EECS 1710
Programming for Digital Media

Week 2 :: Programming Basics

YORK UNIVERSITÉ UNIVERSITY

This Week

Lecture 3:

- Anatomies of a processing sketch
- Language elements & running a program
- Coordinate system in Processing
- Some drawing commands
- Tracing a program

Lecture 4:

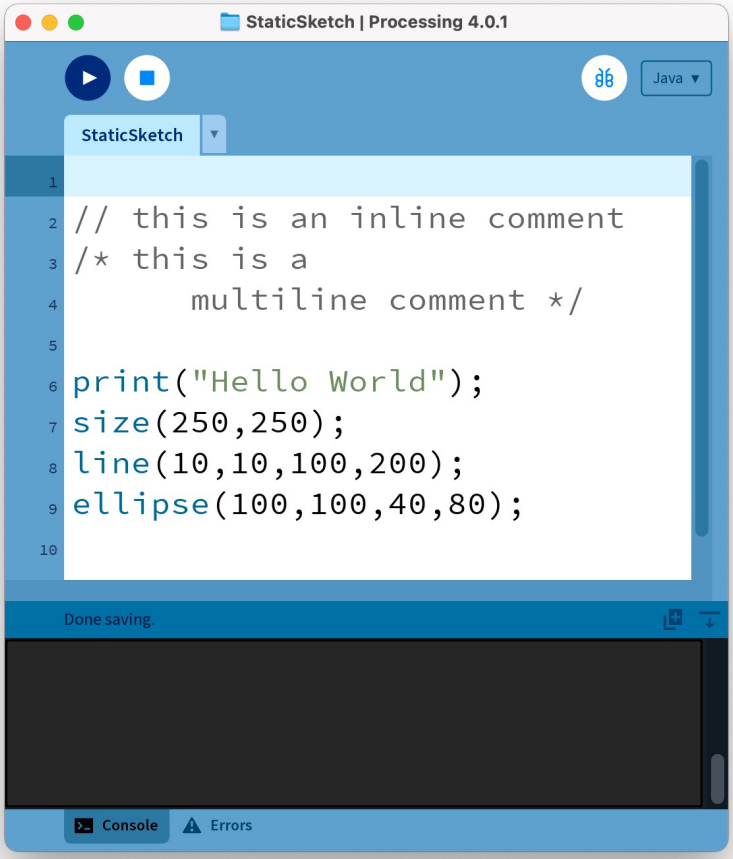
- Variables & Data Types
- Declaration and Assignment

YORK UNIVERSITÉ UNIVERSITY

Topics

- Anatomy of a program
- The declaration statement
- The assignment statement

Processing Program (Sketch)



Program ~ a “sketch”

Stored in a folder called a “sketchbook”
→ Usually a folder within Processing folder in your home directory

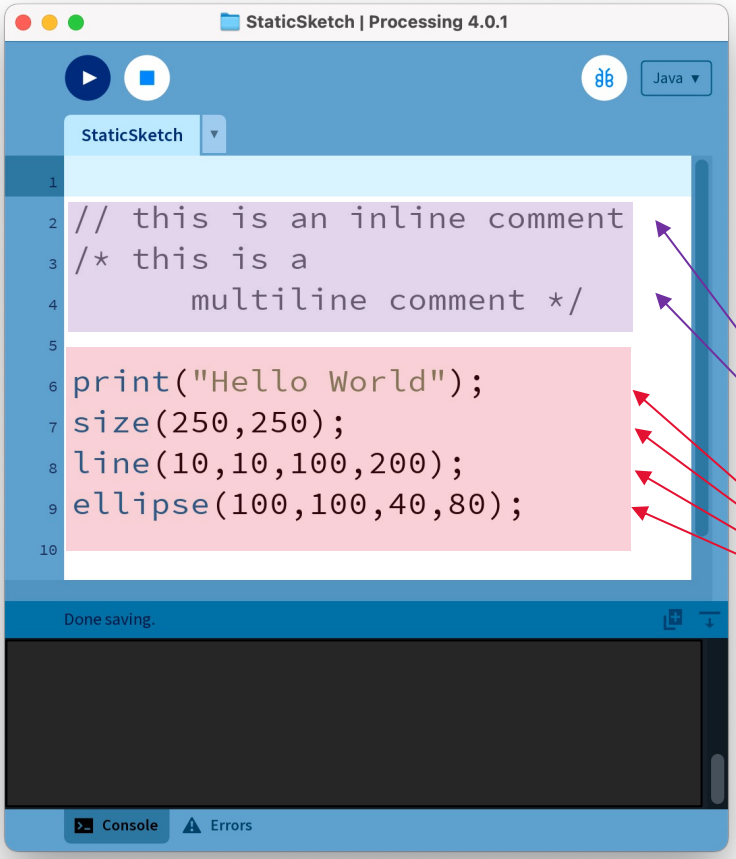
Sketch?
→ Simplified set of java statements
→ When run, is:
- converted into java code
- compiled into machine code
- run by computer

* More on compiling later.. For now this is handled for us by the PDE (Processing Development Environment)

Basic Anatomies of a Processing Program

(sketch)

