Introduction to Creative Coding with Processing

Week 3 - Animation, Interaction and Basic OOP

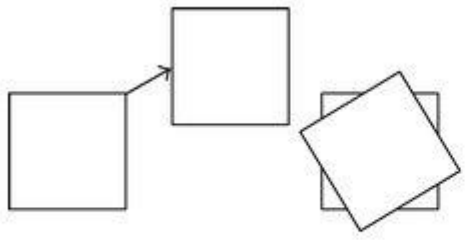
Thomas Deacon, 2019

Resources for each week available at:
https://github.com/VizRCA/intro-to-creative-coding

Topics

Week 3

- Recap week 2
- Object oriented programming (OOP)
- Continuous function evaluation
- Mapping across time
- Capturing and using inputs



Recap wk 2

translate(x,y) and rotate(ø)

intro-to-creative-coding/w01/
translateRotate/
translateRotate.pde

```
size (300, 300);
background (255);

// draw a rectangle at 0, 0: no translation
fill (200);
rect (0, 0, 60, 60);

// translate the co-ordinate grid and redraw the rectangle
(under translation)
pushMatrix ( );
translate (80, 80);
fill (140);
rect (0, 0, 60, 60);
popMatrix( );

// push/pop Matrix re-set the co-ordinate grid
// now perform another translation and rotation
pushMatrix( );
translate (160, 160); // translate co-ordinates
rotate (radians (45)); // rotate co-ordinate grid, convert
degrees to radians.
fill (100);
rect (0, 0, 60, 60);
popMatrix ( );
```

Recap wk 2

random()

intro-to-creative-coding/wk2
randomBars/randomBars.pde

```
/**
 * Random.
 * Random numbers create the basis of this
image.
 * Each time the program is loaded the result is
different.
 */

size(640, 360);
background(255);
strokeWeight(8);

for (int i = 6; i <= width; i+=16) {
  float r = random(255);
  stroke(r);
  line(i, 0, i, height);
}
```

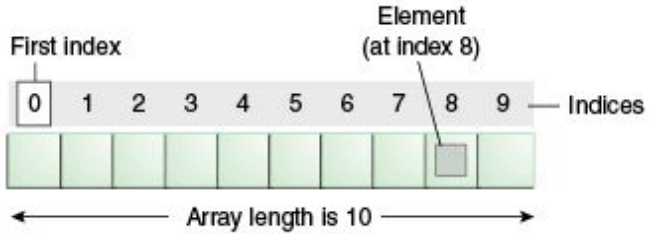
Recap wk 2

beginShape, endShape, vertex, curves

intro-to-creative-coding/wk2 simpleShapes/ simpleShapes.pde

```
/** Custom shapes beyond rect and ellipse etc
 *  beginShape, vertex and endShape allow creation of new geometries
 *  here, z like shapes are made
 */

size(600, 200);
background(255);
noFill();
// Order of vertex position changes the way the shape is drawn
beginShape();
vertex(30, 20);
vertex(width/2-30, 20);
vertex(30, height-20);
vertex(width/2-30, height-20);
endShape();
...
```



Tools

Arrays

intro-to-creative-coding/g_arrays/g_arrays.pde

```
// A list with 4 spaces.
String [] shoppingList = new String [4];

// Each of the 4 spaces in the shoppingList array are populated with information
shoppingList[0] = "apples";
shoppingList[1] = "milk";
shoppingList[2] = "eggs";
shoppingList[3] = "bread";

// items in an array can be found and output:
println (shoppingList[2]); // outputs 'eggs'

// the size (length) of an array can be found by using 'array.length'
println ("number of items in shopping list =  " + shoppingList.length);

// for loops are useful ways to cycle through an array and get each item in turn:
for (int i=0; i<shoppingList.length; i++) {
  println (shoppingList[i]);
}
```

