Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

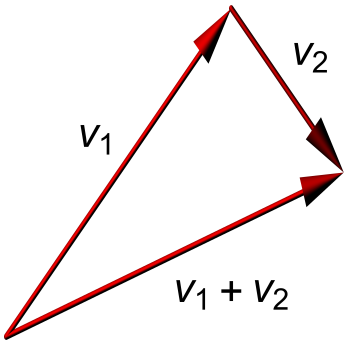

Figure 1: Illustration of vector addition in two dimensions.

0.1 Vector v.2

0.1.1 Teacher's guide

0.1.2 Introduction

This assignment is about 2-dimensional vectors. A 2-dimensional vector or just a vector is a geometric object consisting of a direction and a length. Typically, vectors are represented as a coordinate pair $\vec{v} = (x, y)$. The vector's ends are called its tail and tip, and when the tail is placed in $(0, 0)$, then its tip will be in the (x, y). Vectors have a number of standard operations on them:

$$\vec{v}_1 = (x_1, y_1) \tag{1}$$
$$\vec{v}_2 = (x_2, y_2) \tag{2}$$
$$\vec{v}_1 + \vec{v}_2 = (x_1 + x_2, y_1 + y_2) \tag{3}$$
$$a\vec{v}_1 = (ax_1, ay_1) \tag{4}$$

Addition can be drawn as shown in Figure 2.1. Rotation of a vector counter-clockwise around its tail by a can be done as,

$$R_a\vec{v}_1 = (x\cos(a) - y\sin(a), x\sin(a) + y\cos(a)) \tag{5}$$

In F#, the trigonometric functions are found in `cos` and `sin`, and they both take an angle in radians as the argument. The constant π is found in `System.Math.PI`. In the following, we will use the type abbreviation:

```
type vec = float * float
```

0.1.3 Exercise(s)

0.1.3.1: Using Canvas, you are to draw vectors. For this,

(a) Make a function

```
toInt: vec -> int * int
```

which takes a vector of floats and returns a vector of ints.

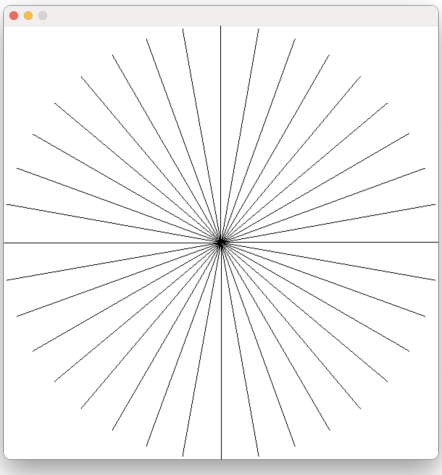

Figure 2: 36 radial lines from the center of a canvas.

(b) Using `add` and `toInt`, make a function

```
setVector: canvas -> color -> vec -> vec -> unit
```

which takes a canvas, a color, a vector `v`, and a position `p` and draws a line from `p` to `p+v` using `setLine`. Demonstrate that this works by creating a horizontal vector with its tail at the center of the canvas, and show it on screen using `show`.

(c) Using `rot` and `setVector` make a function

```
draw: int -> int -> canvas
```

which creates a canvas with a given width and height, adds 36 spokes as illustrated in **??**, and returns the canvas. Demonstrate that this works by showing the canvas on screen with `show`.

(d) Optional: Use these in `runApp` to make an interactively rotating set of spokes as follows: Extend `draw` with a float state parameter `s`, which draws the spokes with the angular offset `s`. Add a reaction function `react` which changes the offset by ± 0.01 when the right and left arrow key are pressed respectively.

The functions are to be documented using the `<summary>`, `<param>`, and `<returns>` XML tags.

0.1.3.2: In the following, you are to work with tuples.

(a) Make a type abbreviation called `vec3`, which is a 3-tuple of `floats`.

(b) Make a value of `vec3`.

(c) Make a function, which takes a `vec3` as argument and returns the squared sum of its elements, and test it on the value, you created. Consider the different ways, the function's type could be written, and what their qualitative differences would be.

0.1.3.3: In an earlier assignment, you implemented a small set of functions for vector operations in F#:

(a) addition of vectors

```
add: vec -> vec -> vec
```

(b) multiplication of a vector and a floating-point number

```
mul: vec -> float -> vec
```

(c) rotation of a vector by radians

```
rot: vec -> float -> vec
```

and wrote a small program to test these functions. Convert your earlier programs into a library called `Vec`, consisting of a signature file, an implementation file, and an application file. Create a `.fsproj` project file, and run the code using first `dotnet fsi` and then `dotnet run`.

0.1.3.4: Building upon the exercises concerning vectors and their representation as tuples, we will construct a small library of functions for 2-dimensional vector operations, with vectors represented as tuples of floats.

(a) Write a function:

```
scaleFloatVec : float * float -> scalar:float -> float * float
```

that given a vector of floats and a scalar, scales the vector and returns the resulting vector.

(b) Write a function:

```
addFloatVecs : float * float -> float * float -> float * float
```

that given two vectors of floats, adds them and returns the resulting vector.

(c) Write a function:

```
subFloatVecs : float * float -> float * float -> float * float
```

that given two vectors of floats, subtracts them with vector subtraction and returns the resulting vector.

(d) Write a function:

```
lengthFloatVec : float * float -> float
```

that given a vectors of floats, computes the length of the vector and returns the result as a float.

(e) Write a function:

```
dotFloatVecs : float * float -> float * float -> float
```

that given two vectors of floats, computes the dot product and returns the result as a float.

0.1.3.5: Using Steps 1, 3, 5, 7, and 8 from the 8-step guide to write a small set of functions in F#:

(a) addition of vectors (**??**)

```
add: vec -> vec -> vec
```

(b) multiplication of a vector and a constant (**??**)

```
mul: vec -> float -> vec
```

(c) rotation of a vector (**??**)

```
rot: vec -> float -> vec
```

The functions are to be documented using the `<summary>`, `<param>`, and `<returns>` XML tags.