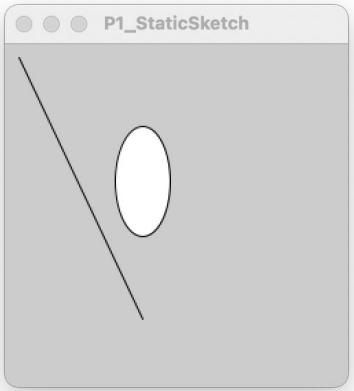
1. Static



statements

comments

commands

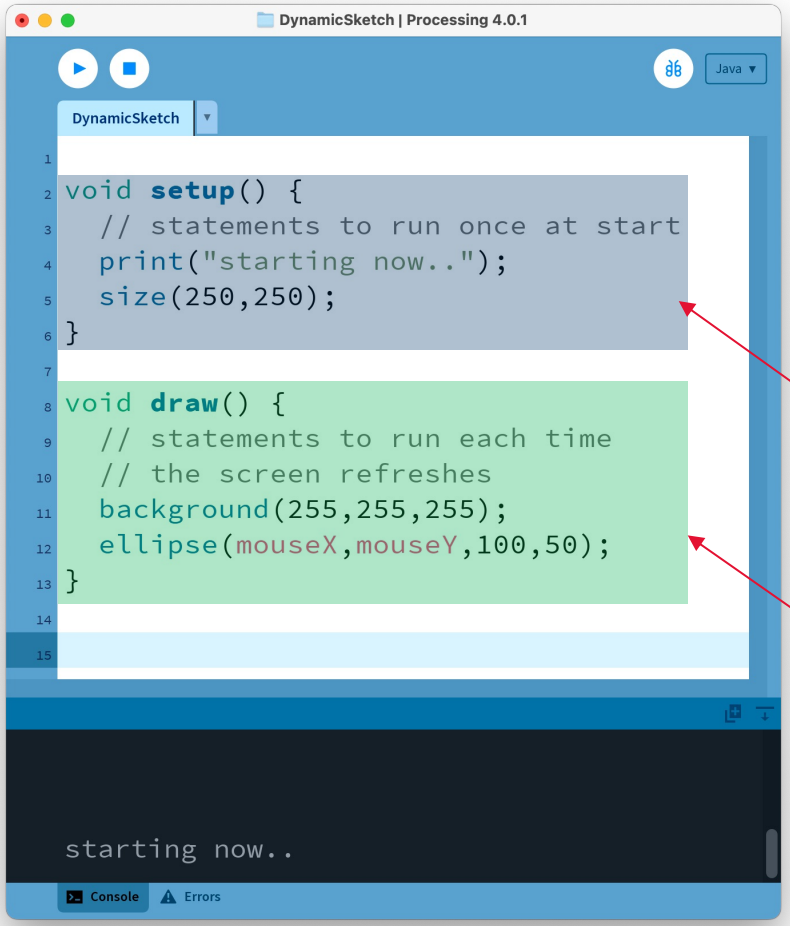


flow

runs statements
start to finish
runs once

Basic Anatomies of a Processing Program

2. Dynamic

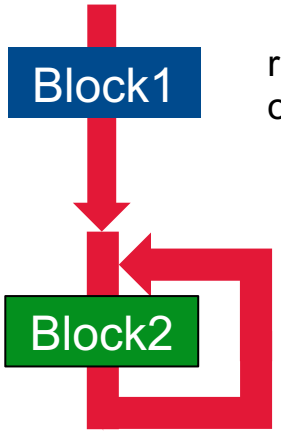


blocks of statements

```
Block1
{
 // setup statements
}
```

```
Block2
{
 // draw statements

}
```