Example

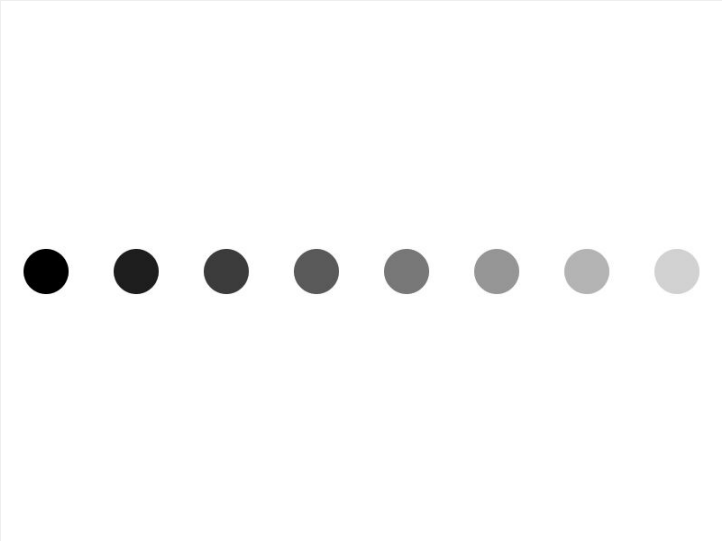
Arrays, Sin, Cos & Tan

intro-to-creative-coding/wk3/
arrayMappingWaves/
arrayMappingWaves.pde

Challenge

Arrays

intro-to-creative-coding/wk3/
array_challenge/
array_challenge.pde



Concept

Continuous Evaluation

Programs that animate or respond to input must run continuously.

In processing this is the draw() function.

You can only have one draw().

Frames (based on code inside draw) run at 60FPS by default, but you can change this.

Tools

draw, frameRate, frameCount

draw() - function to put looping code

frameRate(x) - set the draw function frame rate using integer x

frameCount - returns the current frame count since start of program

Tools

draw, frameRate, frameCount

```
void draw()
{
    frameRate(4);
    println(frameCount);
}
```

Examples

Continuous Evaluation

intro-to-creative-coding/wk3/

- continuousEvaluation_a.pde
- continuousEvaluation_b.pde

Challenge

Continuous Evaluation

intro-to-creative-coding/wk3/continuousEvaluation_challenge/continuousEvaluation_challenge.pde

Challenge

Continuous Evaluation

intro-to-creative-coding/wk3/
animate_challenge/
animate_challenge.pde

Tools

Flow control and wrapping

If things run forever we need to create rules so that things stay on the screen or behave in controllable ways.

