SDES3107
DESIGN AND COMPUTERS 4

INTRODUCTION TO PROCESSING

WEEK 4
MONDAY
D103

WEEK 4

TOPICS COVERED:

CONTINUOUS Continuous evaluation, controlling the flow, variable scope

MOUSE INPUT Getting input from the mouse to control the behaviour of the sketch

DRAWING Drawing simple pictures using mouse input and shapes

FUNCTIONS Making your own function!

KEYBOARD INPUT Reading input from the keyboard

INPUT EVENTS Triggering your own code when an an input happens

STRUCTURING (p.173-176)

So far we have been making programs that only run once. These are good for static compositions. For dynamic (interactive/animation-based) compositions, we need to use a slightly different structure:

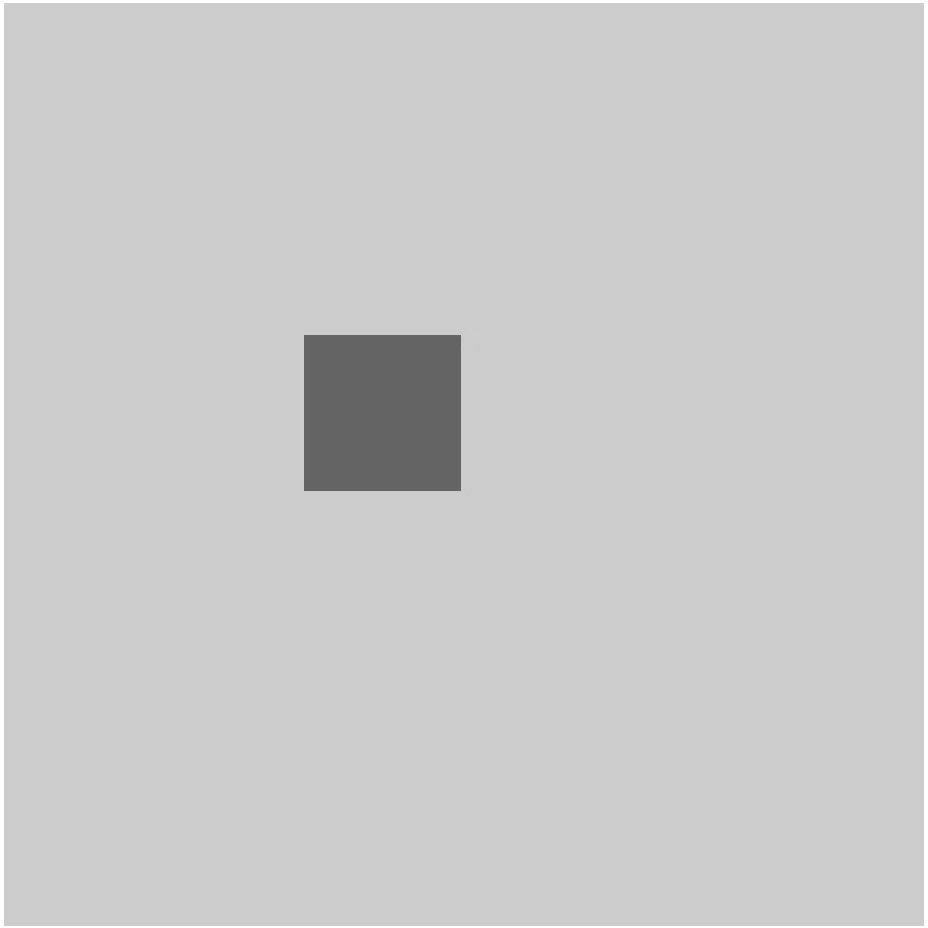
```
size(300,300);
int randomPos =
int(random(300.0));
fill(100);
noStroke();
rect(randomPos,randomPos+
10,50,50);
```

