SDES3607

INTRODUCTIONTOPROCESSING

WEEK 2

MONDAY 9-12PM

D103

LAST WEEK

CODE ELEMENTS	Comments, functions, expressions, statements, console
COORDINATES, PRIMITIVES	Coordinates, primitive shapes, drawing, grey values
VARIABLES	Data types, variables, Processing variables
ARITHMETIC, FUNCTIONS	Arithmetic, operators, grouping, shortcuts, constraining

Don’t forget to use Omnium!

www.online.cofa.unsw.edu.au/2014s1/sdes3607/base/

WEEK 2

TOPICS:

DECISIONS Relational expressions, conditionals, logical operators

REPETITION Iteration, nested iteration, formatting code blocks

VERTICES Vertex, points, lines, shapes, curves

COLOUR BY NUMBERS Setting colours, colour data, RGB, HSB, hexadecimal

DISPLAY, TINT Display, image colour, transparency

CURVES Exponents, roots, normalising, mapping, simple curves

TEXT Characters, strings

SHAPES: VERTICES (p.69)

We can use a series of functions to move beyond primitive shapes and make more complex forms.

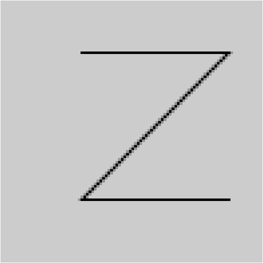
To do this, we use the following structure:

```
beginShape();
vertex(x,y);
endShape();
```

We can add however many vertices we want to make a shape.

e.g.

```
noFill();
smooth();
beginShape();
vertex(30,20);
vertex(85,20);
vertex(30,75);
vertex(85,75);
endShape();
```



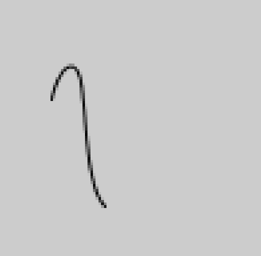
CURVE VERTICES (p.74-77)

Vertices are great for straight lines, but what about curves?

Instead of vertex(x,y); we can use curveVertex(x,y).

The first and last vertex points act as ‘control points’ similar to the bezierCurve() function.

e.g.



```
smooth();
noFill();
beginShape();
curveVertex(20,80);
curveVertex(20,40);
curveVertex(30,30);
curveVertex(40,80);
curveVertex(80,80);
endShape();
```

SHAPE: PARAMETERS (p.71-73)

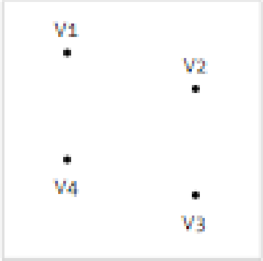
Begin shape can take different parameters to change the drawing method of vertex data.

These parameters are placed inside the beginShape() function.

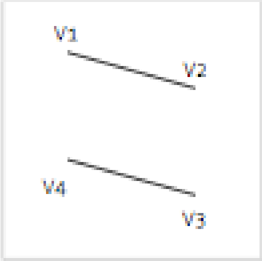
```
beginShape(TRIANGLES);
```

Custom shapes are left ‘open’ by default but can be ‘closed’ by using the CLOSE parameter in the endShape() function.

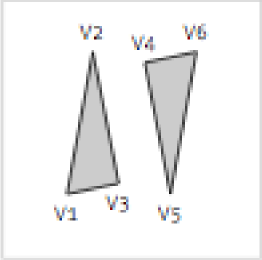
```
endShape(CLOSE);
```



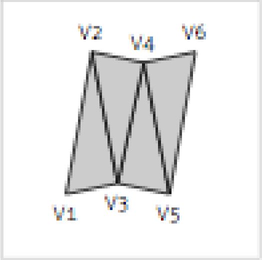
POINTS



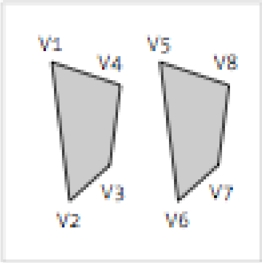
LINES



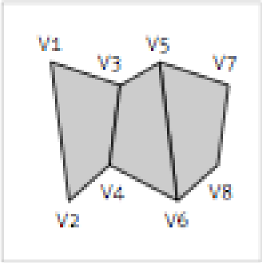
TRIANGLES



TRIANGLE_STRIP



QUADS



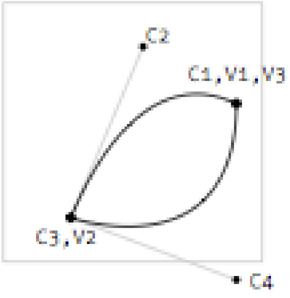
QUAD_STRIP

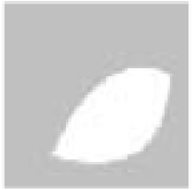
BEZIER VERTICES (p.75-77)

