1. To compose two functions that return Result, with different error types we can:

Use case expression

Use Result.mapError

2.

- Function composition operator >> takes as first parameter a **function** and as second parameter a **function** and returns a **composition.**
- The pipeline operator |> takes as parameter first parameter a **value** and second parameter a **function** and returns a **function.**

3. To transform the value that is inside the Ok variant of Result, we can:

Use a case expression

Use Result.map function

4.

```
type alias Point = {a: Int, b: Int}

points = [{a = 1, b = 3}, {a = 2, b = 4}, {a = 3, b = 3}]

da : List Point -> List Point
da ps = ps |> List.map (\p -> { p | a = p.a + p.b })

db : List Point -> List Point
db ps = ps |> List.map (\p -> { p | b = p.b + p.a })

points |> db |> da |> List.map .b |> List.foldl (+) 0
```

16

5. If we want to write a function that calls divNums and returns Result String Float, we have the following options:

```
type CalculationError = FirstNaN | SecondNaN

divNums : Float -> Float -> Result CalculationError Float
divNums a b =
  if isNaN a then
    Err FirstNaN
  else if isNaN b then
    Err SecondNaN
  else
    Ok (a / b)
```

Use Result.mapError

Use a case expression to transform the error

6.

```
inc x = x + 1
dec x = x - 1
double x = x * 2
twice f x = f (f x)
```

What does the expression below evaluate to?

```
(twice (dec >> double >> inc)) 3
```

9

7.

Given the following function definition:

```
f x a b = x |> a |> b |> b
```

The result of the following expression is:

```
f 3 (\x -> x + 5) (\x -> x * 2)
```

32

8.

Given the following function definition:

```
f a b x = x |> b |> a
```

The result of the following expression is:

```
f (\x -> x + 2) (\x -> x * 3) 2
```

8

9.

Given the following definitions:

```
xs = [2, 1, 3]
```

Select the expression(s) which will produce the following result:

```
[1, 2, 3]
```

List.sort xs

List.reverse xs |> List.sort

List.sort <| List.reverse <| xs

10.

Given the following function definition:

```
f x a b = a <| b <| x
```

The result of the following expression is:

```
f 2 (\x -> x * 2) (\x -> x + 3)
```

10

11. In the context of functions used for testing HTML, select the functions that belong to HTML.Test.Selectors module:

Tag

Text

12. In the context of Elm web apps, the Msg type represents:

All possible actions that can cause the app to change its state

13.

```
type DivError = DivByZero

divNums : Int -> Int -> Result DivError Int
divNums a b =
  if a == 0 then
    Err DivByZero
  else
    Ok (b // a)
```

The type of the following expressions is:

`divNums 2 10 |> Result.mapError (\_ -> "Division by zero!")` **Ok 5 : Result String Int**

`divNums 2 10 |> Result.andThen (divNums 2)` **Ok 2 : Result DivError Int**

`divNums 2 10` **Ok 5 : Result DivError Int**

14. The function countVowels can be rewritten using pipelines as:

countVowels s = List.length (List.filter isVowel (List.map Char.toLower s))

countVowels s = s |> List.map Char.toLower |> List.filter isVowel |> List.length

countVowels s = List.length <| List.filter isVowel <| List.map Char.toLower <| s

15. Select all the true statements about the Elm runtime:

Handles communication with servers

Handles communication with browser

16. Select the true statements:

Record accessors can be composed with function composition

Elm has “built-in” definitions for all possible records accessors

17. Match the concepts with their definition:

Test.HTML.Selector **= Module which exposes predicates related to HTML nodes in web app testing**

Test.HTML.Query = **Module which exposes functions related to finding HTML nodes in web app testing**

18. To get the value that is inside the Just variant of Maybe or provide a default value, we can:

Use case expressions

Use the Maybe.withDefault function

19. Select all the false statements:

When we define a record, Elm generates its unique accessors that can’t be used to access other records

Trying to view the type of an accessor in the REPL will result in an error, because they have a special type

20. In the context of Elm web apps, the Model type represents:

The state of the app

21.

The result of the following expression is:

```
type alias Point = {x: Int, y: Int}

points = [{x = 3, y = 1}, {x = 3, y = 2}, {x = 3, y = 5}]

mx : Int -> List Point -> List Point
mx d ps = ps |> List.map (\p -> { p | x = p.x * d - 2 })

my : Int -> List Point -> List Point
my d ps = ps |> List.map (\p -> { p | y = p.y * d - 1 })

points |> mx 1 |> my 2 |> List.map .y |> List.foldl (+) 0
```

13

22. Fill the functions in the following code such that it tests that the view contains 3 h2 elements.

Import Test.Html.Query as Q

Import Test.Html.Selector as S

…

view

|> **Q** . fromHtml

|> **Q** . **findAll** [**S** . **tag** “h2”]

|> **Q** . **count** (Expect.equal 3)

23. Select all the true statements:

To pass an accessor to a map or filter, we use the . accessor syntax

We can use the same accessor to access fields that have the same name but are from different record types

24. Select all the true statements:

Records use structural typing

25.

The result of the following expression is:

```
type alias Point = {x: Int, y: Int}

points = [{x = 1, y = 2}, {x = 2, y = 4}, {x = 3, y = 5}]

mx : Int -> List Point -> List Point
mx d ps = ps |> List.map (\p -> { p | x = p.x + d })

my : Int -> List Point -> List Point
my d ps = ps |> List.map (\p -> { p | y = p.y + d })

points |> mx 1 |> my 2 |> List.map .x |> List.foldl (+) 0
```

9

26. The following code:

View = div[] [style “color” “red”, text “Some text”]

Will fail to compile