SDES3107

INTRODUCTIONTOPROCESSING

WEEK 3

MONDAY 9-12PM

D103

WEEK 3

TOPICS COVERED:

TYPOGRAPHY Load and diplay text

RANDOM Unexpected numbers, noise

TRANSFORMATIONS Transform, matrix, rotate, scale

TRIGONOMETRY Angles, waves, circles, arcs, spirals

TYPOGRAPHY (p.111-116)

Loading a font is very similar to what we have already seen with images.

To load a font we use the PFont data type and the loadFont() function

```
PFont font;
font = loadFont("Ziggurat-32.vlw");
```

The font file must be in the sketch's data folder and the name used in the loadFont function must exactly match the name of this file.

3 ways to place a font file into your sketch's data folder:

1) Place the file into the data folder.

2) Drag and drop the font file onto the code area of your Processing window.

3) Navigate to: tools/create font - in the Processing menu

TYPOGRAPHY (p.111-116)

Once you have loaded a font it is then time to display it. To display a font we will need to display text.

We set the textFont(font) so any text that follows this function will be displayed in the font referred to.

To display text we use the text function

text(“Hello“, 20, 50);

Where the first parameter is the string that is being displayed and the next two coordinates are the x and y positions.

Want to have multiple fonts to display?

Create multiple PFont variables that are all loaded with different font files.

Use the textFont() function to alternated between these fonts.

Like fill() the textFont() function will effect everything that comes after it until it is changed.

TEXT ATTRIBUTES (p.114-116)

There are functions that can effect the styling of your text. These include:

textSize() textLeading() textAlign()

These functions SET the attributes for the text that follows them.

textWidth() This function RETURNS the width in pixels of the text that is being displayed.


```
PFont font;
font = loadFont("Ziggurat-32.vlw");
textFont(font);
fill(0);
char c = 'U';
float cw = textWidth(c);
text(c, 22, 40);
rect(22, 42, cw, 5);
String s = "UC";
float sw = textWidth(s);
text(s, 22, 76);
rect(22, 78, sw, 5);
```

EXERCISE

13.2. Draw a paragraph of text to the display window. Carefully select the composition.

(p.130-131)

RANDOM (p.127-130)

We can use a function in Processing that generates **random values**. This enables us to create more unique and unexpected compositions.

The following **generate random values** depending on what parameters we use:

```
random(high);
```

from 0.0 to whatever value 'high' is

```
random(low, high);
```

from 'low' to whatever value 'high' is

Whatever we put into random() must be a float (decimal) value!

```
float f = random(5.2);
```

```
int i = random(5.2);
```

wrong! we can't assign a float to an int

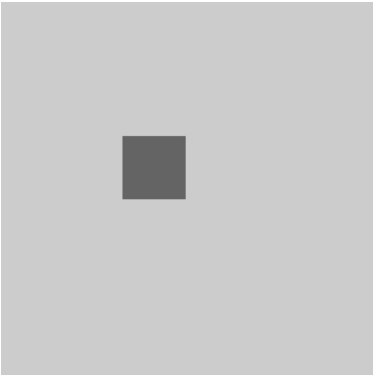
```
float j =
int(random(5.2);
```

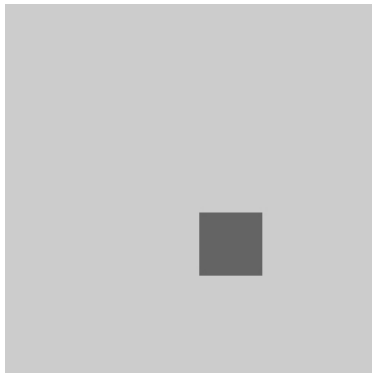
but we can convert a float to an int!

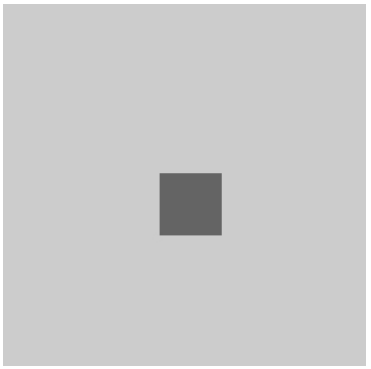
RANDOM (p.127-130)

For example, if we wanted to draw a rectangle at random positions in our sketch, we could write:

```
size(300,300);
int randomPos = int(random(300.0));
fill(100);
noStroke();
rect(randomPos,randomPos+10,50,50);
```







Because we've converted that float to an int, we can use it to set random positions each time we press run. We can use this to randomise any attribute we like! (colour, shape, size, position, etc...)

randomSeed(0) makes the same value each time the program runs (makes it random only once).

NOISE (p.130-131)

Using a Perlin Noise function, we can generate more controllable random numbers.

