The following code snippet:

1. view = div [style "color" "red"] ["Some text"]

a.

Will render the text "Some text" with red color

b.

Will generate invalid HTML that causes the browser to show an error

c.

Will fail to compile

d.

Will render the text "Some text" with black color, because the style is not applied correctly

2. Select the correct way(s) of creating input fields using Elm.

a. input ["text" "email" email EmailChanged] []

b. inputAttrs ty p v msg = [ type\_ ty, placeholder p, value v, onInput msg ]

input (inputAttrs "text" "email" email EmailChanged) []

c. input ("text" "email" email EmailChanged) []

d. input [ type\_ "text", placeholder "email", value email, onInput EmailChanged] []

3. The result of the following expression is:

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

points [{1,4} {2, 6}, {3, 6}] -> points [{5,4} {8, 6}, {9, 6}] -> [4, 6, 6] -> 16

4. Function composition operator >> takes as first parameter a function and second parameter a function and returns a function

The pipeline operator |> takes as parameter first parameter a value and second parameter a function and returns a value

Note: first parameter is on the left hand side of the operator and second parameter is on the right hand side of the operator.

function/value

5. In the context of functions used for testing HTML, select the functions that belong to the Html.Test.Query module

a.class

b.tag

c.has

d.atLeast

e.find

6. 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

points = [{1,1}, {1,2}, {1,5}] -> points = [{1,1}, {1,3}, {1,9}] -> [1, 3, 9] -> 13

7. To transform the value that is inside the Just variant of Maybe, we can:

a. Use a case expression

b. Use the Maybe.map function

c. Use the Maybe.withDefault function

d.Use the Maybe.transform function

8. Given the following definitions:

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)

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

1. Use Result.map
2. Use Result.withDefault
3. Use a case expression to transform the error
4. Use Result.mapError

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

a. Data which the browser needs to correctly display the app

b. Internal data used by the Elm runtime

c. The current view

d. The state of the app

10. The function countPalindromes can be rewritten using pipelines as:

countPalindromes s = List.length (List.filter isPalindrome (List.map Char.toLower s))

1. countPalindromes s = s |> List.map String.toLower |> List.filter isPalindrome |> List.length
2. countPalindromes s = List.map String.toLower <| List.filter isPalindrome <| List.length <| s
3. countPalindromes s = List.length <| List.filter isPalindrome <| List.map String.toLower<| s
4. countPalindromes s = s |> List.length |> List.filter isPalindrome |> List.map String.toLower