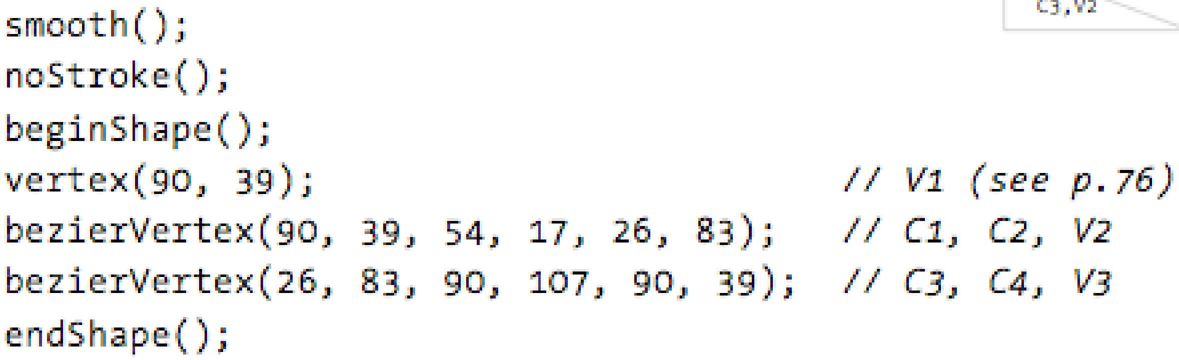
We can use bezier vartices to have more control over the movement of our curves.

Instead of curveVertex(x,y) we can use bezierVertex(cx1,cy1,cx2, cy2,x,y)

Now we can mix and match vertex points that do not have controls with bezierVertex points to control different segments of our lines.







```
smooth();
noStroke();
beginShape();
vertex(90, 39);                          // V1 (see p.76)
bezierVertex(90, 39, 54, 17, 26, 83);    // C1, C2, V2
bezierVertex(26, 83, 90, 107, 90, 39);   // C3, C4, V3
endShape();
```

EXERCISE

7.1. Use beginShape() to draw a shape of your own design.

7.3. Draw a complex curved shape of your own design using bezierVertex()

(p.77)

COLOUR (p.85-93)

So far we have only been using grey values to set fills and strokes. To set **colours** of elements in Processing, we usually use the **RGB** model, which is standard for screen-based media (red, green, blue). The **HSB** (hue, saturation, brightness) model can also be used.

```
background(242, 204, 47);
```

```
background(174, 221, 60);
```

To set a colour, we use up to four parameters:

```
fill(value1, value2,
value3, transparency);
```

EXERCISE

9.1. Explore a wide range of colour combinations within one composition.

(p.93)

IMAGES (p.96)

We can load and display images in Processing.

To put an image in our sketch, we must **create an image variable**, **load** it, and then **call** it:

```
PImage img;
img = loadImage(“image.jpg”);
image(img,0,0);
```

↑

coordinates for position of image

The image must be in a folder called “data” in your sketch folder.

TINTING (p.74-77)

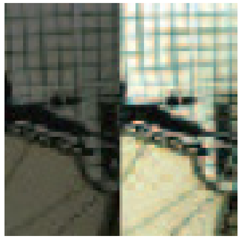
Once you've loaded and displayed an image, you can tint it with a particular grey value or a colour.

```
tint(grey value);
tint(value1, value2, value3,
alpha);
```

Set the tint before you call the image with image(img, 0,0);

noTint() disables the tint.

e.g.



```
PImage img;
img = loadImage
("arch.jpg");
tint(102);
image(img, 0, 0);
noTint();
image(img, 50, 0);
```

EXERCISE

10.1. Draw two images in the display window.

(p.99)

EXPONENTS, ROOTS (p.79-81)

