

Functions

Creative Coding & Generative Art in Processing 2
Ira Greenberg, Dianna Xu, Deepak Kumar, Sonu Khullar

Review: Drawing Basics

Canvas

- `size(width, height)`
- **Drawing Tools**
- `point(x, y)`
- `line(x1, y1, x2, y2)`
- `triangle(x1, y1, x2, y2, x3, y3)`
- `quad(x1, y1, x2, y2, x3, y3, x4, y4)`
- `rect(x, y, width, height)`
- `ellipse(x, y, width, height)`
- `arc(x, y, width, height, startAngle, endAngle)`
- `curve(cpx1, cpy1, x1, y1, x2, y2, cpx2, cpy2)`
- `beginShape()`
- `endShape(CLOSE)`
- `vertex(x, y)`
- `curveVertex(x, y)`
- **Colors**
- `grayscale [0.255], RGB [0.255][0.255][0.255], alpha [0.255]`
- `background(color)`
- **Drawing & Shape Attributes**
- `smooth()`, `noSmooth()`
- `stroke(color)`, `noStroke()`, `strokeWeight(pixelWidth)`
- `fill(color)`, `noFill()`



GJK2013

2

Variables: Naming Values

Values

- 42, 3.14159, 2013, "Hi, my name is Joe!", true, false, etc.
- **Numbers**
- **Integers**
- `int meaningOfLife = 42;`
- `int year = 2013;`
- **Floating point numbers**
- `float pi = 3.14159;`
- **Strings**
- `String greeting = "Hi, my name is Joe!";`
- **Boolean**
- `boolean keyPressed = true;`

GJK2013

3

Processing: Predefined Variables

width, height

The width & height of the canvas used in the sketch

-
- **PI, HALF_PI, TWO_PI**
- For different values of π . Note that
-
- `HALF_PI = PI/2`
- `TWO_PI = 2*PI`
-
- **displayWidth, displayHeight**
- The width and height of the monitor being used. This is useful in running fullscreen

GJK2013

4

Simple Program Structure

```
// Create and set canvas
size(width, height);
smooth();
background(color);

// Draw something
...
// Draw something else
...
// etc.
```

GJK2013

5

Simple Program Structure

```
// Draw a simple house
// Create and set canvas

size(300, 300);
smooth();
background(187, 193, 127);

// wall
fill(206, 224, 14);
rect(50, 150, 200, 100);

// Draw Door
fill(72, 26, 2);
rect(125, 200, 50, 50);
```



GJK2013

6

Program Structure: Dynamic Mode

Most Processing programs we will write will have the following structure:

<Declare variables>

```
void setup() {
    <initial canvas set up goes here>
} // setup()
```

Program Structure: Dynamic Mode

Most Processing programs we will write will have the following structure:

```
// Draw a simple house
void setup() {
    // Create and set canvas
    size(300, 300);
    smooth();
    background(187, 193, 127);
} // setup()
```



Processing: Dynamic Sketches

```
// Draw a simple house
void setup() {
    // Create and set canvas
    size(300, 300);
    smooth();
    background(187, 193, 127);
} // setup()
```

Code Block:
{
...
...
}

```
void draw() {
    // wall
    fill(206, 224, 14);
    rect(50, 150, 200, 100);
}
```

Processing: Dynamic Sketches

```
// Draw a simple house
void setup() {
    // Create and set canvas
    size(300, 300);
    smooth();
    background(187, 193, 127);
} // setup()
```

setup() block:
Commands here are executed once each time a sketch is played.

draw() block:
Commands here are repeated ~60 times/sec.

```
void draw() {
    // wall
    fill(206, 224, 14);
    rect(50, 150, 200, 100);
}
```

Processing: Dynamic Sketches

```
// Draw a simple house
void setup() {
    // Create and set canvas
    size(300, 300);
    smooth();
    background(187, 193, 127);
} // setup()
```

But...
What are these???

For now...
Necessary syntax

More later...

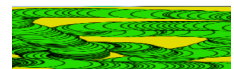
```
void draw() {
    // wall
    fill(206, 224, 14);
    rect(50, 150, 200, 100);
}
```

Something More Interesting...

```
color color1 = color(227, 220, 0);
color color2 = color(37, 220, 0);
color color3 = color(0);
```

```
void setup() {
    // create and set canvas
    size(300, 300);
    smooth();
    background(color1);
} // setup()
```

```
void draw() {
    stroke(color3);
    fill(color2);
    ellipse(mouseX, mouseY, 40, 40);
} // draw()
```



Predefined variables:

pmouseX, pmouseY

```
color color1 = color(227, 220, 0);
color color2 = color(37, 220, 0);
color color3 = color(0);
```

```
void setup() {
  // create and set canvas
  size(300, 300);
  smooth();
  background(color1);
} // setup()
```



GJK2013

13

Events: More Interactivity

```
color color1 = color(227, 220, 0);
color color2 = color(37, 220, 0);
color color3 = color(0);
```

Circles are drawn
ONLY when mouse is pressed.

```
void setup() {
  // create and set canvas
  size(300, 300);
  smooth();
  background(color1);
} // setup()
```



GJK2013

14

Something More Interesting...

```
color color1 = color(227, 220, 0);
color color2 = color(37, 220, 0);
color color3 = color(0);
```

```
void setup() {
  // create and set canvas
  size(300, 300);
  smooth();
  background(color1);
} // setup()
```

```
void draw() {
  stroke(color3);
  fill(color2);
  ellipse(mouseX, mouseY, 40, 40);
} // draw()
```

What happens when...