flow

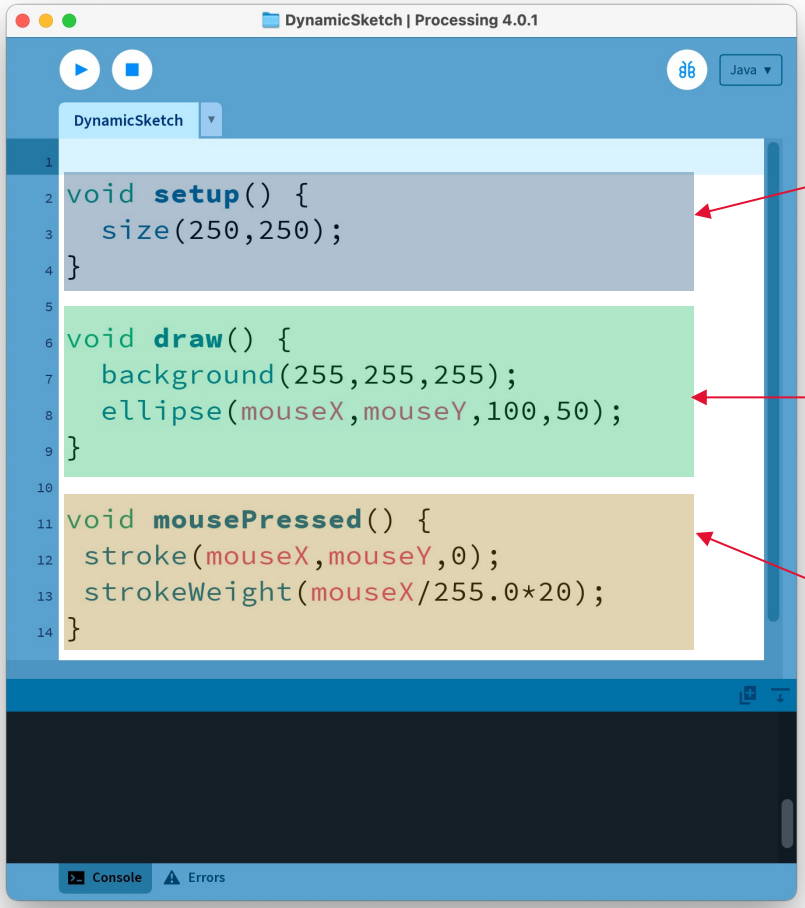


runs once

repeats forever

Basic Anatomies of a Processing Program

3. Dynamic (with Events)



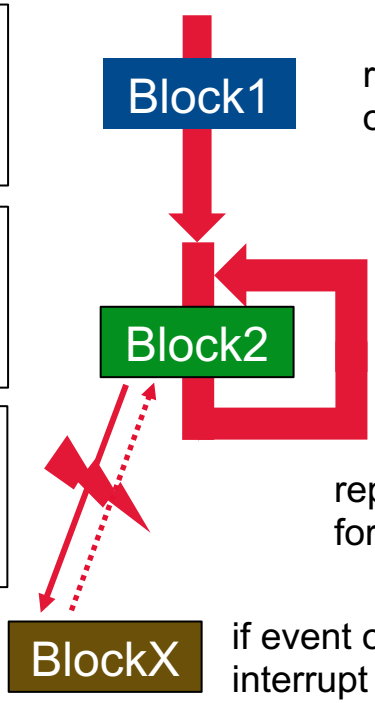
blocks of statements

```
Block1
{
 // setup statements
}
```

```
Block2
{
 // draw statements
}
```

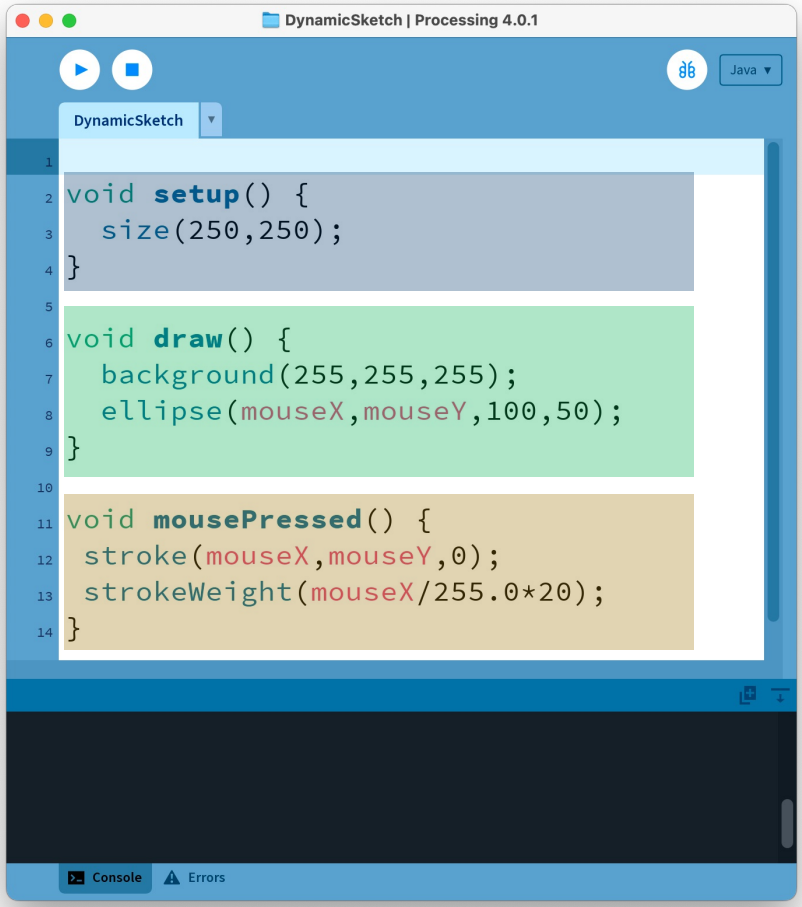
```
BlockX
{
 // other statements
}
```

flow



Basic Anatomies of a Processing Program

3. Dynamic (with Events)



Language Elements

StaticSketch2.pde

```
// this is an inline comment
/* this is a
      multiline comment */

print("Hello World");
int endX = 60;
size(250,250);
line(0,height/2.0,endX*0.75,200);
ellipse(100,100,40,80);
```

DynamicSketch2.pde

```
import processing.pdf.*;

void setup() {
  size(250,250,PDF);
}

void draw() {
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}
```

Keywords Identifiers Literals Operators Separators

Language Elements

StaticSketch2.pde

```
// this is an inline comment
/* this is a
      multiline comment */

print("Hello World");
int endX = 60;
size(250,250);
line(0,height/2.0,endX*0.75,200);
ellipse(100,100,40,80);
```

DynamicSketch2.pde

```
import processing.pdf.*;

void setup() {
  size(250,250,PDF);
}

void draw() {
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}
```

Keywords Identifiers Literals Operators Separators

Language Elements

StaticSketch2.pde

```
// this is an inline comment
/* this is a
      multiline comment */

print("Hello World");
int endX = 60;
size(250,250);
line(0,height/2.0,endX*0.75,200);
ellipse(100,100,40,80);
```

DynamicSketch2.pde

```
import processing.pdf.*;

void setup() {
  size(250,250,PDF);
}

void draw() {
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}
```