Processing uses several functions that can perform various mathematical tasks:

sq(value) = a number squared (e.g. x^2)

sqrt(value) = square root of a number

pow(num,exponent) = num raised to an exponent

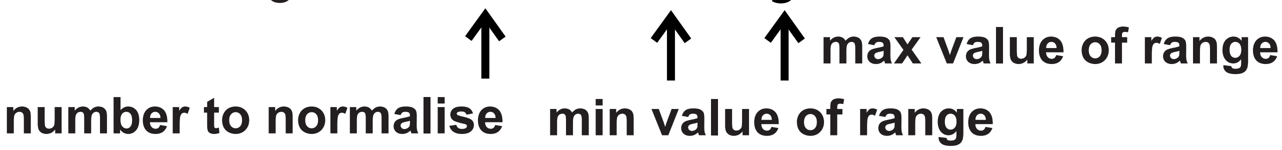
↑ ↑

number to multiply **how many times to multiply num by itself**

NORMALISING, MAPPING (p.80-81)

Sometimes we need to convert numbers to a smaller range: 0.0 - 1.0.

This is called normalising: norm(value, low, high)



number to normalise

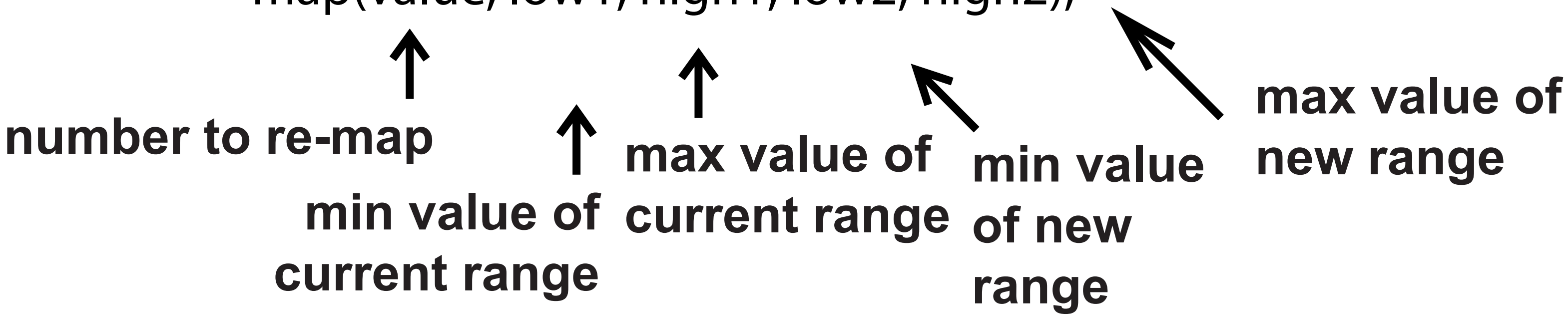
min value of range

max value of range

e.g. float x = norm(102.0,0.0,255.0);

Numbers can also be mapped directly to new ranges:

map(value, low1, high1, low2, high2);



number to re-map

min value of current range

max value of current range

min value of new range

max value of new range

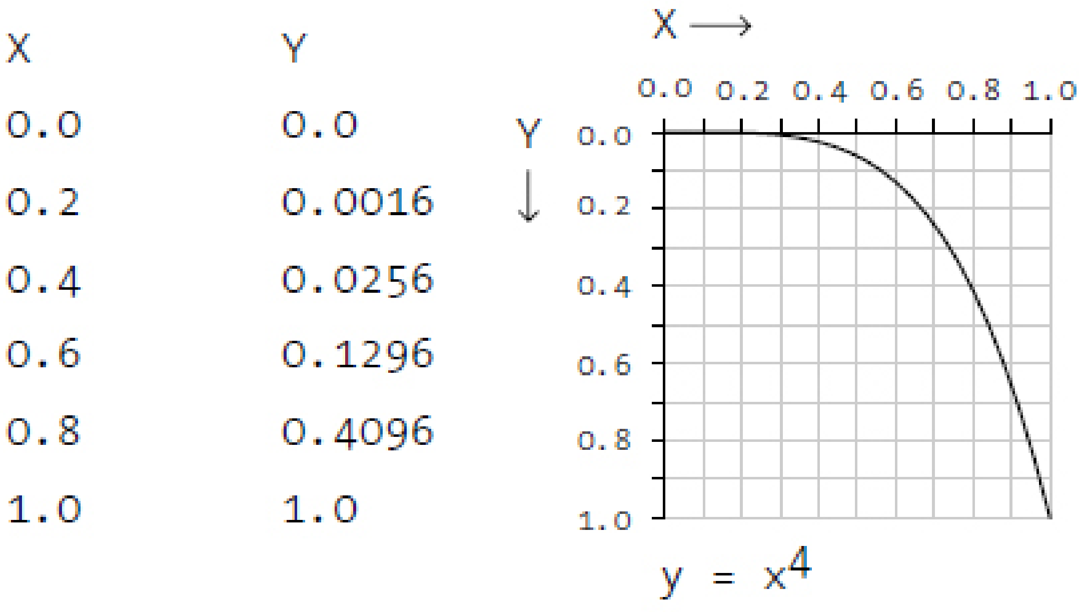
CURVES (p.83-84)

We can use exponential functions to create simple curves.