The noise() function generates a series of random numbers from 0.0 - 1.0 depending on what we put as a parameter:

noise(x);

We use noiseSeed(0) to force the same numbers to be generated each time the program is run.

```
size(600, 100);
float v = 0.0;
float inc = 0.1;
noStroke();
fill(0);
noiseSeed(0);

for (int i = 0; i <
width; i = i+4) {
      float n = noise(v) *
      70.0;
      rect(i, 10 + n, 3,
      20);
      v = v + inc;
}
```

EXERCISE

15.1. Use three variables assigned to random values to create a composition that is different every time the program is run.

(p.130-131)

TRANSLATION (p.133-134)

Translations are ways to ‘transform’ or change the position, orientation and scale of things on screen. So far we have been doing this by changing the parameters inside the functions:

```
rect(10,10,30,30);
rect(20,10,30,30);
```

Above: the second rectancle has been moved 10 pixels to the right.

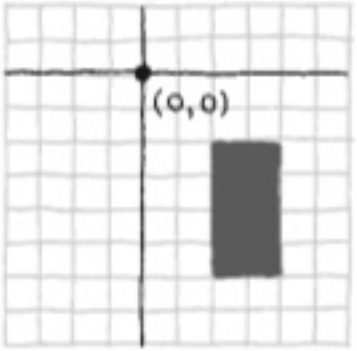
```
translate(10,0);
rect(10,10,30,30);
```

Above: this rectangle will be drawn at the same position as the second rectangle in the first example.

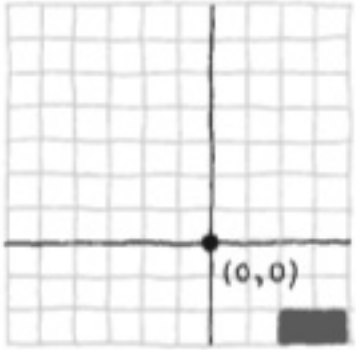
TRANSLATION (p.133-134)

What is happening? When we ‘translate’ we are actually moving the coordinate system

```
translate(40, 20);
rect(20, 20, 20, 40);
```




```
translate(60, 70);
rect(20, 20, 20, 40);
```



We can move a complex composition of images + text + shapes by simply translating the coordinate system

translate(x,y);

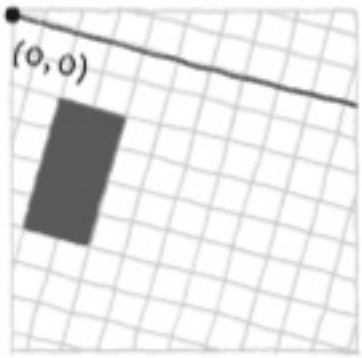
and not each variable inside each element of our composition.

Also note that each translation is accumulative.

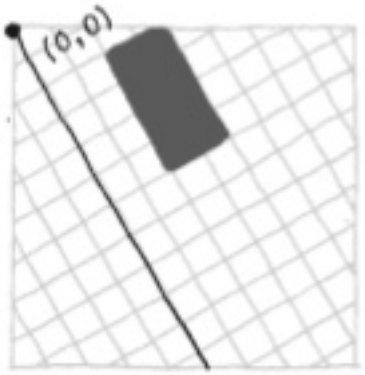
translate(10,0);
translate(10,0);

ROTATE AND SCALE (p.133-134)

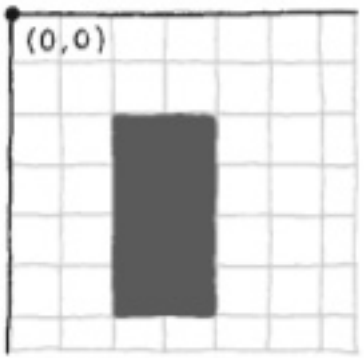
```
rotate(PI/12);
rect(20, 20, 20, 40);
```



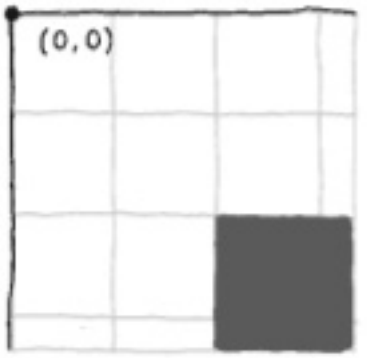

```
rotate(-PI/3);
rect(20, 20, 20, 40);
```




```
scale(1.5);
rect(20, 20, 20, 40);
```




```
scale(3);
rect(20, 20, 20, 40);
```



scale(angle);

rotate(size);

size refers to a percentage
as a float 2.0 is 200%

MATRIX (p.134-135)

The matrix refers to the cooordinate system. When we translate the coordinate system we are changing the ‘matrix’.

pushMatrix(); is a function that remembers the ‘current’ position of the coordinate system.

```
pushMatrix();
translate(10,0);
rect(0,0,10,10);
popMatrix();
```

popMatrix(); is a function that must be called for every pushMatrix() that is used.

