Solved exercises

- **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.

Solution:

```
type Point =
    {
        X : float
        Y : float
    }
```

- **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.

Solution:

```
let distance (p0 : Point) (p1 : Point) =
    let dx = p0.X - p1.X
    let dy = p0.Y - p1.Y
    sqrt(dx * dx + dy * dy)
```

- **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

Solution:

```
type Geometry =
| Vertex of Point
| Line of Point * Point
| Circle of Point * float
```

- **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).

Solution:

```
let size (g : Geometry) =
    match g with
    | Vertex _ -> 0.0
    | Line (p0,p1) -> distance p0 p1
    | Circle (_,r) -> 2.0 * r
```

- **Exercise 5: Option.** (1 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)

Solution:

```
let divSafe (e : int) (d : int) =
    if d <> 0 then Some(e/d)
    else None
```

- **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).

Solution:

```
// a total function produces a value for every possible input
```

- **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.

Solution:

```
let optionTimesTwo (o : Option<int>) =
    match o with
    | Some o -> Some (2*o)
    | None -> None
```

- **Exercise 8: Mapping Options.** (2 Points) Define a function `optionMap` of type `optionMap : ('a -> 'b) -> Option<'a> -> Option<'b>` which applies a 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.

Solution:

```
let optionMap f o =
    match o with
    | Some x -> Some (f x)
    | None -> 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.*)

Solution:

```
let optionTimesTwo' = optionMap ( fun x -> 2*x )
```