These come in the form: **$y = x^n$**

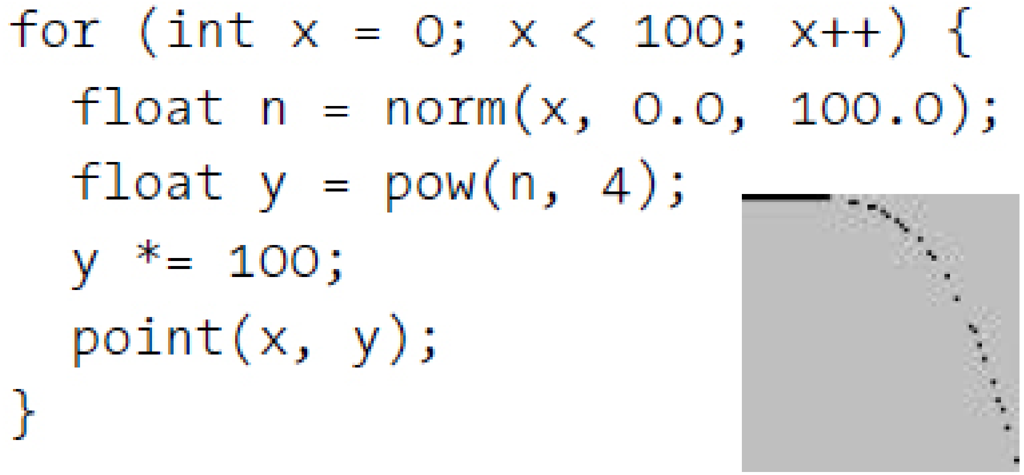
As x gets multiplied by itself over and over, y increases exponentially, creating a curve.

We use a for structure to cycle through numbers, normalise the current number, create the equation, multiply it by the maximum value of the range, then draw a point at the co-ordinates produced. This makes a curve.



X	Y
0.0	0.0
0.2	0.0016
0.4	0.0256
0.6	0.1296
0.8	0.4096
1.0	1.0

$y = x^4$



```
for (int x = 0; x < 100; x++) {
  float n = norm(x, 0.0, 100.0);
  float y = pow(n, 4);
  y *= 100;
  point(x, y);
}
```

EXERCISE

8.1. Draw the curve y = 1 - x(to the power of 4)

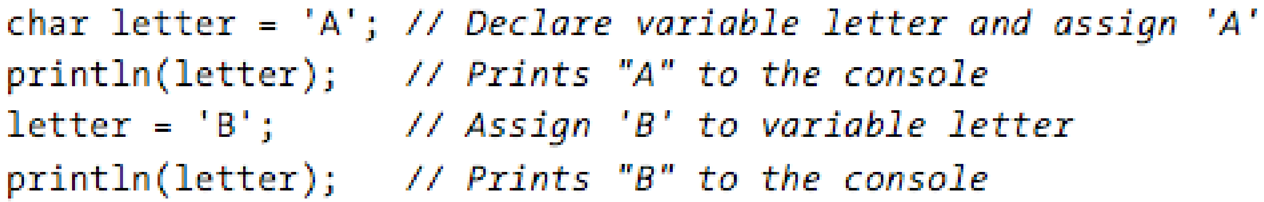
8.2. Use the data from the curve
y = x (to the power of 8) to draw something unique.

(p.84)

CHARACTERS (p.102-103)

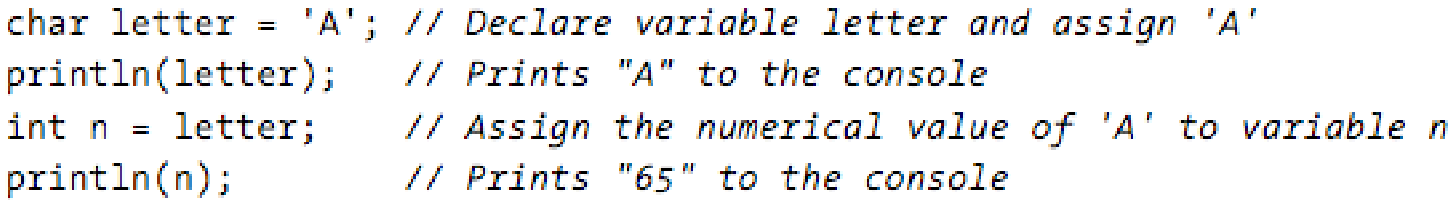
The char data type stores characters - single letters.