You move the
background(...) command to draw()?

GJK2013

15

Redo: A Better House Sketch

```
// Draw a simple house
int houseX = 50;
  // bottom left
  corner of house
int houseY = 250;
```

```
int houseHeight = 100;
  // overall
  height of house
```



GJK2013

16

Controlling Frame Rate

frameRate(N);
Changes frame rate to N
times/second

noLoop();
Controls the use of
frame rate.

<Declare variables>

<Declare variables>

```
void setup() {
  ...
  frameRate(30);
} // setup()
```

```
void setup() {
  ...
  noLoop();
} // setup()
```

GJK2013

17

Mathematical Functions

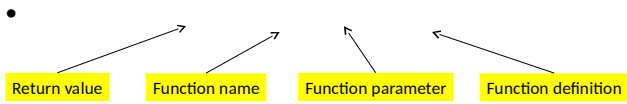
•

$y = \text{sum}$

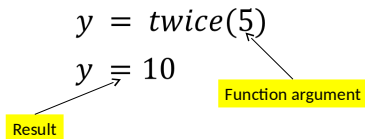
GJK2013

18

Functions: Terminology



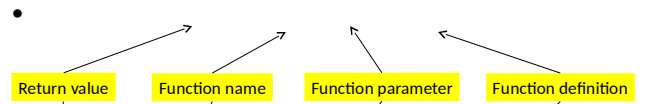
Function application:



GJK2013

19

Processing: Defining Functions



```

float twice(float x) {
    return 2*x;
} // twice()

```

GJK2013

20

Processing: Defining Functions

Syntax:

```

returnType functionName(parameters) {
    ...
    return expression;
}

```

Example:

```

float twice(float x) {
    return 2*x;
} // twice()

```

Use:

GJK2013

21

Defining Functions: void

Use **void** as *returnType* when no value is returned.

Syntax:

```

returnType functionName(parameters) {
    ...
    return expression;
}

```

Example:

```

void square(float x, float y, float
side) {

```

22

Program Structure: Functions

```

color color1 = color(227, 220,
0);
color color2 = color(37, 220, 0);

```

```

void setup() {
    // create and set
size(300, 300);
smooth();
background(color1);
} // setup()

```

GJK2013

23

Variables & Scope

```

color color1 = color(227, 220,
0);
color color2 = color(37,

```

```

void setup() {
    // create and set up ca
size(300, 300);
smooth();
background(color1);
} // setup()

```

Global Variables
Either pre-defined
Or defined at top
Are visible everywhere
In the program

GJK2013

24

Variables & Scope

```
color color1 = color(227, 220, 0);
color color2 = color(37, 220, 0);

void setup() {
  // create and set up canvas
  size(300, 300);
  smooth();
  background(color1);
} // setup()
```

Local Variables

Either parameters Or defined inside blocks

Are visible ONLY in the block After they are defined

GJK2013

25

House() function example

```
// Draw a simple house
```

```
void setup() { // Create and set canvas
  size(300, 300);
  smooth();
  background(187, 193, 127);
} // setup()
```

```
void draw() {
  // Draw a house at 50, 250 in 200x200 pixels
```



26

House() function example

```
// Draw a simple house
```

```
void setup() { // Create and set canvas
  size(300, 300);
  smooth();
  background(187, 193, 127);
} // setup()
```

```
void draw() {
  // Draw a house at 50, 250 in 200x200 pixels
```



27

House() function example

```
// Draw a simple house
```

```
void setup() { // Create and set canvas
  size(300, 300);
  smooth();
  background(187, 193, 127);
} // setup()
```

```
void draw() {
  // Draw a house at 50, 250 in 200x200 pixels
  house(50, 250, 200, 200);
  house(20, 100, 50, 50);
  house(230, 100, 50, 75);
} // draw()
```

```
void house(int houseX, int houseY, int houseWidth, int houseHeight) {
  // Draw a house at <houseX, houseY> (bottom left corner)
  // with width houseWidth and height houseHeight
```

```
...
```

```
} // house()
```



GJK2013

28

Processing: Math Functions

Math functions return values:

Example:

```
void square(float x, float y, float side) {
  rectMode(CORNER);
  rect(x, y, side, side);
} // square()
```

Use:

```
square(50, 50, 100); // draws a 100x100 square at 50, 50
```

Processing has several pre-defined Math functions for calculation, trigonometry, and random number generation

GJK2013

29

Processing: Math Functions

Math functions return values:

Example:

```
void square(float x, float y, float side) {
  rectMode(CORNER);
  rect(x, y, side, side);
} // square()
```

Use:

```
square(50, 50, 100); // draws a 100x100 square at 50, 50
```

Processing has several pre-defined Math functions for calculation, trigonometry, and random number generation

GJK2013

30

Processing: Pre-defined Math Functions

Calculation

- abs(), ceil(), constrain(), dist(), exp(), floor(), lerp(), log(), mag(), map(), max(), min(), norm(), pow(), round(), sq(), sqrt()
- **Trigonometry**
- acos(), asin(), atan(), atan2(), cos(), degrees(), radians(), sin(), tan()
- **Random**
- noise(), noiseDetail(), noiseSeed(), random(), randomGaussian(), randomSeed()

GJK2013

31

Math Functions: Examples

Calculation

```
float x, y;  
y = 42;  
x = sqrt(y);
```

• Trigonometry

```
• float rad = radians(180);  
• float deg = degrees(PI/4);
```

• Random

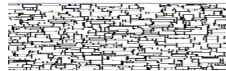
```
• float x = random(10); // returns a random number [0.0..10.0)  
• float y = random(1, 6); // returns a random number [1.0, 6.0)  
• int ix = int(random(10)); // returns a random number [0..10)  
• int iy = int(random(1, 6)); // returns a random number [1..6)
```

GJK2013

32

Example: Using random()

```
void setup() { // Create and set canvas  
  size(300, 300);  
  smooth();  
  background(255);  
} // setup()  
  
void draw() {  
  stroke(0);  
  rect(random(width),  
        random(height),  
        random(5, 20),  
        random(5, 20));  
} // draw();
```

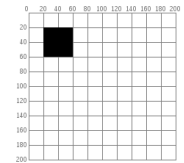


GJK2013

33

2D Transformations: Translate

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

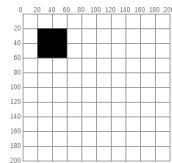


GJK2013

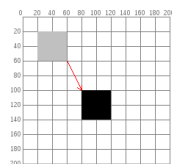
34

2D Transformations: Translate

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



```
rect(20+60, 20+80, 40, 40);
```

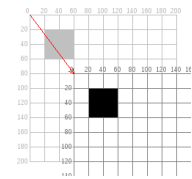


GJK2013

35

2D Transformations: Translate

```
translate(60, 80);  
rect(20, 20, 40, 40);
```



GJK2013

36

Preserving Context

`translate()` will change the coordinate system for the entire duration of the `draw()` cycle. It resets at each cycle.

- Use `pushMatrix()` and `popMatrix()` to preserve context during a `draw()` cycle. i.e.
- `pushMatrix();`
- `translate(<x>, <y>);`
- *<Do something in the new coordinate context>*
- `popMatrix();`

GJK2013

37

Example: House() again!

```
// Draw a simple house

void setup() { // Create and set
  canvas
  size(300, 300);
  smooth();
  background(187, 193, 127);
} // setup()
```



```
void draw() {
```

GJK2013

38

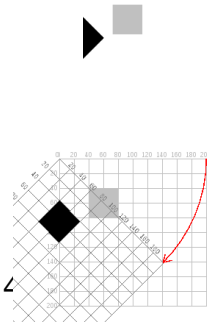
2D Transformations: Rotate

```
void setup() {
  size(200, 200);
  background(255);
  smooth();
  fill(192);
  noStroke();
```

```
  rect(40, 40, 40, 40);
```

```
  pushMatrix();
```

```
  rotate(radians(45));
```



GJK2013

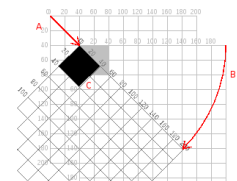
39

2D Transformations: Rotate

```
void setup() {
  size(200, 200);
  background(255);
  smooth();
  fill(192);
  noStroke();

  rect(40, 40, 40, 40);

  pushMatrix(); // move the origin to the pivot point
  translate(40, 40); // then pivot the grid
  rotate(radians(45)); // and draw the square at the origin
  fill(0);
  rect(0, 0, 40, 40);
  popMatrix();
} // setup()
```



GJK2013

40

2D Transformations: Scaling

```
void setup() {
  size(200, 200);
  background(255);

  stroke(128);
  rect(20, 20, 40, 40);

  stroke(0);
  pushMatrix();
  scale(2.0);
  rect(20, 20, 40, 40);
  popMatrix();
} //setup()
```



GJK2013

41

GJK2013

42