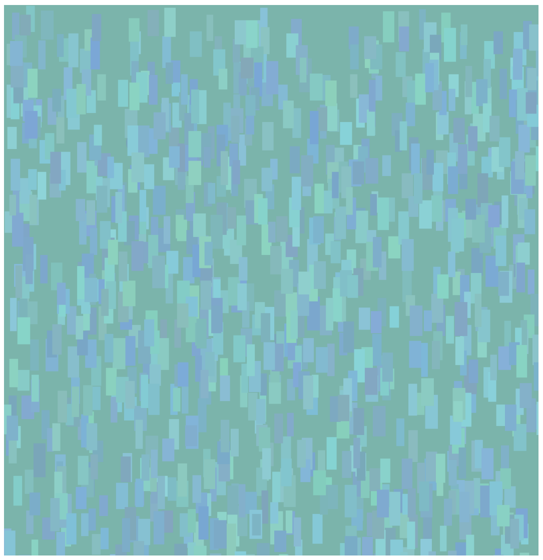
A Tutorial to Panko

I. What is Panko?

Panko (Paint n Kode) is a programming language that allows users to create paintings using digital brush strokes. Panko is also a Japanese-style of bread crumbs, which means that the author did not spell Code with a K just to be quirky. Panko strives to be a true painting language by both making the process of creating a Panko program resemble the physical process of painting and allowing more organic images to be produced.

So that's the sales pitch, but let's see what Panko can do! Below is a sample program written in Panko and its result.



```
canvas 800 800
background 6AB6AB

coolRectBrush = stroke rectangle 10 20 30 50
colorOne = 7DD7DD
colorTwo = 6BA6BA

repeat 1000 (
x = randLoc 0 800
y = randLoc 0 800
gradient coolRectBrush colorOne colorTwo x y
)
```

These lines might make no sense to you right now, but let's break it down! For this portion of the tutorial, it is highly recommended that you open up your favorite text editor and type along. At the end, you will have a painting that looks somewhat (why "somewhat" and not "exactly" will be explained later) like the sample painting above.

II. Setting up Panko

Open up a terminal window and navigate to the `lang` directory. To set up Panko, open up your favorite text editor and create a file called `HelloPanko.pk` in the `lang` directory. This is the file that we'll be writing our very first Panko program in.

To run the program (do not do this yet), type in the terminal

```
> dotnet run HelloPanko.pk
```

The command above will create a directory called `output` within the lang directory and save the output of the program to a file called `HelloPanko.html` in the output directory.

If you try to run Panko with an empty program (i.e. right now), Panko will complain that the program is invalid. This is because every Panko program requires two lines specifying the canvas and the background to work correctly.

One note before we start typing: Panko is new-line sensitive. It doesn't really care how many new lines you have between lines of code, but they must be on *different lines*. Something like `canvas 800 800 background 6AB6AB` would *not* work!

Read on and follow along!

III. Painting with Panko

A program in Panko can be broken down into two main steps: gathering the materials (the canvas and the brushes) and painting (the paint and gradient functions).

First, let's take a look at our materials: the canvas and the brushes.

1. The canvas and the coordinate system

Panko uses the coordinate system to draw onto the screen. This coordinate system is different from the one you've probably seen in Math class. In Panko, point (0,0) is the top-left point of the screen. The unit of the canvas is pixel.

In Panko, creating a canvas is *required*. To make a canvas, we specify its width and height. Let's set the size for our canvas to be 800-by-800 pixels.

```
canvas 800 800
```

Now we had the first line of the sample painting above! Let's make the rest of the painting one line at a time.

2. The background

Sometimes we might want to fill the entire canvas with a solid color to save ourselves some work later. In Panko, setting the background color is *required*.

Colors in Panko are represented by 6-digit hex values (000000 - FFFFFF). This color picker can be a great resource to find hex values of colors.

Let's paint the background a light teal color.

```
background 6AB6AB
```

Note that these two lines of code above are (1) required for any Panko program and (2) must be in that exact order. Afterall, we can't paint the canvas *before* getting the canvas!

Now that our program has these two lines, type in the terminal

```
> dotnet run HelloPanko.pk
```

Navigate to the `lang/output` directory and click on the file `HelloPanko.html`. This should open up your browser window with a teal square.

From this point of the tutorial, everytime you want to run the program simply type `> dotnet run HelloPanko.pk` and either open the `HelloPanko.html` file again or refresh the browser window.

3. The brushes

a. The simple brush

A brush is essentially a shape with some *predefined* size (not really, but we'll get to the other kind of brushes later). In Panko, all brushes must have names assigned to them.

Let's make a rectangle-shaped brush with width 100px and height 300 px. Let's also call it `rectBrush`.

```
rectBrush = stroke rectangle 10 30
```

Panko also has a circle-shaped brush. This following line of code creates a circle-shaped brush with radius 100.

```
myCircleBrush = stroke circle 100
```

But let’s stick with the rectangle brush for now.

b. The magic brush

Notice in real life how brush strokes never turn out the same depending on how much pressure you put on the brush? Well, that's the beauty of painting! Panko has a special version of brushes that will help you achieve this effect.

This following line of code will create a rectangle-shaped brush stroke that has width between 10 and 20 pixels and height between 30 and 50 pixels. Everytime we paint using this brush, the width and height will be randomly generated. Let’s go back and change our simple brush to this new magic brush. Simply delete `rectBrush = stroke rectangle 10 30` and type in:

```
coolRectBrush = stroke rectangle 10 20 30 50
```

Take a look at the output file. Do you see anything new? No? Well that’s because we’ve only created the brush and have not *used* it yet.

