   pEnd = pMap (const []) end
   pContent = pMap (\(x,xs) -> x:xs) (andThen s (manyTill end s))
```

That will keep parsing `content` until the `end` parser succeeds.
This happens by first trying the end parser, if it succeeds, the parsing ends, otherwise the content parser is tried and the result is added to a list.

Given:

```
result = Success ("123","")
```

and

```
input = "112233abc"
```

Select the parser definition that satisfies `runParser p input == result`

Hint: Try to find a pattern in the input and connect that with the output before considering the parser definitions below!

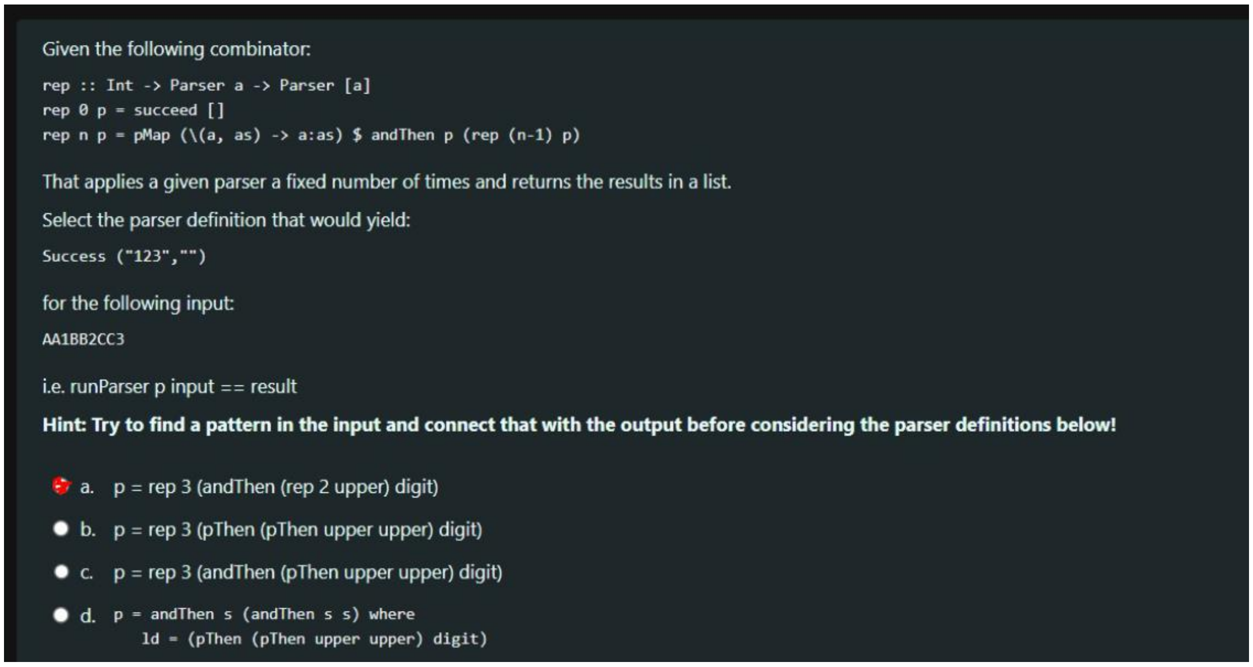
[✓ a.] `p = manyTill (digit `pThen` digit) lower`

b. `p = manyTill digit lower`

c. `p = manyTill (some digit) lower`

d. `p = manyTill digit upper`

Given the following combinator ? (CHAT ZICE A)



Given the following combinator:

```
rep :: Int -> Parser a -> Parser [a]
rep 0 p = succeed []
rep n p = pMap (\(a, as) -> a:as) $ andThen p (rep (n-1) p)
```

That applies a given parser a fixed number of times and returns the results in a list.

Select the parser definition that would yield:

```
Success ("123","")
```

for the following input:

```
AA1BB2CC3
```

i.e. runParser p input == result

Hint: Try to find a pattern in the input and connect that with the output before considering the parser definitions below!

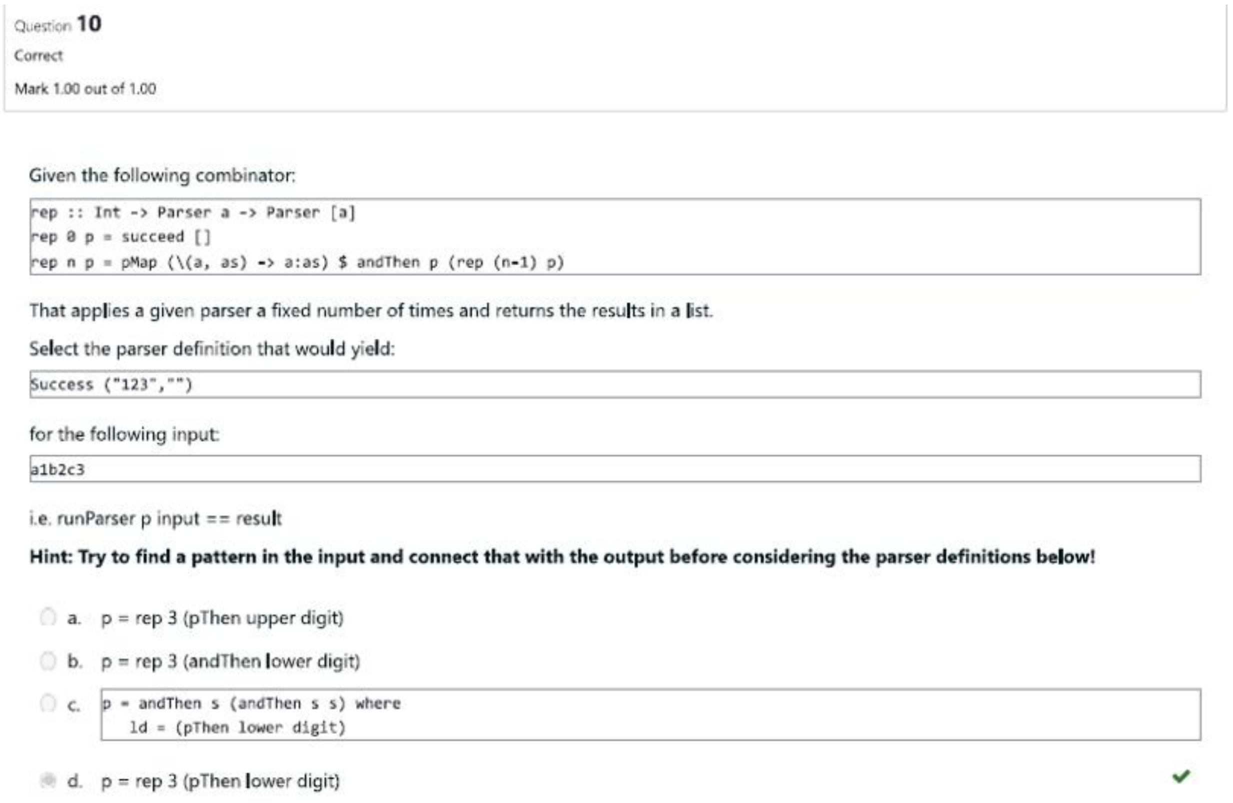
[✓ a.] p = rep 3 (andThen (rep 2 upper) digit)

b. p = rep 3 (pThen (pThen upper upper) digit)

c. p = rep 3 (andThen (pThen upper upper) digit)

d.
```
p = andThen s (andThen s s) where
    ld = (pThen (pThen upper upper) digit)
```

Given the following combinator



Question **10**

Correct

Mark 1.00 out of 1.00

Given the following combinator:

```
rep :: Int -> Parser a -> Parser [a]
rep 0 p = succeed []
rep n p = pMap (\(a, as) -> a:as) $ andThen p (rep (n-1) p)
```

That applies a given parser a fixed number of times and returns the results in a list.

Select the parser definition that would yield:

```
Success ("123","")
```

for the following input:

```
a1b2c3
```

i.e. runParser p input == result

Hint: Try to find a pattern in the input and connect that with the output before considering the parser definitions below!

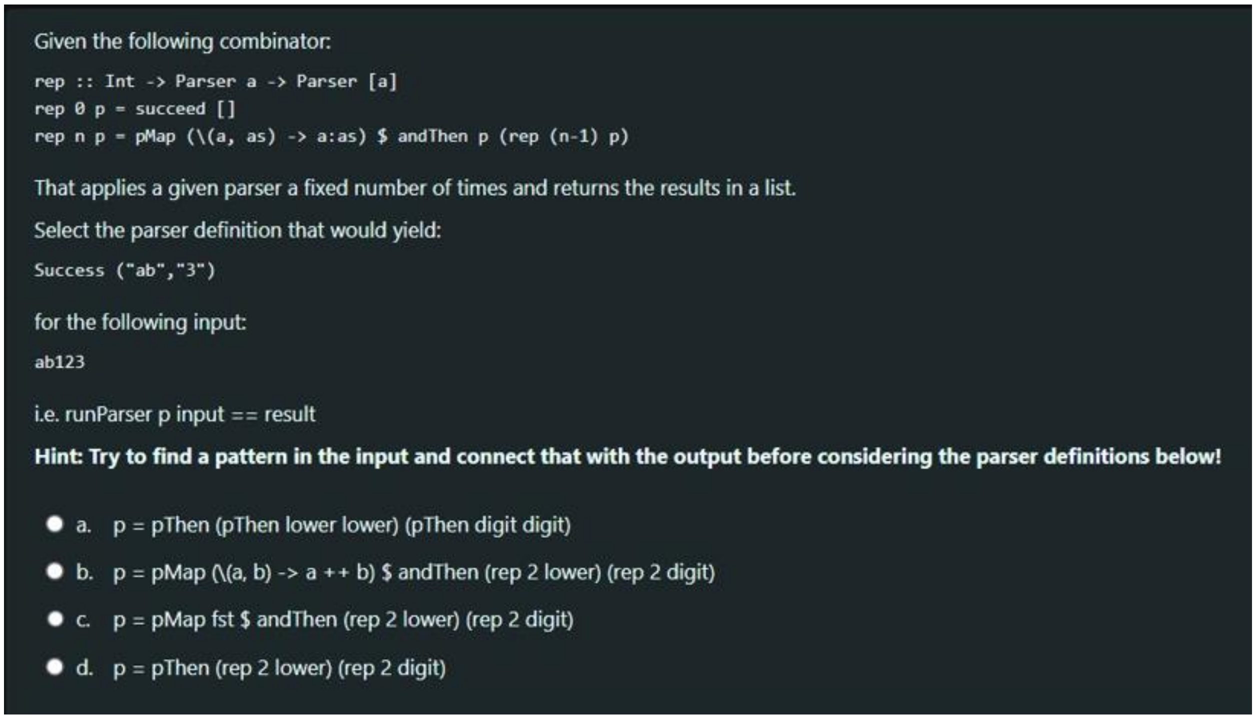
a. p = rep 3 (pThen upper digit)

b. p = rep 3 (andThen lower digit)

c.
```
p = andThen s (andThen s s) where
  ld = (pThen lower digit)
```

[✓ d. p = rep 3 (pThen lower digit)] ✓

Given the following combinator ?(CHAT ZICE A)



Given the following combinator:

```
rep :: Int -> Parser a -> Parser [a]
rep 0 p = succeed []
rep n p = pMap (\(a, as) -> a:as) $ andThen p (rep (n-1) p)
```

That applies a given parser a fixed number of times and returns the results in a list.

Select the parser definition that would yield:

```
Success ("ab","3")
```

for the following input:

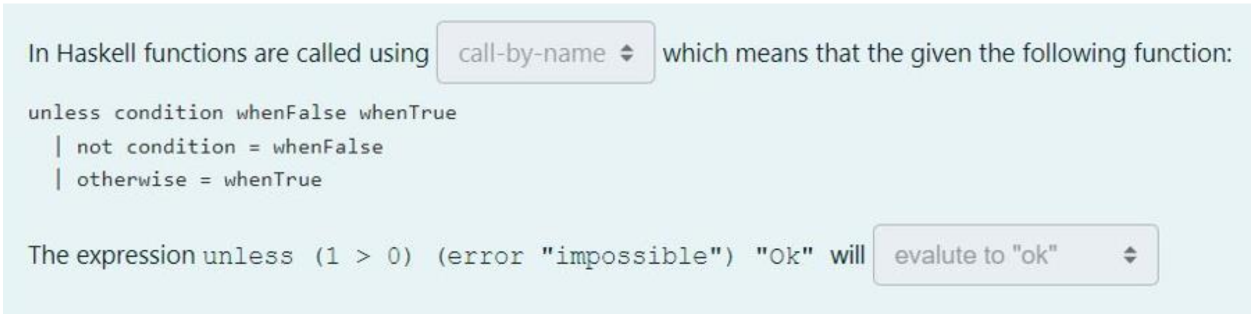
```
ab123
```

i.e. runParser p input == result

Hint: Try to find a pattern in the input and connect that with the output before considering the parser definitions below!

- a. p = pThen (pThen lower lower) (pThen digit digit)
- b. p = pMap (\(a, b) -> a ++ b) $ andThen (rep 2 lower) (rep 2 digit)
- c. p = pMap fst $ andThen (rep 2 lower) (rep 2 digit)
- d. p = pThen (rep 2 lower) (rep 2 digit)

In Haskell function are called using **call-by-name** which means that the given the following function

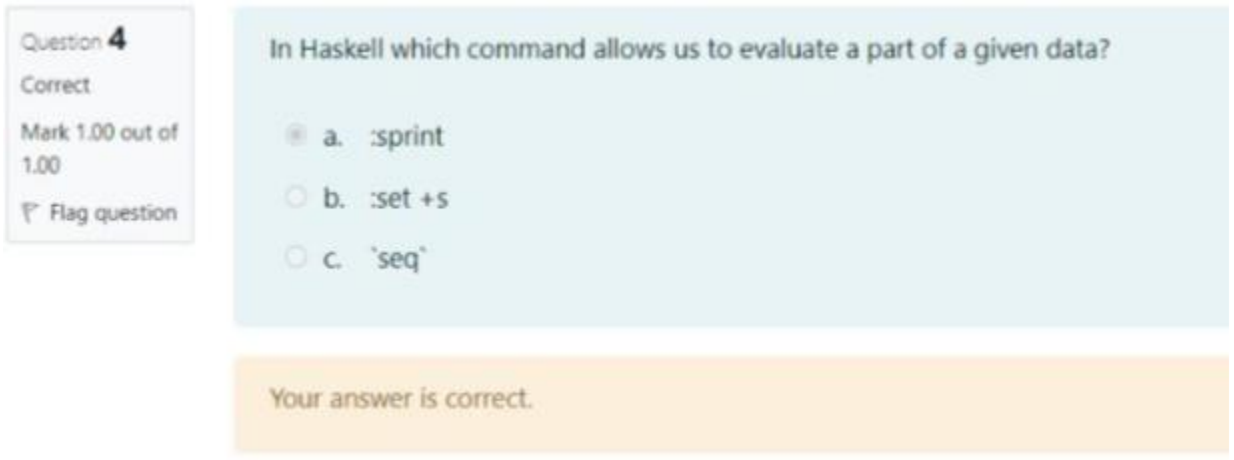


In Haskell functions are called using [call-by-name] which means that the given the following function:

```
unless condition whenFalse whenTrue
  | not condition = whenFalse
  | otherwise = whenTrue
```

The expression `unless (1 > 0) (error "impossible") "Ok"` will [evalute to "ok"]

In Haskell which command allows us to evaluate a part of a given data?



Question **4**
Correct
Mark 1.00 out of 1.00
Flag question

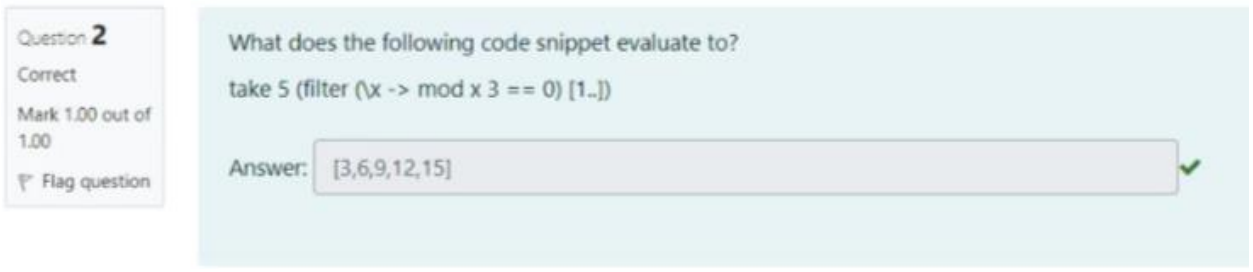
In Haskell which command allows us to evaluate a part of a given data?

[✓ a. :sprint]
b. :set +s
c. `seq`

Your answer is correct.

s

What does the following code snippet evaluate to



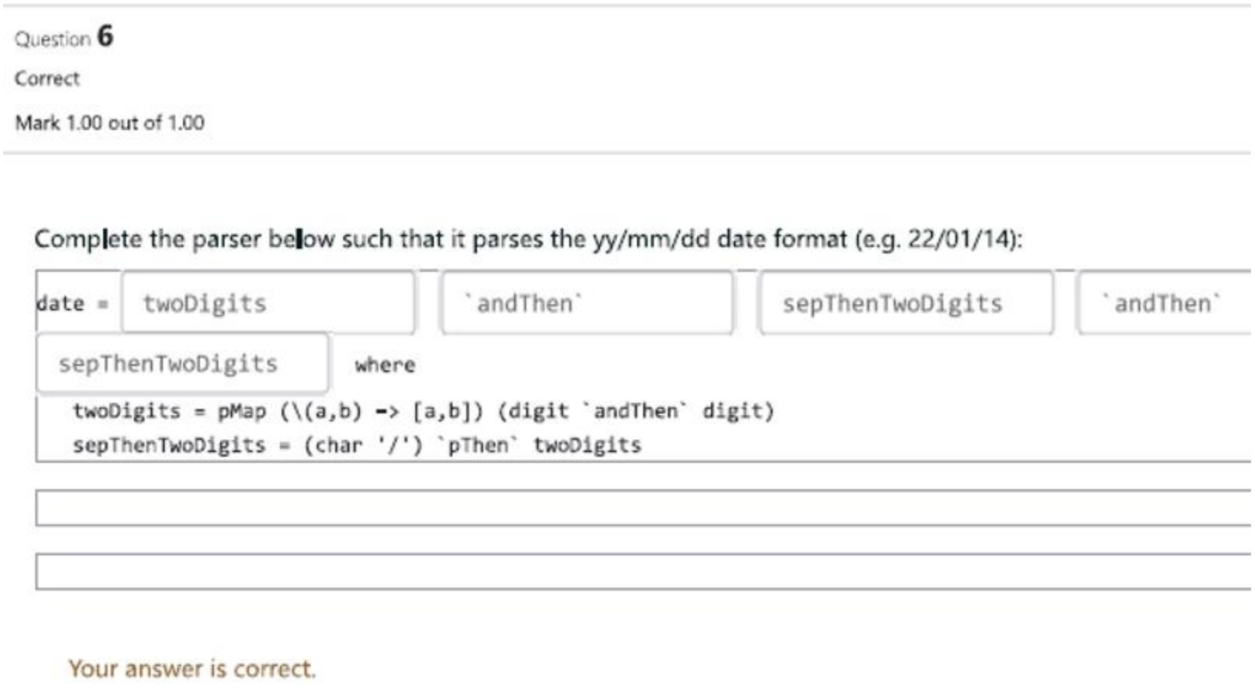
Question **2**
Correct
Mark 1.00 out of 1.00
Flag question

What does the following code snippet evaluate to?