To assign a character to a char variable you need to use single quotation marks.



```
char letter = 'A'; // Declare variable letter and assign 'A'
println(letter);   // Prints "A" to the console
letter = 'B';      // Assign 'B' to variable letter
println(letter);   // Prints "B" to the console
```

Characters often have corresponding numbers that are used as numerical references. These numbers are called ASCII codes.

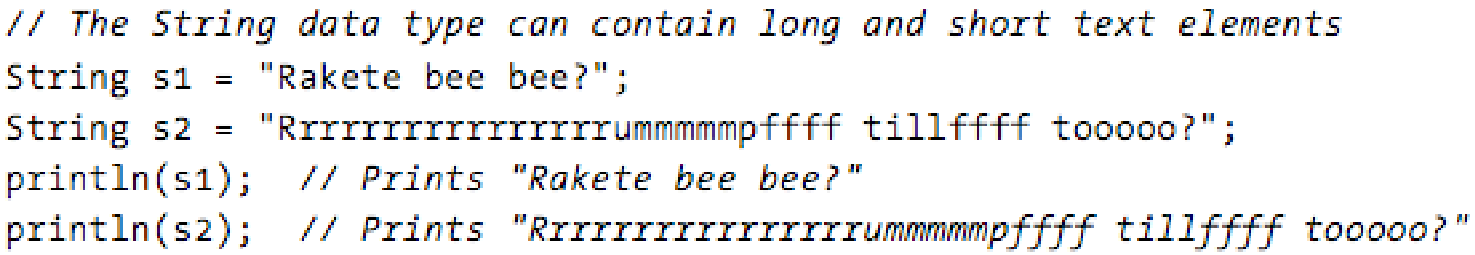


```
char letter = 'A'; // Declare variable letter and assign 'A'
println(letter);   // Prints "A" to the console
int n = letter;    // Assign the numerical value of 'A' to variable n
println(n);        // Prints "65" to the console
```

STRINGS (p.103-104)

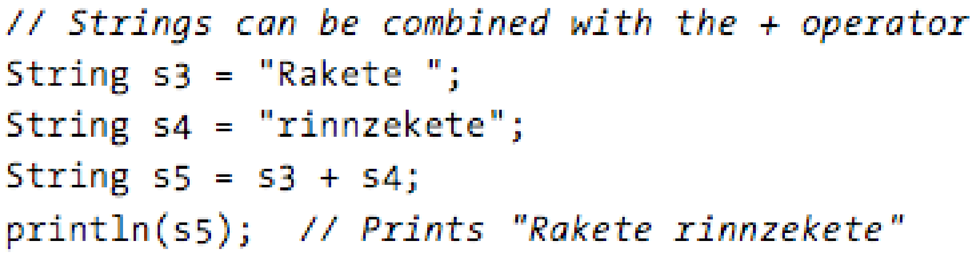
The String data type stores words and sentences.

To assign a word or sentence to a String variable you need to use double quotation marks.



```
// The String data type can contain long and short text elements
String s1 = "Rakete bee bee?";
String s2 = "Rrrrrrrrrrrrrrrrummmmmpffff tillffff tooooo?";
println(s1);  // Prints "Rakete bee bee?"
println(s2);  // Prints "Rrrrrrrrrrrrrrrrummmmmpffff tillffff tooooo?"
```

You can join strings together by using the + operator.



```
// Strings can be combined with the + operator
String s3 = "Rakete ";
String s4 = "rinnzekete";
String s5 = s3 + s4;
println(s5);  // Prints "Rakete rinnzekete"
```

EXERCISE

11.1. Create five char variables and assign a character to each. Write each to the console.

11.2. Create two Sting variables and assign a word to each. Write each to the console.

(p.104)

FOR NEXT WEEK

EXERCISES: 13.1, 13.2, 13.3, 14.1, 14.2, 15.1, 16.1, 16.2, 16.3, 17.1

READINGS: This week's topics

Next week's topics: have a flick through TYPOGRAPHY 1, MATHS 3, MATHS 4, TRANSFORM 1, TRANSFORM 2, INTERVIEWS 2*

Don't forget to upload your exercises to the classwork server.