Basic
1 2 3 4 5
Intermediate
6 7 8
Advanced
9 10
Show one page at a time
Finish review

Completed on	Tuesday, 17 January 2023, 8:33 AM
Time taken	14 mins 57 secs
Grade	**9.50** out of 10.00 (**95**%)

Question **1**
Correct
Mark 1.00 out of 1.00
Flag question

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

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

a. digit
b. lower
c. char
d. upper ✔

Your answer is correct.

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] ✔

Question **3**
Correct

I want 1 bonus mark

Select one:

Correct

Mark 1.00 out of 1.00

Flag question

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

Answer: [3,6,9,12,15] ✔

Question **3**

Correct

Mark 1.00 out of 1.00

Flag question

I want 1 bonus mark

Select one:

- True ✔
- False

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.

Question **5**

Correct

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

Your answer is correct.

Question **5**
Correct
Mark 1.00 out of 1.00
Flag question

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

- a. String -> Result ParseError a
- b. String -> a
- c. String -> Result ParseError (a, String) ✔
- d. [Int] -> Result Int a

Your answer is correct.

Question **6**
Correct
Mark 1.00 out of 1.00
Flag question

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.

Question **7**

Question **7**

Correct

Mark 1.00 out of 1.00

Flag question

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

Your answer is correct.

Question **8**

Correct

Mark 1.00 out of 1.00

Flag question

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.

- [x] a. 1-2 ✔
- [x] b. abc ✔
- [ ] c. 1-
- [ ] d. Abc

Question **9**

Correct

Mark 1.00 out of 1.00

Flag question

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) ✔

Your answer is correct.

Question **10**

Partially correct

Given the following code that generates the hamming numbers:

```
merge3 x y z = merge (merge x y) z where
```

Partially correct

Mark 0.50 out of 1.00

Flag question

```
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:_ ✓

Your answer is partially correct.

You have correctly selected 1.