take 5 (filter (\x -> mod x 3 == 0) [1..])

Answer: [3,6,9,12,15] ✓

Complete the parser below such that it parses the yy/mm/dd date format



Question **6**

Correct

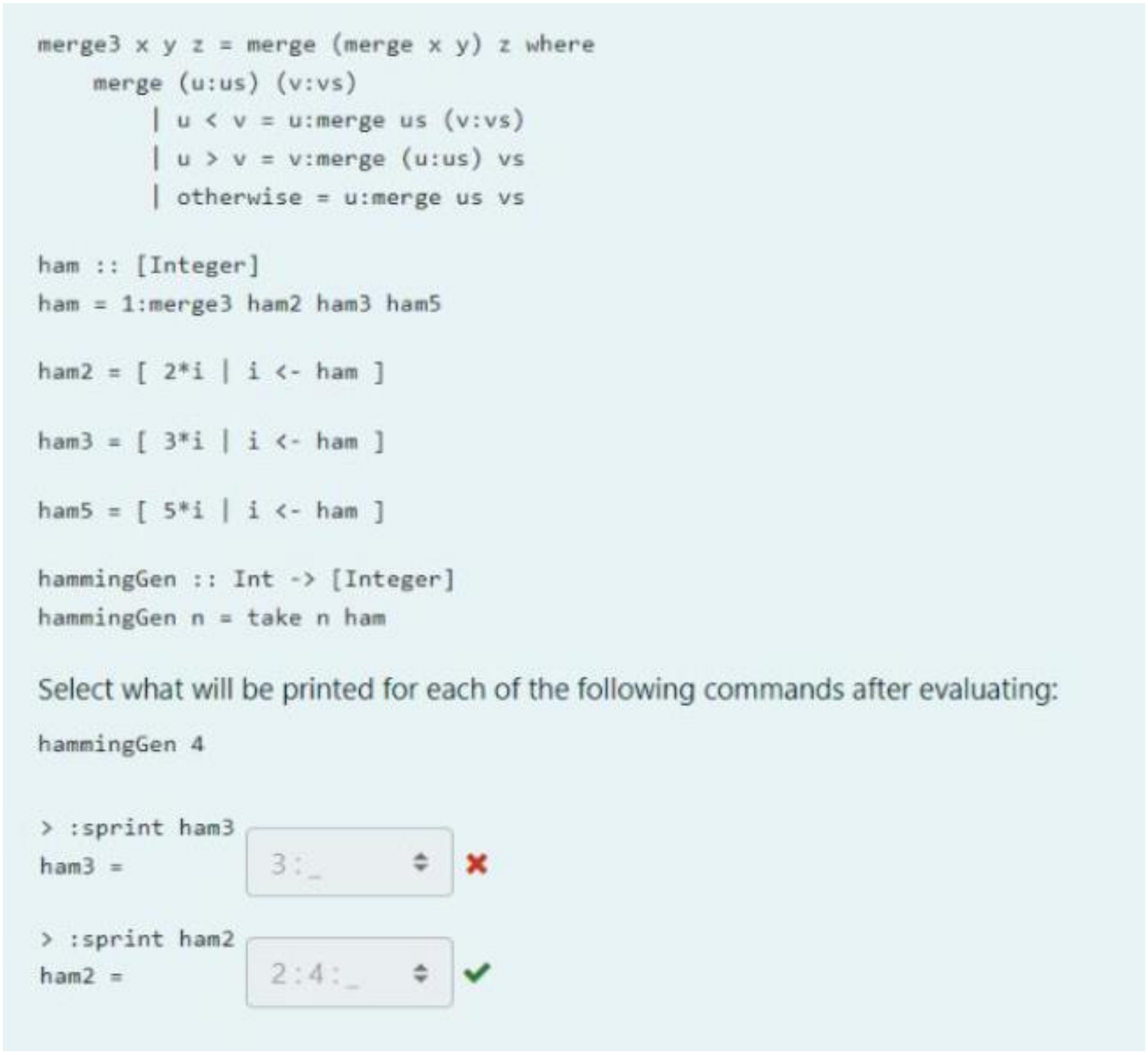
Mark 1.00 out of 1.00

Complete the parser below such that it parses the yy/mm/dd date format (e.g. 22/01/14):

```
date = twoDigits `andThen` sepThenTwoDigits `andThen`
sepThenTwoDigits where
  twoDigits = pMap (\(a,b) -> [a,b]) (digit `andThen` digit)
  sepThenTwoDigits = (char '/') `pThen` twoDigits
```

Your answer is correct.

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 4
```

```
> :sprint ham3
ham3 =
```
3:_ ✗

```
> :sprint ham2
ham2 =
```
2:4:_ ✓