Keywords **Identifiers** Literals Operators Separators

Language Elements

StaticSketch2.pde

```
// this is an inline comment
/* this is a
      multiline comment */

print("Hello World");
int endX = 60;
size(250,250);
line(0,height/2.0,endX*0.75,200);
ellipse(100,100,40,80);
```

DynamicSketch2.pde

```
import processing.pdf.*;

void setup() {
  size(250,250,PDF);
}

void draw() {
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}
```

Keywords Identifiers **Literals** Operators Separators

Language Elements

StaticSketch2.pde

```
// this is an inline comment
/* this is a
      multiline comment */

print("Hello World");
int endX = 60;
size(250,250);
line(0,height/2.0,endX*0.75,200);
ellipse(100,100,40,80);
```

DynamicSketch2.pde

```
import processing.pdf.*;

void setup() {
  size(250,250,PDF);
}

void draw() {
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}
```

Keywords Identifiers Literals **Operators** Separators

Language Elements

StaticSketch2.pde

```
// this is an inline comment
/* this is a
      multiline comment */

print("Hello World");
int endX = 60;
size(250,250);
line(0,height/2.0,endX*0.75,200);
ellipse(100,100,40,80);
```

DynamicSketch2.pde

```
import processing.pdf.*;

void setup() {
  size(250,250,PDF);
}

void draw() {
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}
```

Keywords Identifiers Literals Operators **Separators**

Processing/Java Language Elements

Keywords

The keywords are the reserved words plus the literals `true`, `false`, and `null`.

Identifiers

Not be a reserved word; begin with a letter; character set is { `0-9, A-Z, a-z, _` }

Literals

Recognized by the presence of a number, ‘character’, “characters”, or `true`, `false`, or `null`.

Operators

The character set of operators:
`= > < ! ~ ? : & | + - ^ * / %`

Separators

The separators:
`. , ; … ( ) [ ] { }`

YORK UNIVERSITÉ UNIVERSITY

A Note on Terminology

( ) Parentheses

[ ] Brackets

{ } Braces

Java Keywords

Reserved words:

abstract	assert				
boolean	break	byte			
case	catch	char	class	const	continue
default	do	double			
else	enum	extends			
final	finally	float	for		
goto					
if	implements	import	instanceof	int	interface
long					
native	new				
package	private	protected	public		
return					
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	
void	volatile				
while					

Literals: `true, false, null`

*** in Processing sketches, we usually see only a subset of these*

Java Keywords

~ Relate to Flow

Reserved words:

abstract	assert				
boolean	break	byte			
case	catch	char	class	const	continue
default	do	double			
else	enum	extends			
final	finally	float	for		
goto					
if	implements	import	instanceof	int	interface
long					
native	new				
package	private	protected	public		
return					
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	
void	volatile				
while					

Literals: `true`, `false`, `null`

Java Keywords

~ Relate to Data

Reserved words:

abstract	assert				
boolean	break	byte			
case	catch	char	class	const	continue
default	do	double			
else	enum	extends			
final	finally	float	for		
goto					
if	implements	import	instanceof	int	interface
long					
native	new				
package	private	protected	public		
return					
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	
void	volatile				
while					

Literals: `true`, `false`, `null`

Java Keywords

~ Relate to packaging/ organizing code

Reserved words:

abstract	assert				
boolean	break	byte			
case	catch	char	class	const	continue
default	do	double			
else	enum	extends			
final	finally	float	for		
goto					
if	implements	import	instanceof	int	interface
long					
native	new				
package	private	protected	public		
return					
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	
void	volatile				
while					

Literals: `true`, `false`, `null`

Important Keywords

- **class**
  - contains all the elements of an individual program
  - similar/related classes can be organized into groups (packages)

- **import**
  - contains references to other classes (programs) that we may want to include and use within our program

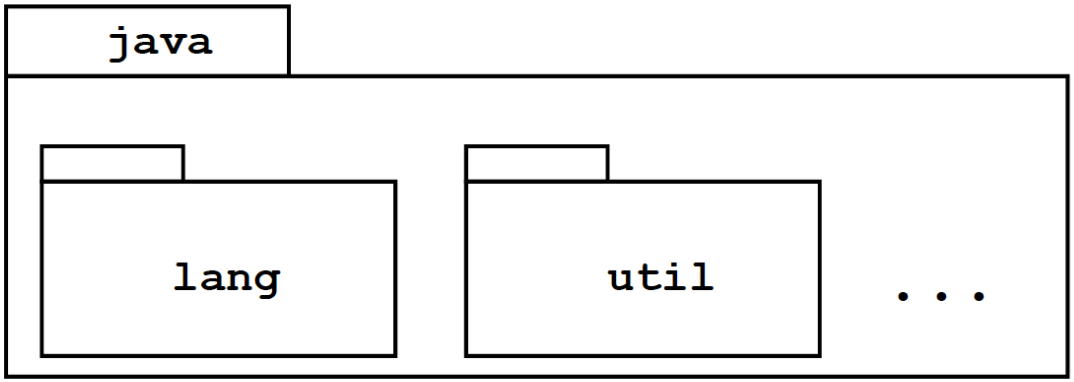
Typical java import

```
import java.lang.System;
```