- bounce_a.pde
- bounce_b.pde
- bounce_c.pde

Challenge

Bounce around all sides

intro-to-creative-coding/wk3/
bounce_challenge/
bounce_challenge.pde

Concept

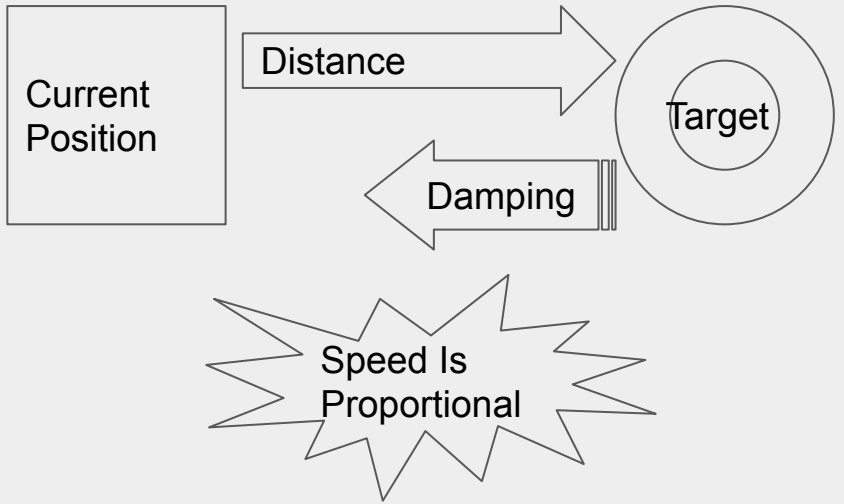
Forces

- Inertia and Damping
- Gravity
- Bouncing
- Wind
- Springs

Example

Inertia and Damping

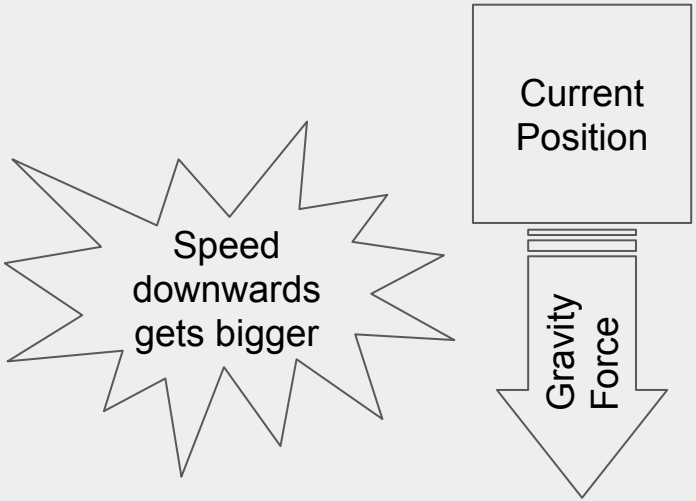
intro-to-creative-coding/wk3/
inertia_example/
inertia_example.pde



Example

Gravity

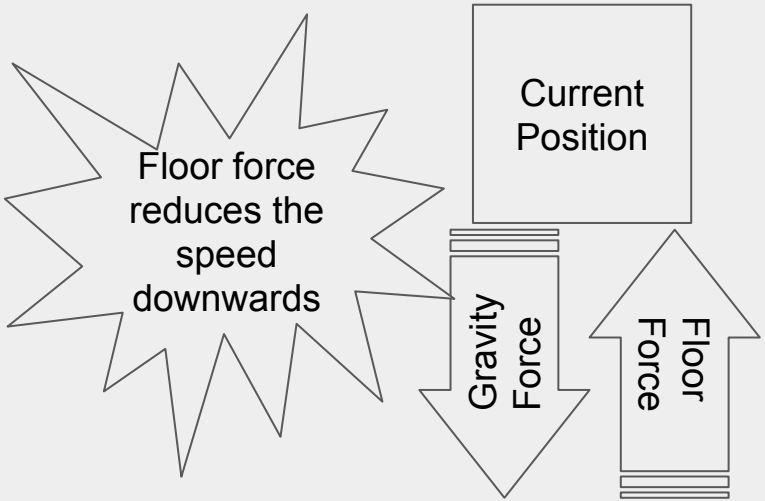
intro-to-creative-coding/wk3/
gravity_example/
gravity_example.pde



Challenge

Gravity Bounce

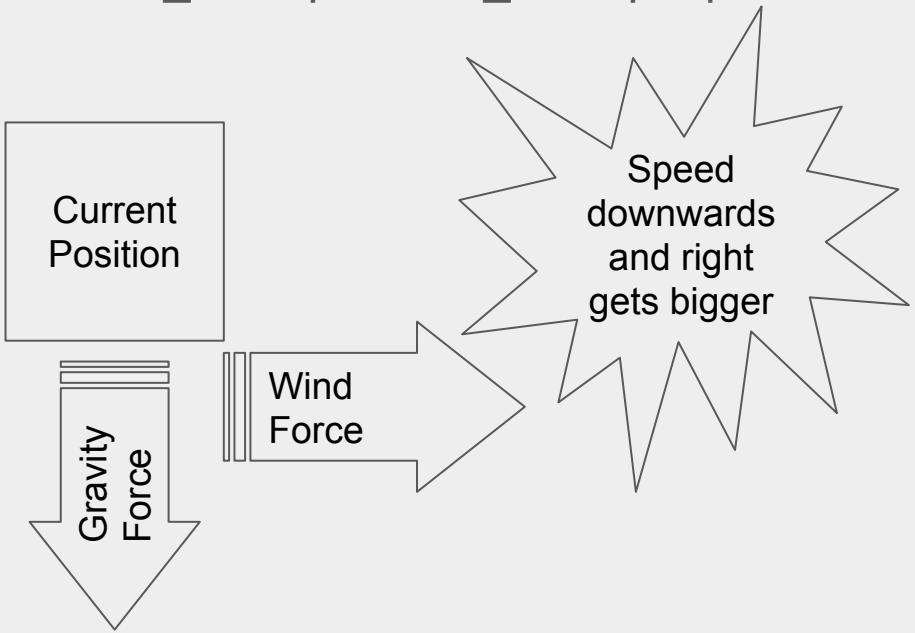
intro-to-creative-coding/wk3/
gravity_challenge/
gravity_challenge.pde



