Lab 2
Due: May 26, 2023 at the end of lab

1 Lab 2 Specification

1.1 Introduction

The skeleton code at `https://classroom.github.com/a/jcy4-Mrz` is the start of today's lab. Follow the setup and import directions.

1.2 Implementation

Write and test the following functions. If you want to include type information in the function signatures, feel free to add it. Do not change the function names though. Use the examples to check your solutions, but write more tests of your own to ensure your functions work correctly. Assume that the functions parameters will always be the correct type.

Reminder: only use `;;` in interactive mode – do not include expression endings in your submitted code.

1.2.1 addTuples

Define the function `addTuples` to add 2-tuples of ints by adding each element. Return the resulting tuple.

```
> addTuples (3, 4) (5, 6);;
val it : int * int = (8, 10)
```

1.2.2 quadratic

Define the function `quadratic` to calculate $(a + b)^2$. Assume that the input a is a 2-tuple of ints.

```
> quadratic (5, 2);;
val it : int = 49
```

1.2.3 hypot

Define the function `hypot` to calculate the corner-to-corner distance in a 3D box. The length of the three sides are given as a tuple named a, and the distance can be calculated as hypot$= \sqrt{a_1^2 + a_2^2 + a_3^2}$. F# can calculate a square root with the built-in `sqrt` function. Assume that the input is a 3-tuple of positive floats.

```
> hypot (6.0, 18.0, 13.0);;
val it : float = 23.0
```

1.2.4 third

Following the example of the notes' function `halve` in lecture 3, slides 29–31, define the function `third` that divides a list into a tuple of three nearly-equally-sized lists. Make sure that your function matches all the possible list patterns. In the special case of an empty list, `third` will fail like `halve` fails. Your function should succeed in all cases when the input list has a length of at least 1.

```
> third [1;2;3;4;5;6;7;8;9;10];;
val it : int list * int list * int list = ([1;4;7;10], [2;5;8], [3;6;9])
```

2 Grading

Before submitting, confirm that your return types match the examples shown above. The types should match exactly.

Be professional. Make results easy to understand and grade. Include only those parts to be graded. Leave comments where necessary, especially if it aids in grading.

Each of the 4 functions is worth $\frac{1}{4}$ of the lab grade.