Package

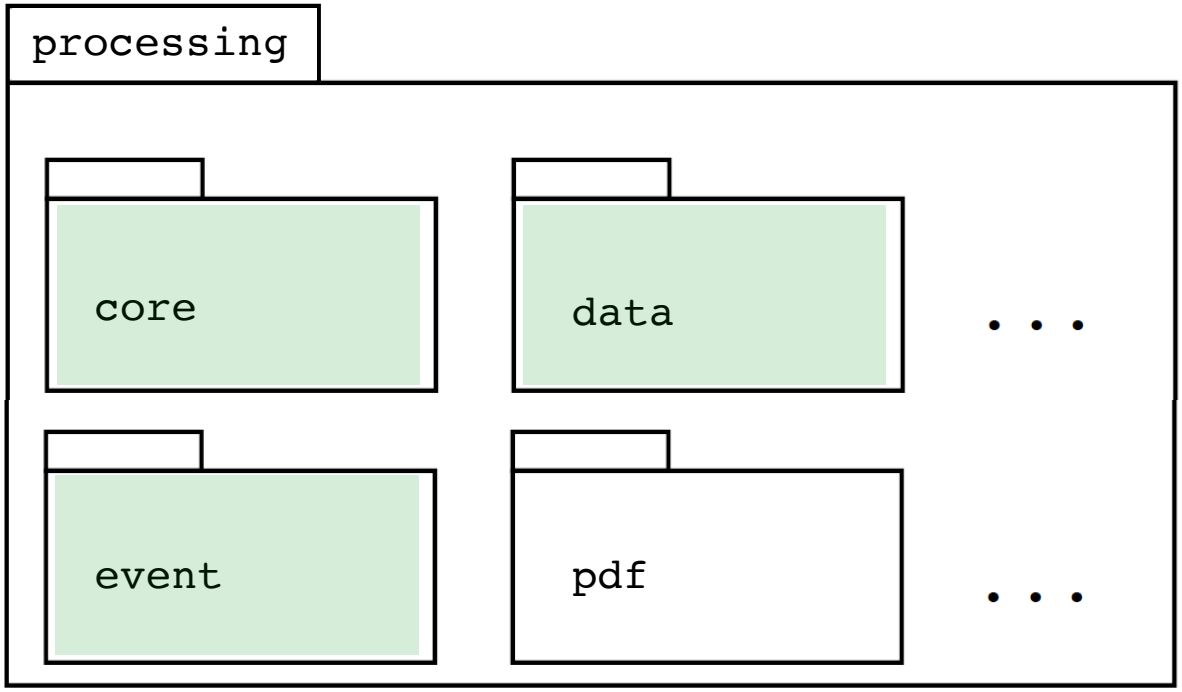
Subpackage

Class



Core Processing packages

- Most common packages are automatically included (added by PDE) so we don't have to explicitly import



What happens when we “play” a sketch?

- Code
  - A program is written up as **source code**
    - The recipe is written in a high level language (e.g. Processing/Java - easy for humans to read)
  - A program is converted into **machine code**
    - So computer can understand and run it

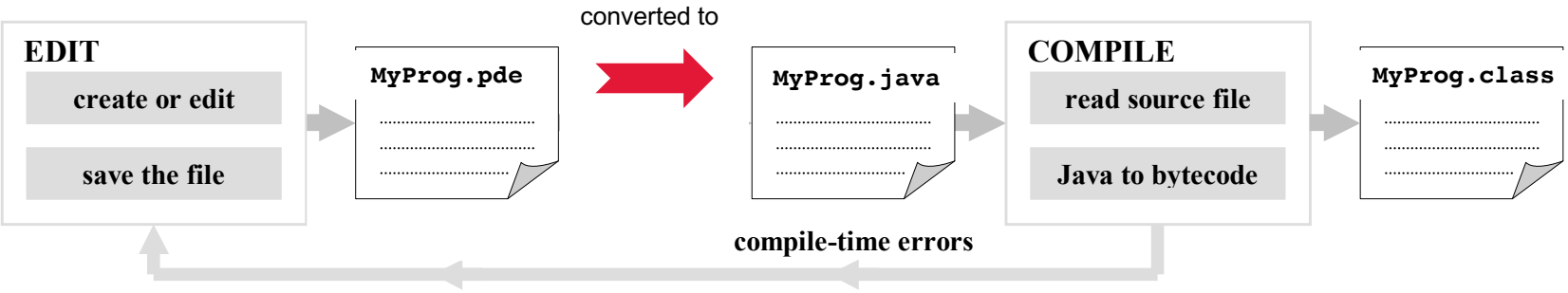
Byte Code

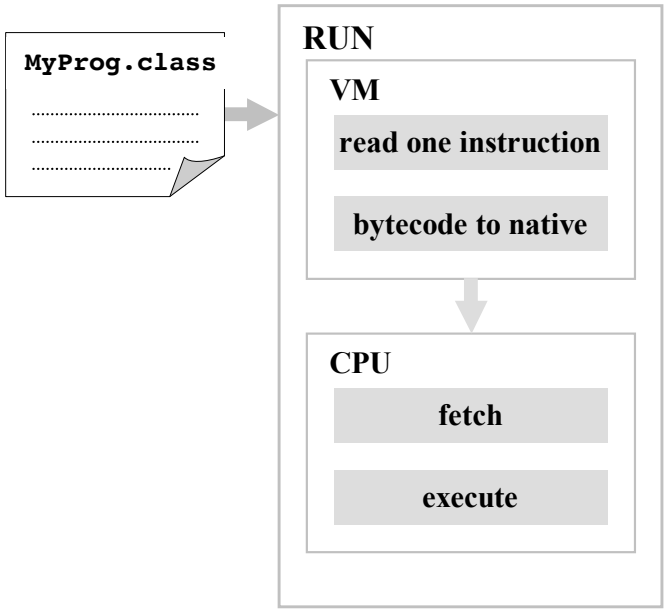
- Normally **machine code** is unique to different machines
  - So different computers generally need their own unique version of the program
    - Converting source code into machine code => “compiling”
    - the software that does this conversion => a “compiler”