↑
only runs once

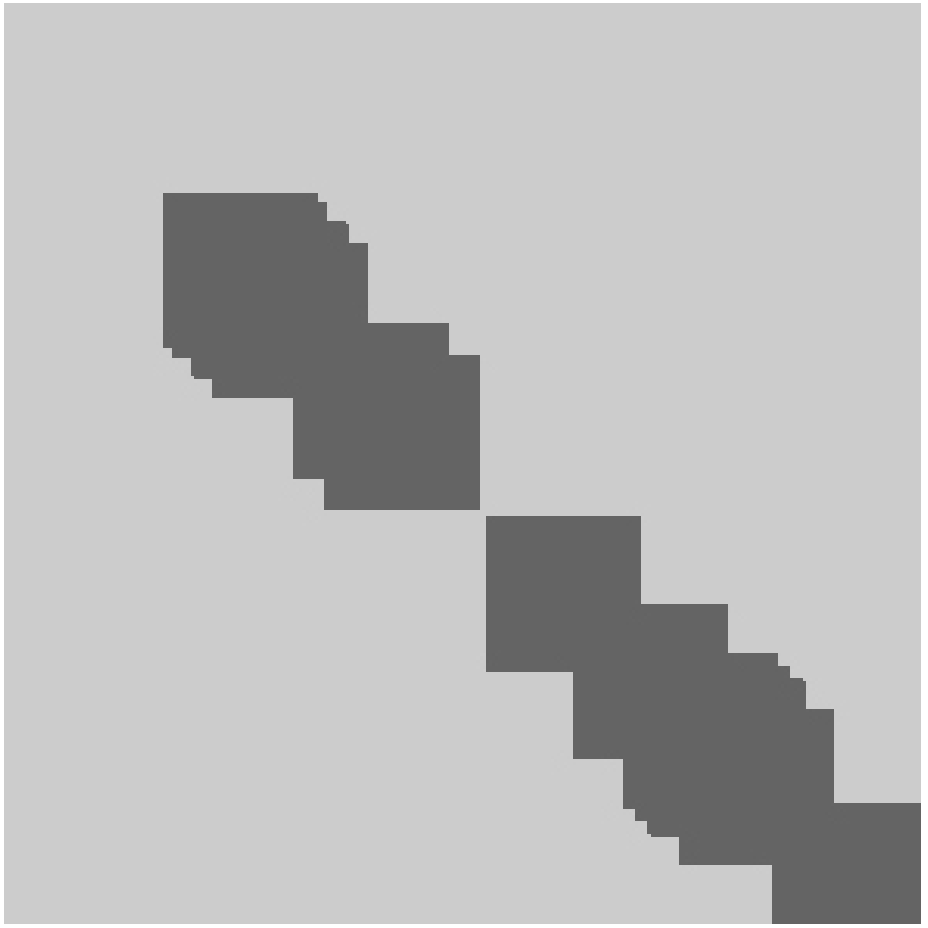
```
void setup() {
size(300,300);
fill(100);
noStroke();
}
void draw() {
int randomPos =
int(random(300.0));
rect(randomPos,randomPos+
10,50,50);
}
```

↑
runs over and over

program that only runs once



program that runs over and over



STRUCTURING (p.173-176)

```
int winSize = 300;
```

Most variables should be declared at the top (so they can be accessed anywhere).

```
void setup() {
size(winSize,winSize);
fill(100);
noStroke();
}
```

Think about your code. Anything that **only needs to be set once** should go in **void setup()**.

```
void draw() {
int randomPos =
int(random(300.0));
rect(randomPos,randomPos
+10,50,50);
}
```

Anything that we want to be **drawn over and over**, or **changed in any way as the program runs**, should go in **void draw()**.

STRUCTURING (p.173-176)

There are a few things you can do to control how this works:

Putting **`noLoop()`** in the **`void draw()`** section stops the program from running over and over.

To stop shapes from building up, put **`background(255)`** (or whatever your background is) to “refresh” the background. This gives the illusion of motion!

