Functional Languages 2021 Exercise 1 (15 Points)

- **Exercise 1: Records.** (1 Point) Define a record `Point` with two fields `X` and `Y`, each of type `float`, which represent the point's 2D coordinates.

- **Exercise 2: Accessing record fields.** (1 Point) Define a function `distance` of type `distance : Point -> Point -> float` which calculates the distance between two points. The formula for the distance is `sqrt(dx * dx + dy * dy)`, where `dx` and `dy` are the differences between the points' X and Y coordinates, and `sqrt` is the already-defined square root function.

- **Exercise 3: Unions.** (2 Points) Define a union `Geometry` which has three cases:

  - Vertex, which takes a `Point` to indicate its location
  - Line, which takes two `Point`s as its start and end locations
  - Circle, which takes a `Point` and a `float` as center and radius

- **Exercise 4: Pattern Matching.** (2 Points) Define a function `size` of type `size : Geometry -> float` which calculates the size of a geometry. For a vertex, the size should be 0.0. For a line, the size should be its length (the distance between start and endpoint). For a circle, the size should be its diameter (2.0*radius).

- **Exercise 5: Option.** (2 Point) Define a function `divSafe` of type `divSafe : int -> int -> Option<int>` which performs integer division if possible. The function should divide the enumerator by the denominator and return `Some` result, unless the denominator is zero, in which case we want to return `None`. (You can use F#'s `if ... then ... else ...` syntax)

- **Exercise 6: Discussion.** (1 Point) Write a comment in which you describe what a *total function* is. Looking at the previous example, the regular integer divison is not total since it can't produce a value when the denominator is zero (it throws an exception instead).

- **Exercise 7: Working with Options.** (2 Points) Define a function `optionTimesTwo` of type `optionTimesTwo : Option<int> -> Option<int>` which doubles the value of the integer contained inside the option. In case the optional value contains Some integer, the result should contain the integer times two. In case the optional value was None, the result should be None.

- **Exercise 8: Mapping Options.** (2 Points) Define a function `optionMap` of type `optionMap : ('a -> 'b) -> Option<'a> -> Option<'b>'` which applies an input function to the value contained inside the option. In case the optional input was Some value, the result should be the result of the function applied to the value. In case the optional value was None, the result should be None.

- **Exercise 9: Partial Application.** (2+1 Points) Implement or re-implement Exercise 7 using the mapping function from Exercise 8.

(*You can submit Exercise 9 instead of Exercise 7 to get points for both Exercise 7 and Exercise 9.*)

- **Sumbission.** Submit your solution as zip file containing the following text files:

  - `exercise1.fs` or `exercise1.fsx`

  (*If you don't like F#, feel free to use any other typed functional programming language instead.*)