- In Java, source code is instead turned into an intermediary form called **Byte Code**
  - Byte code is closer to machine code, but common to all machines
  - Byte code is portable (same code can run on any machine)

- How?
  - Via an interpreter (Java Virtual Machine – JVM)
  - JVM is a key component in the Java Runtime Environment (JRE)

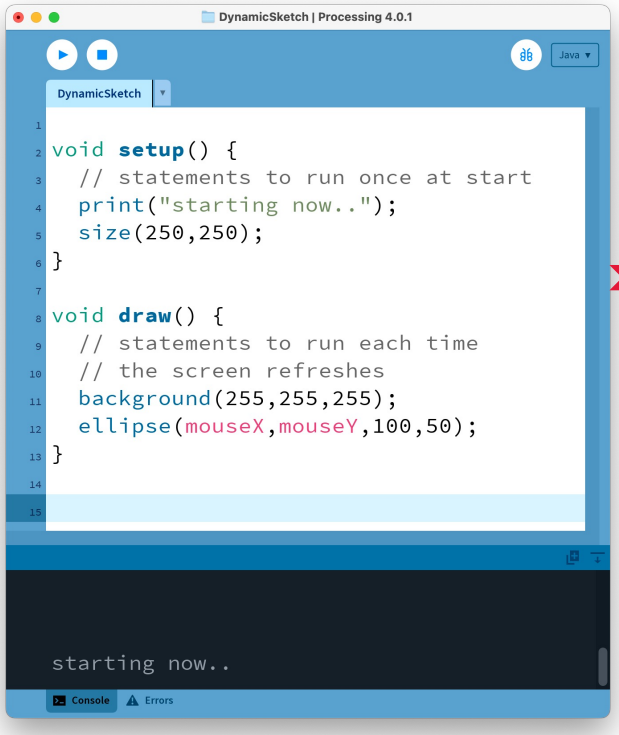
defining & executing a Processing sketch





Source code

DynamicSketch.pde



Processing ~ Java (lite)

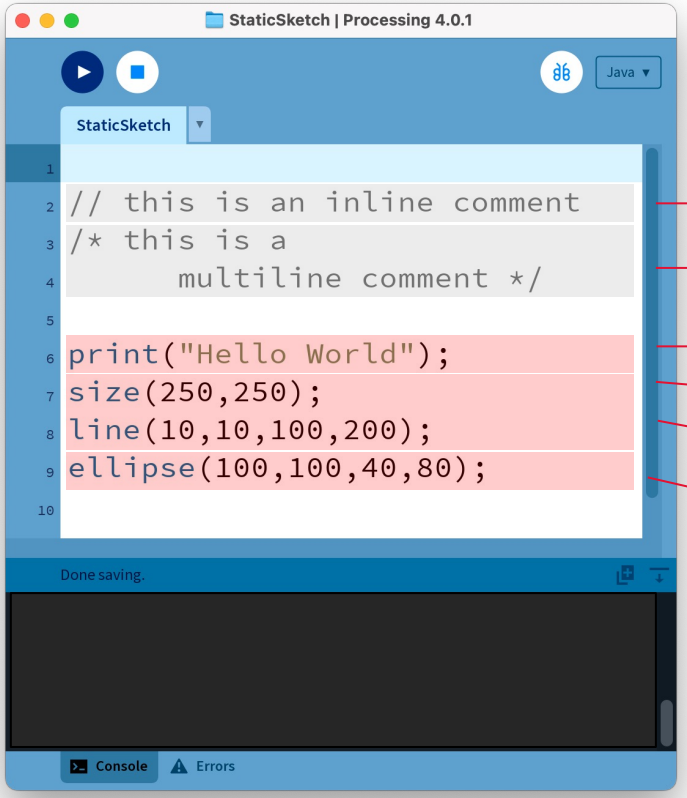
DynamicSketch.java

```
import processing.core.PApplet;

public class DynamicSketch extends PApplet {

      public void settings() {
       print("starting now..");
       size(250,250);
      }

      public void draw() {
       background(255,255,255);
       ellipse(mouseX,mouseY,100,50);
      }

      public static void main(String[] args) {
       String[] processingArgs = {"HelloSketch"};
       DynamicSketch mySketch = new DynamicSketch();
       PApplet.runSketch(processingArgs, mySketch);
      }
}
```

Java (full) – note the extra scaffolding!