You can slow down the speed of the program by using **`frameRate()`**.
e.g. **`frameRate(10)`**.

```
int winSize = 300;

void setup() {
size(winSize, winSize);
fill(100);
noStroke();
frameRate(10);
}

void draw() {
int randomPos =
int(random(300.0));
background(255);
rect(randomPos,randomPos
+10,50,50);
}
```

MOUSE INPUT (p.205-213)

We can add interactivity to our programs by using the mouse input variables: **mouseX** and **mouseY**. They store the **horizontal** and **vertical positions** of the **current mouse position**.

```
void setup() {
size(100,100);
smooth();
noStroke();
}

void draw() {
background(126);
ellipse(mouseX,mouseY,
33,33);
}
```

"refreshes" the screen → `background(126);`

uses the position of the mouse to set the position of an ellipse → `ellipse(mouseX,mouseY, 33,33);`

EXERCISE

23.1. Control the position of a shape with the mouse. Strive to create a more interesting relation than one directly mimicking the position of the cursor.

(p.215)

MOUSE INPUT (p.212-213)

We can also make things happen when a **mouse button is pressed**.

To do this we can use an **if structure to test if the mouse has been pressed**:

```
if (mousePressed == true) {
   //something happens here
} else {
   //something else happens
}
```

Example:

```
void setup() {
size(100,100);
smooth();
noStroke();
}

void draw() {
background(204);
if (mousePressed == true) {
   fill(255);
} else {
   fill(0);
}
rect(25,25,50,50);
}
```

DRAWING (p.218-221)

Using the mouse inputs **mouseX**, **mouseY**, and **mousePressed**, we can create our own drawing programs.

We can draw with points, shapes, images, etc...

```
void setup() {
size(100,100);
}

void draw() {
point(mouseX, mouseY);
}
```

Draw only when the mouse is pressed:

```
void setup() {
size(100,100);
stroke(255);
}

void draw() {
if(mousePressed == true){
   line(mouseX, mouseY,
   pmouseX, pmouseY);
   }
}
```

EXERCISE

24.3. Load an image and use it as a drawing tool.

(this is described on p.221)

MAKING YOUR OWN FUNCTIONS (p.181-196)

Functions (like **ellipse(x,y,width,height)** and **fill(colour value)**) are made to complete a specific task. They have very specific purposes.

Parameters allow the function to change depending on what that parameter is. E.g. the fill changes depending on what colour we put in as a parameter.

Processing allows us to make our own functions.

This is extremely useful when we want to do or make something many times.

MAKING YOUR OWN FUNCTIONS (p.181-196)

To make a function, you need to:

- Have a **clear idea of what you want it to do** (e.g. make a diamond)

- Decide what parameters you want (**how will the function be controlled?**)

- Break what you want to do into small steps (sizing, positioning, fill, etc)

We use the code on the right to create our own function. This goes *below* void draw().

```
void setup() {
size(100,100);
smooth();
noLoop();
}

void draw() {
turnRed();
rect(25,25,50,50);
}

void turnRed() {
  fill(252,0,0);
}
```

use your function here

write your function here

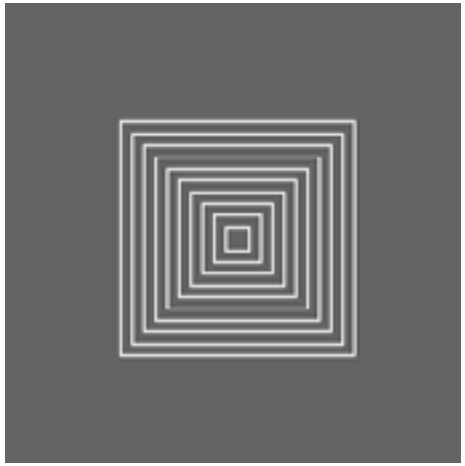
SOLUTION

```
/*Write a function to
draw a shape to the
screen multiple times,
each at a different
position.*/

void setup() {
  size(200,200);
  background(100);
  smooth();
  noLoop();
  stroke(255);
  noFill();
  rectMode(CENTER);
}
```

```
void draw() {
squareNest(100,100,100,100);
}

void weirdSquare(int size,
int number, int x, int y) {
  for(int counter = number;
counter > 0; counter-=10) {
    rect(x,y,size,size);
    size-=10;
  }
}
```