popMatrix(); puts the coordinate system back to its position at the point where the proceding pushMatrix was called.

```
pushMatrix();
translate(10,0);
rect(0,0,10,10);
popMatrix();
rect(0,0,10,10);
```

EXERCISE

16.2. Use multiple translations to reposition a series of shapes.

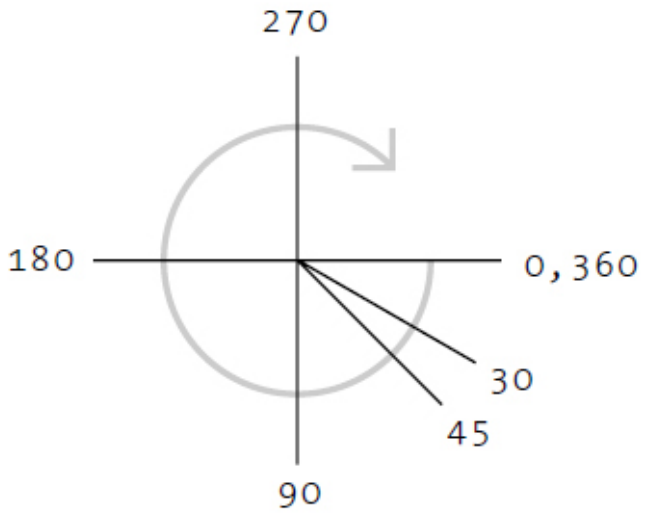
(p.135)

17.2. Use scale with a for structure to scale a shape multiple times.

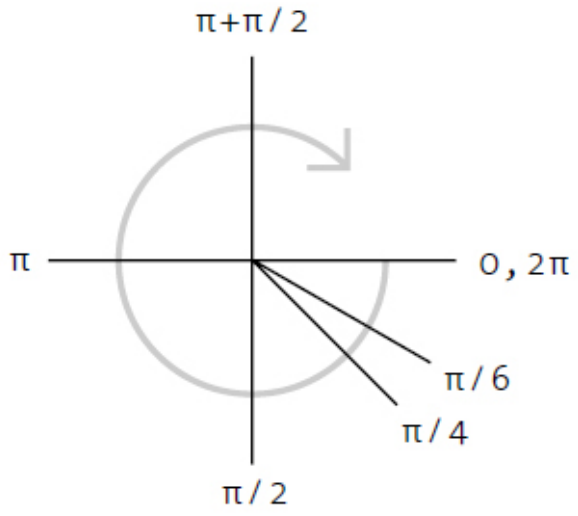
(p. 143)

TRIGONOMETRY (p.117)

If we want to do things with angles in Processing, we need to measure them in radians. These are calculated in relation to pi.



Degree values



Radian values

TRIGONOMETRY (p.117)

The sin() and cos() functions are used to determine the sine and cosine value of any angle. Each of these functions requires one parameter:

sin(angle);
cos(angle);

“angle” needs to be filled with a **radian value**. The most common ones can be expressed as **PI, HALF_PI, or QUARTER_PI**.

sin(HALF_PI);

We can use sin() and cos() to create wave-based compositions (p.121).

```
size(700, 100);
noStroke();
smooth();
fill(0);
float offset = 50.0; // Y offset
float scaleVal = 35.0; // Scale
value for the wave magnitude
float angleInc = PI/28.0; //
Increment between the next angle
float angle = 0.0; // Angle to
receive sine values from
for (int x = 0; x <= width;
x += 5) {
float y = offset + (sin(angle) *
scaleVal);
rect(x, y, 2, 4);
angle += angleInc;
}
```

CIRCLES AND ARCS (p.123-124)

Circles can be created with sin() and cos() using the following code. Change the **radius** to modify the size of the circle. Experiment with the size of the ellipses (at the moment set to 6).

```
noStroke();
smooth();
int radius = 38;
for (int deg = 0; deg < 360; deg += 12) {
float angle = radians(deg);
float x = 50 + (cos(angle) * radius);
float y = 50 + (sin(angle) * radius);
ellipse(x, y, 6, 6);
}
```

CIRCLES AND ARCS (p.123-124)

You can make arcs with a special function:

arc(x, y, width, height, start, stop);

Arcs are drawn along the outer edge of an ellipse defined by the x, y, width, and height parameters. The start and stop parameters specify the angles needed to draw the arc form in units of radians.

e.g. arc(50, 55, 60, 60, HALF_PI, PI);

SPIRALS (p.125)

To create a spiral, all you do is take the code for a circle and make the variable **radius** increase gradually over time (changes from circle code in bold).

```
noStroke();
smooth();
float radius = 1.0;
for (int deg = 0; deg < 360*6; deg += 11) {
  float angle = radians(deg);
  float x = 75 + (cos(angle) * radius);
  float y = 42 + (sin(angle) * radius);
  ellipse(x, y, 6, 6);
  radius += 0.34;
}
```