DynamicSketch.class (Byte Code)

Execution = running the byte code on the JVM (Tracing = to trace steps in order of execution)



ignored

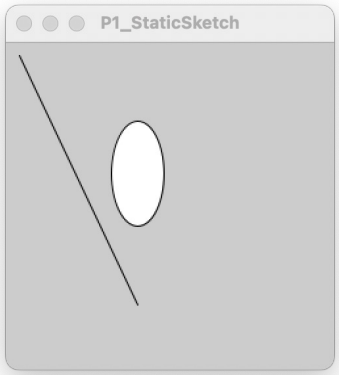
ignored

prints “Hello World” to console

creates application window (250 wide x 250 high)

draws a line from (10,10) to (100,200)

draws an ellipse size 40 wide,80 high centred at (100,100)



Co-ordinate System in Processing

Cartesian Co-ords (typical)

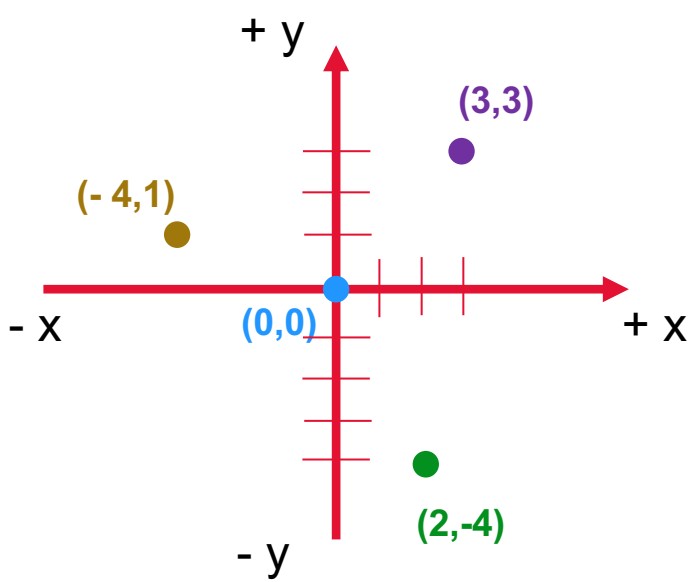
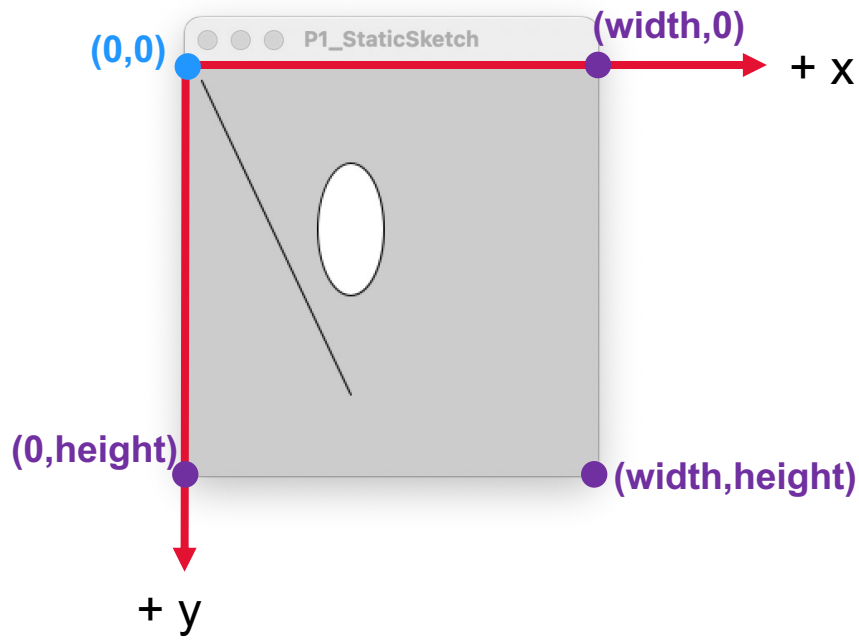


Image Co-ords (Processing)



Some useful drawing commands

`arc()`	Draws an arc in the display window
`circle()`	Draws a circle to the screen
`ellipse()`	Draws an ellipse (oval) in the display window
`line()`	Draws a line (a direct path between two points) to the screen
`point()`	Draws a point, a coordinate in space at the dimension of one pixel
`quad()`	A quad is a quadrilateral, a four sided polygon
`rect()`	Draws a rectangle to the screen
`square()`	Draws a square to the screen
`triangle()`	A triangle is a plane created by connecting three points

https://processing.org/reference/#shape

line()

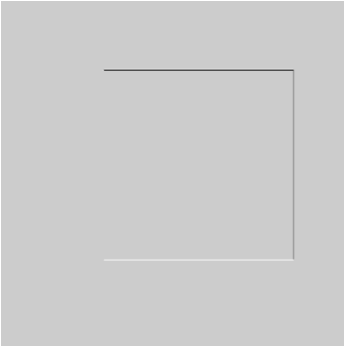
Syntax

```
line(x1, y1, x2, y2)
line(x1, y1, z1, x2, y2, z2)
```

Parameters

x1	(float)	x-coordinate of the first point
y1	(float)	y-coordinate of the first point
x2	(float)	x-coordinate of the second point
y2	(float)	y-coordinate of the second point
z1	(float)	z-coordinate of the first point
z2	(float)	z-coordinate of the second point

```
//Example
size(400, 400);
line(120, 80, 340, 80);
stroke(126);
line(340, 80, 340, 300);
stroke(255);
line(340, 300, 120, 300);
```



https://processing.org/reference/line_.html

int, float ?? → numeric data types

- int = integers
- (whole numbers)

Can be positive/negative

No decimal places

e.g. 1
-15
189

- float = floating point
- (type of real number)

Can be positive/negative

Decimals allowed

e.g. 0.4
-1.45
189.2411

More on data types next lecture

YORK UNIVERSITÉ UNIVERSITY

stroke()