EXERCISES

21.1. Write a function to draw a shape to the screen multiple times, each at a different position.

21.2. Extend the function created for exercise 1 by creating more parameters to control additional aspects of its form.

(p196)

KEYBOARD INPUT (p.223-228)

We can also make things happen when a **keyboard key is pressed**.

Checking if **`keyPressed`** variable is true will tell you if a key has been pressed since last frame.

Checking **`key`** will tell you what key has been last pressed, and ALT, CONTROL, SHIFT, UP, DOWN, LEFT, and RIGHT keys can be checked for using **`keyCode`** (see p227).

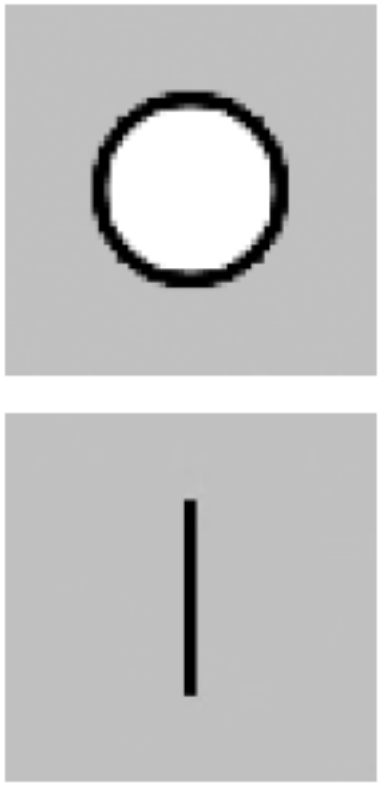
You can find the values for all keys in the ASCII table on p665.

```
PFont myFont;

void setup() {
myFont = createFont("Arial");
  textFont(myFont, 32);
}

void draw() {
background(0);
text(key, 28, 75);
}
```

KEYBOARD INPUT (p.223-228)



```
void setup() {
   size(100, 100);
   smooth();
   strokeWeight(4);
}
void draw() {
   background(204);
   // If the ‘A’ key is pressed draw a line
   if ((keyPressed == true) && (key == ‘A’)) {
      line(50, 25, 50, 75);
   } else { // Otherwise, draw an ellipse
      ellipse(50, 50, 50, 50);
   }
}
```

EXERCISE

25.1. Use the number keys on the keyboard to modify the movement of a line.

(p227)

INPUT EVENTS (p.229 - 236)

Input events are specially named functions **which are triggered when their input condition happens**, i.e:

```
mousePressed()
mouseReleased()
mouseMoved()
mouseDragged()
keyPressed()
keyReleased()
```

The code inside the function `mousePressed()` is called whenever the mouse button is pressed.

```
float gray = 0;

void setup() {
   size(100, 100);
}

void draw() {
   background(gray);
}

void mousePressed() {
   gray += 20;
}
```

Note: `mousePressed()` **is a function, but** `mousePressed` **is a variable!**

EXERCISES

26.1. Animate a shape to react when the mouse is pressed and when it is released.

26.3. Write a program to update the display window only when a key is pressed.

(p236)

HOMEWORK

EXERCISES: 20.1, 20.2, 21.1, 21.2, 22.1, 22.3, 23.1, 23.2, 24.3, 25.1, 25.2, 26.1, 26.3

READINGS: STRUCTURE 2, STRUCTURE 3, SHAPE 3, INPUT 1, DRAWING 1, INPUT 2, INPUT 3

Next week's topics: DATA 4, IMAGE 2, IMAGE 3, TYPOGRAPHY 2, TYPOGRAPHY 3