Example

Wind

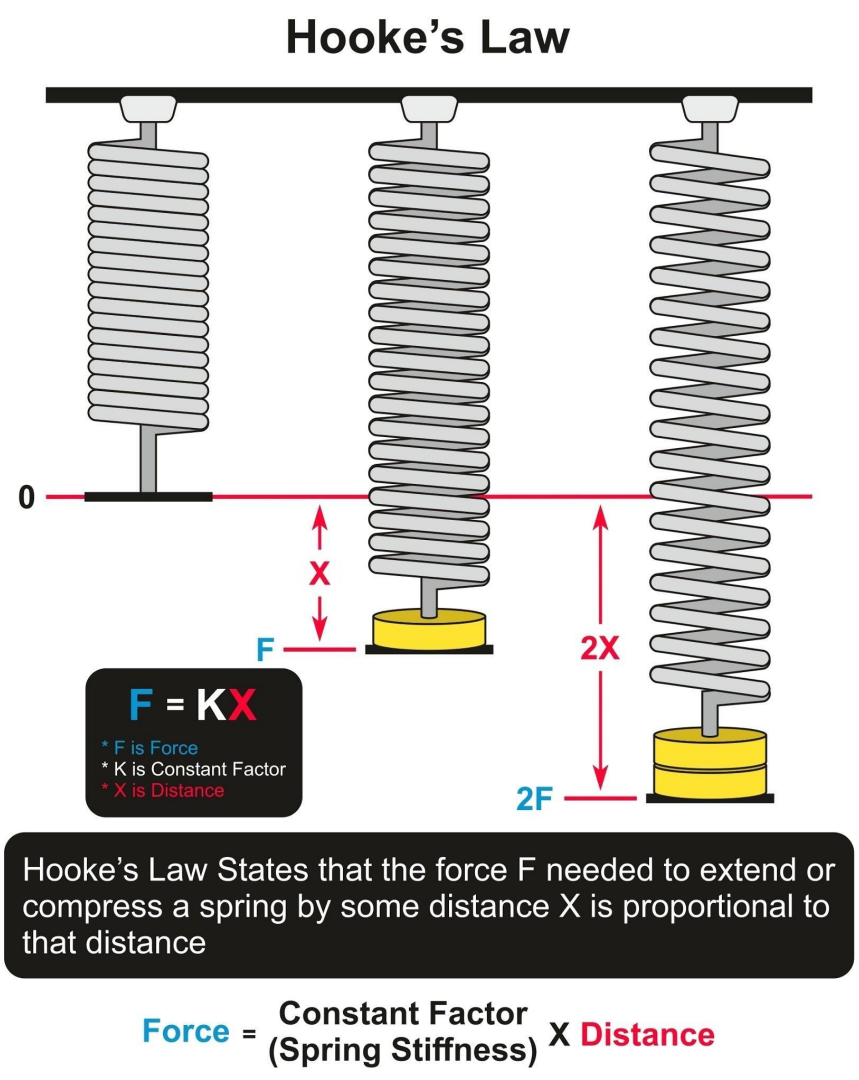
intro-to-creative-coding/wk3/
wind_example/wind_example.pde

