Lab – 8

Week of Mar. 11, 2018

1 Instructions

1. Use the OCaml top-level to develop and debug your code.
2. You may assume that all inputs are valid unless otherwise stated in the problem.
3. In questions that require string outputs, be careful not to include leading or trailing whitespace.
4. You may submit and evaluate your code a *maximum* of 15 times without penalty. Subsequently, you will lose 2 marks per additional evaluation. Therefore, please ensure that you have thoroughly debugged your code before submitting.

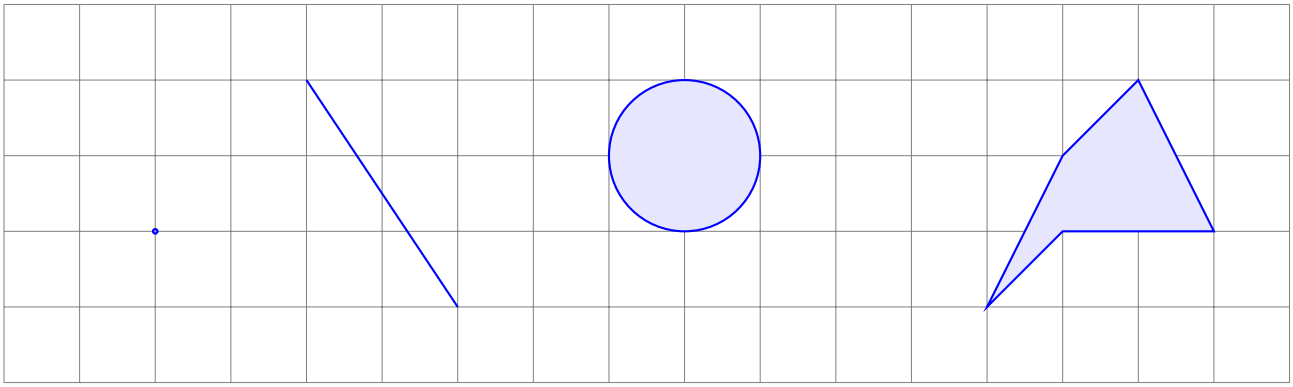
The following submission file is *required*:

1. `geometry.ml`

2 Assignments

This week you will implement a module called `Geometry`, to handle some simple geometric shapes. Specifically, 2D shapes. Your module must support the following types of shapes.

1. **Point**: A 2D point
2. **Line**: Defined by 2 2D points
3. **Circle**: Defined by its centre (a point in 2D) and its radius
4. **Polygon**: Defined by n points



The above grid shows the following shapes (from left to right):

1. **Point**: `(2,2)`
2. **Line**: endpoints - `(4,4)`, `(6,1)`
3. **Circle**: centre - `(9,3)`, radius - `1`
4. **Polygon**: vertices in cyclic order - `(13,1)`, `(14,3)`, `(15,4)`, `(16,2)`, `(14,2)`

The inputs to your functions will involve reading shapes from a file as follows.

1. Each line of the file takes the following form: An integer n denoting the n^{th} shape, followed by a colon `:` and then listing the points required by the shape. For example:

   (a) `1: (0,0), 5` – the first shape is a circle, centred at `(0,0)`, with radius 5 units

   (b) `2: (1,1), (4,4)` – a line

   (c) `3: (2,3), (5,6), (1,1)` – this is a polygon whose corners are given by the points.

2. A line in the file can also correspond to one of the following 'action' commands:

   (a) `P <#shape>` – asks that you output the perimeter of the shape corresponding to the shape number

   (b) `A <#shape>` – asks you output the area of the shape corresponding to the shape number

   (c) `T <#shape>` – asks for the kind of shape. The possible responses are: "P" (point), "C" (circle), "L" (line), "PLG <#n>" (polygon with n sides)

   (d) `D <#shape1> <#shape2>` – asks for the distance between the *centroids* of shape 1 and shape 2. Note that the distance between two lines is just the distance between their mid-points

The output of your program should simply write onto the terminal outputs for *each* command, one per line. Note that your answer type for the actions `P`, `A` and `D` should be of `float` type and your answer will be considered correct if it is correct upto 3 decimal places.

Sample Input:

```
1:  (0,0), 5
A 1
2:  (1,1), (4,4)
3:  (2,3), (5,6), (1,1)
T 1
P 2
4:  (0,0), (1,0), (1,1), (0,1)
T 4
A 3
D 2 4
```

Sample Output:

```
78.53981634
C
4.242640687
PLG 4
1.5000
2.828427125
```