4. The paint function

The `paint` function requires four pieces of information: the brush, the x-coordinate, the y-coordinate, and the color. It places the brush of our choice at position (x,y) with the specified color.

For rectangle-shaped brushes, the position of the brush is the top-left of the rectangle. For circle-shaped brushes, the position of the brush is the center of the circle.

Recall that the color is a hex color code with 6 digits, e.g. 7DD7DD. We can also give this color a name so that it's easier for us later when we have multiple colors.

```
colorOne = 7DD7DD
```

Now everytime you want to use the color 7DD7DD, just substitute in its name instead.

Let's paint a single teal stroke on our canvas:

```
paint coolRectBrush colorOne 200 300
```

Your output file should have a small rectangle now. Yay! Now we only need to do this ....1000 more times?

5. The repeat command

To repeatedly paint a brush stroke some number of times, simple wrap all your lines of code in a `repeat` block.

```
repeat 1000 (
paint coolRectBrush colorOne 200 300
)
```

Take a look at your output file. Do you see only one brush stroke still? This is because we've been layering 1000 brush strokes in the same position (200,300).

6. randLoc

To vary the position of the brush stroke, use `randLoc` to specify the x and y coordinates. `randLoc` takes two arguments, a lower bound and an upper bound, and will return a random number within that range (lower bound is inclusive while upper bound is exclusive). Just like the brush strokes, you have to assign `randLoc` to names. Let's just call our x and y coordinates x and y.

```
x = randLoc 0 800
y = randLoc 0 800
```

randLoc and the `repeat` command really shine together.

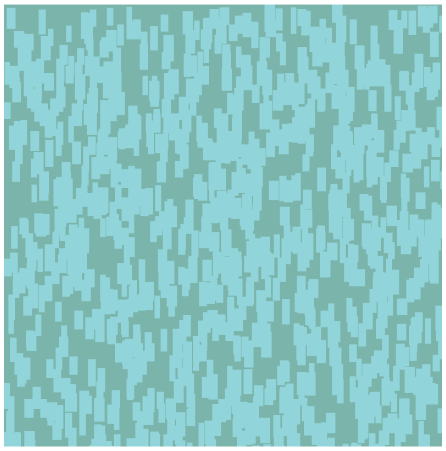
```
repeat 1000 (
x = randLoc 0 800
y = randLoc 0 800
paint coolRectBrush colorOne x y
)
```

Phew, that was a lot. Here's everything we have so far. You should see something like the output on the right. Your output might differ, because we've been using the `randLoc` function.

```
canvas 800 800
background 6AB6AB

coolRectBrush = stroke rectangle 10 20 30
50
colorOne = 7DD7DD

repeat 1000 (
x = randLoc 0 800
y = randLoc 0 800
paint coolRectBrush colorOne x y
)
```



7. The gradient function

Sometimes we might want our brush strokes to vary in color from one stroke to the next. The `gradient` function allows us to do so. It takes 5 pieces of information, the first 3 are identical to the paint function (namely, the brush, the x-coordinate, the y-coordinate) but instead of requiring one solid color, the `gradient` function requires 2 colors.

The resulting brush stroke will have the color that is somewhere in between the two specified colors.

What does it mean to have a color in between 2 other colors? Well, mathematically, the new color has its (r,g,b) values between those of the 2 specified colors. This concept is definitely not intuitive (i.e. it's not easy to imagine what the in-between colors of two colors would be, given their hex codes, and the results are usually surprising), so just experiment!

Let's paint with this `gradient` function. Name another color, `colorTwo = 6AB6AB` outside of the `repeat` block. Replace `paint coolRectBrush colorOne x y` in the repeat block with

```
gradient coolRectBrush colorOne x y
```

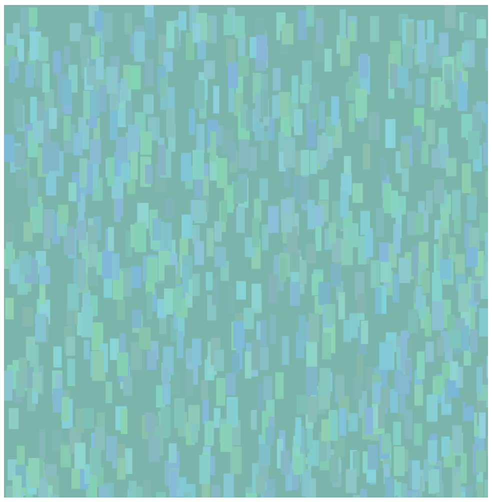
Note that for the line of code above, you can also supply the actual color value, the x-coordinate, and the y-coordinate values rather than their names. But since we're wrapping it in the `repeat` block, `x` and `y` must be names for the `randLoc` function to work.

And you're done! Here's all the code and the resulting image.

```
canvas 800 800
background 6AB6AB

coolRectBrush = stroke rectangle 10 20 30 50
colorOne = 7DD7DD
colorTwo = 6BA6BA

repeat 1000 (
x = randLoc 0 800
y = randLoc 0 800
gradient coolRectBrush colorOne colorTwo x y
)
```



Notice how this image doesn't look exactly like the one at the beginning of the tutorial. That is because we incorporated several elements of randomness in our painting: (1) the size of the brush stroke via the magic brush, (2) the random positions of brush strokes via randLoc, and (3) the varied colors of the brush strokes via the gradient function.

And those are all the elements that make up Panko! Now let's paint away!

The full program that we've written above is also included in the `lang` directory, along with two other example programs: `Primary.pk` and `FlowerField.pk`.