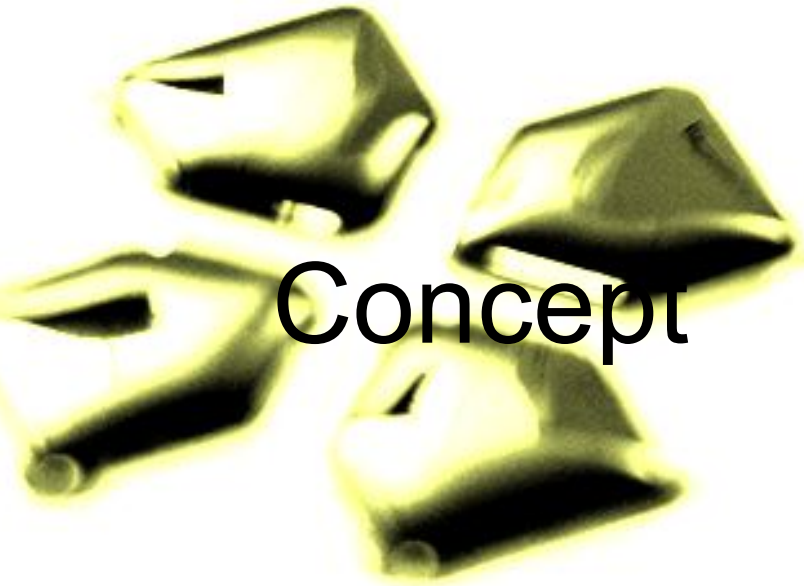


Example

Springs

intro-to-creative-coding/wk3/
spring_example/
spring_example.pde





Concept

User Input

As programmes can run continuously, we can check if things change on input devices during each iteration of the draw loop.

In processing this is typically mouse movement, mouse button presses and keystrokes. But it can be any input device such as arduino, or another app via OSC.

Tools

mouseX, mouseY

Open **mouse_example.pde**

mouseX - current frames mouse position in x direction

mouseY - current frames mouse position in y direction

pmouseX - as above but previous frame

pmouseY - as above but previous frame

Challenge

Mouse Follow

Convert the spring forces sketch to follow the mouse in both directions (X,Y)

Open
mouseFollow_challenge.pde

Challenge

Store mouse positions using arrays

Open **storeInputs_challenge.pde**

Need to use the modulo (%) operator in this example. Modulo operation finds the remainder after division of one number by another (called the modulus of the operation)

24 % 2 = 0 < 2 is a whole divisor

24 % 5 = 4 < 5 leaves 4 behind

Modulo is very useful in coding for counting related to frame rates

Example

Mouse Functions

mouseButton, mouseClicked(), mouseDragged(), mouseMoved(), mousePressed(), mousePressed, mouseReleased(), mouseWheel()

mouseFunctions_example.pde

Checks for “hit” of mouse over object, this is important feature.

Uses functions dedicated to mouse events to change data in program.

Resources

Keyboard Input

Check in your own time:

- keyboard_example
- keyboardFunctions_example

Concept

Abstraction

Collecting together similar instructions or data structures to improve the design of programs.

It helps to reduce programming complexity and effort.

It can be done at a variety of levels: functions, classes and architecture.

Example

Mapping mouse interaction

Open **ribbonShapes.pde**

Abstraction is used to collect draw instructions that just need a single input.

Global variable scope allows this to work.

Concept

Object Oriented Programming

OOP is a massive topic, broadly it is the process of making abstractions about groups of procedural commands that can be reused. This includes grouping bits of functionality into a class with variables and functions.

Concept

Classes

A Class is a template from which lots of similar objects can be created.

Open **simpleOOP_example.pde**

Examples

Classes, Objects, Composite Classes

Objects are instances of classes.

Classes can be made up of other classes too.

Open **oop_example_a.pde**

Abstraction like this allows cleaner code and creation of lots of the same things.

Open **oop_example_b.pde**

Once the principles of OOP are understood you can achieve quite a large level of complexity in a program.

Open Open **oop_example_c.pde**

Tools

PVector

Convert mouse follow forces sketches to use PVectors

Lissajous sketch combines keystrokes, pvector arrays, abstraction