Mission 4: Curve Introduction

Start date: 24 August 2016
Due: 29 August 2016, 23:59

Readings:

- Textbook Sections 1.1.5 to 1.1.8

Drawing Curves

A *curve* is a basic drawing element. Complex pictures can be obtained by drawing multiple curves. A curve is a unary function which takes one real argument within the unit interval `[0,1]` and returns a point (a pair of real numbers).

Intuitively, the real argument to a curve can be thought of as time, and the point returned by the curve can be thought of as the position of the pen at the time indicated by the argument. We represent curves by functions of Source type Curve, where

```
Curve:  Number -> Point
```

and Point is some representation of pairs of Numbers. If `C` is a curve, then the starting point of the curve is always `C(0)`, and the ending point is always `C(1)`.

To work with Points, we need a *constructor*, `make_point`, which constructs Points from Numbers, and `selectors`, `x_of` and `y_of`, for getting the `x` and `y` coordinates of a Point. We require only that the constructors and selectors obey the rules

```
x_of(make_point(n,m)) === n;
y_of(make_point(n,m)) === m;
```

for all Numbers m,n.

We can write down the *types* of these functions as follows:

```
make_point :  (Number, Number) -> Point
x_of, y_of :  (Point) -> Number
```

We also define the Curve `unit_circle` and the Curve `unit_line` (along the x axis):

```
function unit_circle(t) {
    return make_point(Math.sin(2 * Math.PI * t),
                      Math.cos(2 * Math.PI * t));
}

function unit_line_at(y) {
    return function(t){
        return make_point(t, y);
    };
}

var unit_line = unit_line_at(0);
```

Drawing Functions

In the previous section we introduced the concept of curve as the basic drawing unit. However, in order to actually draw a curve on the window, you will need a drawing function. It is not required that you understand the implementation of the drawing functions, but you should know how to use them in order to visualize and test your solutions.

When you begin your mission, you should see an empty Canvas.

In the Interpreter, type:

```
(draw_connected(200))(unit_circle);
(draw_connected_squeezed_to_window(200))(unit_circle);
```

After the first `draw` statement, you will see a quarter circle, and after the second, a full circle. Note that all Curve functions accepted by the `draw` methods are of the form defined above:

`Curve:  Number -> Point`

The 200 in `draw_connected` refers to the number of points to draw on the screen. Since `1/200 = 0.005`, `unit_circle` will be called for values of `t = 0, 0.005, 0.01, 0.015, 0.02`, etc. until `t = 1`. `draw_connected` will then join two adjacent points returned by `unit_circle` to draw a connected Curve. If you want to draw the points without connecting them, use `draw_points_on`. The following is an example of how to use `draw_points_on`:

```
(draw_points_on(200))(unit_circle);
```

Note that the origin of the drawing window is at the bottom-left. Moving right along the drawing window increases the value of the
x-axis until the x-coordinate equals 1. Likewise moving up along the window increases the value of the y-axis until the y-coordinate equals 1. This is why `draw_connected` shows only a quadrant of the `unit_circle`, since for some values of `t`, the x- and y-coordinates are outside the range `[0, 1]`. For example, try:

```
unit_circle(0.5);
```

The y-coordinate of `-1.0` is outside the range `[0, 1]` and hence cannot be displayed. Another function, `draw_connected_squeezed_to_window`, takes care of this by scaling and translating the Curve as required so that all points fall in the range `[0, 1]` for both axes.

More on Drawing Curves

Now that you have learnt the basics on how to visualise curves on the window, Grandmaster Martin would like you to take your drawing skills to the next level. There are other drawing functions, listed as follows:

1. `draw_points_on`
2. `draw_connected`
3. `draw_points_squeezed_to_window`
4. `draw_connected_squeezed_to_window`

The differences between these functions are suggested by their names. They are used in a similar way. For example, to draw the pre-defined curve `unit_circle` on the window using `200` points, you may call:

```
(draw_points_on(200))(unit_circle);
```

In order to draw a connected curve, you may use:

```
(draw_connected(200))(unit_circle);
```

If you want to make the curve fit in the current window, you may use:

```
(draw_connected_squeezed_to_window(200))(unit_circle);
```

Note that the more points you use, the more accurately the curve will be drawn.

This mission has **two** tasks.

Task 1:

Recall the definition of `unit_line_at`:

```
function unit_line_at(y){
    return function(t){
        return make_point(t, y);
    };
}
```

1. What is the type of `unit_line_at`?
   Hint: The format for expressing a type and an example is shown below:
   ```
   <function> :  <argument-type>[, ...]  -> <return-type>
   make_point :  (Number, Number) -> Point
   ```
2. Define a function `vertical_line` which takes two arguments, a point and a length, and returns a vertical line of that length beginning at that point. Note that the line should be drawn upwards (i.e., towards the positive-y direction) from the point.
3. What is the type of `vertical_line`?
4. Using `draw_connected` and your function `vertical_line` with suitable arguments, draw a vertical line in the drawing window which is centered horizontally and vertically and has half the length of the sides of the drawing window.

Task 2:

Apply `draw_connected(200)` to `unit_circle`, i.e.

```
(draw_connected(200))(unit_circle);
```

Then apply `draw_connected(200)` to `alternative_unit_circle`. Can you see a difference? Now try using `draw_points_on` instead of `draw_connected`.

Also try `draw_points_squeezed_to_window`.

Use appropriate drawing functions to draw `unit_circle` and `alternative_unit_circle`. Write down the difference between `unit_circle` and `alternative_unit_circle`. You should also point out why this difference exists by examining the implementations of both `unit_circle` and `alternative_unit_circle`. You may do this by executing `unit_circle;` and `alternative_unit_circle;` in the Interpreter.

Submission

To submit your work to the Source Academy, place your program in the "Source" tab of the online editor within the mission page, save the program by clicking the "Save" button, and click the "Submit" button. Please ensure the required function from each Task is included in your submission. Note that submission is final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.

IMPORTANT: Make sure you've saved the latest version of your work by clicking the "Save" button before finalizing your submission!