- sets colour of a stroke

Syntax

```
stroke(rgb)
stroke(rgb, alpha)
stroke(gray)
stroke(gray, alpha)
stroke(v1, v2, v3)
stroke(v1, v2, v3, alpha)
```

Parameters

rgb	(int)	color value in hexadecimal notation
alpha	(float)	opacity of the stroke
gray	(float)	specifies a value between white and black
v1	(float)	red or hue value (depending on current color mode)
v2	(float)	green or saturation value (depending on current color mode)
v3	(float)	blue or brightness value (depending on current color mode)

More on colour later, but usually specified as 3 values (red,green,blue)

Where each value (0-255)
0=no colour, 255 = full colour

i.e.

red = (255,0,0)
blue = (0,0,255)
green = (0,255,0)
purple = (255,0,255)
white = (255,255,255)
black = (0,0,0)

** many colours from mixing these

https://processing.org/reference/stroke_.html

strokeWeight()

- sets width of a stroke

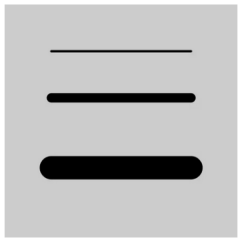
Syntax `strokeWeight(weight)`

Parameters

`weight` (`float`) the weight (in pixels) of the stroke

Examples

```
size(400, 400);
strokeWeight(4);  // Default
line(80, 80, 320, 80);
strokeWeight(16);  // Thicker
line(80, 160, 320, 160);
strokeWeight(40);  // Beastly
line(80, 280, 320, 280);
```



stroke() controls outline of a shape

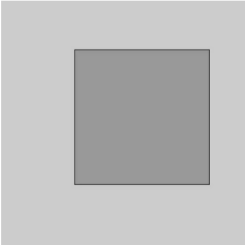
fill() controls the space within a shape

fill()

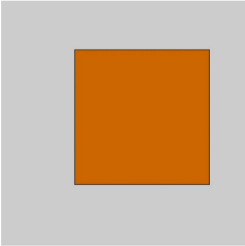
Syntax

```
fill(rgb)
fill(rgb, alpha)
fill(gray)
fill(gray, alpha)
fill(v1, v2, v3)
fill(v1, v2, v3, alpha)
```

```
size(400, 400);
fill(153);
rect(120, 80, 220, 220);
```



```
size(400, 400);
fill(204, 102, 0);
rect(120, 80, 220, 220);
```



Parameters

rgb	(int)	color variable or hex value
alpha	(float)	opacity of the fill
gray	(float)	number specifying value between white and black
v1	(float)	red or hue value (depending on current color mode)
v2	(float)	green or saturation value (depending on current color mode)
v3	(float)	blue or brightness value (depending on current color mode)

https://processing.org/reference/fill_.html

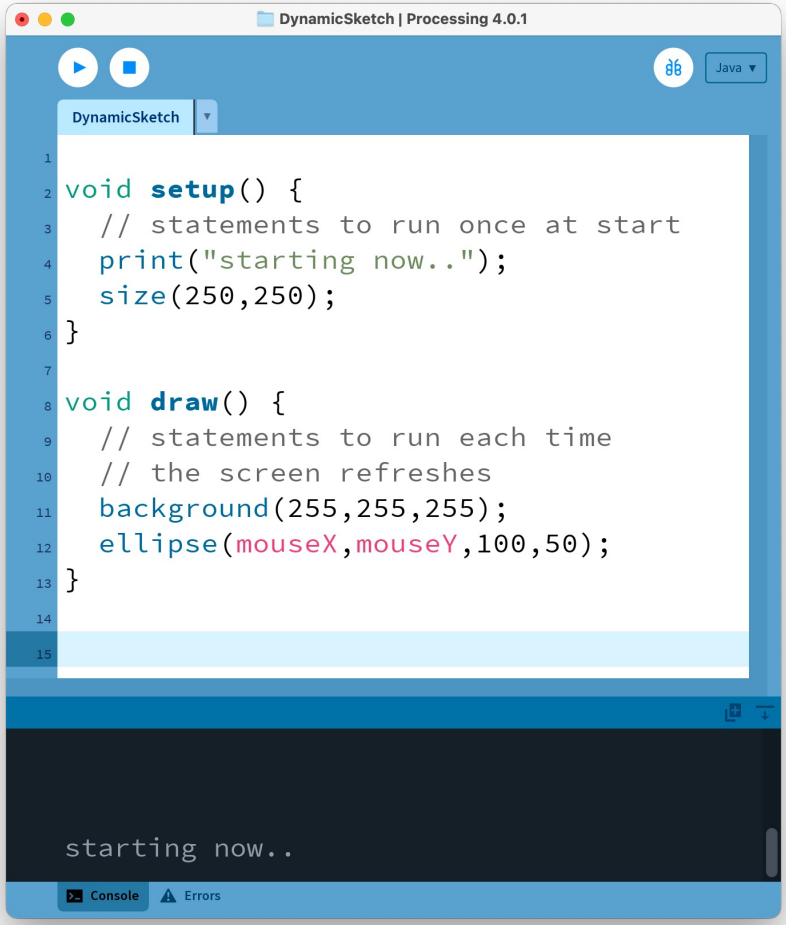
This week, goal is to create a 2D cartoon with two/three of these!

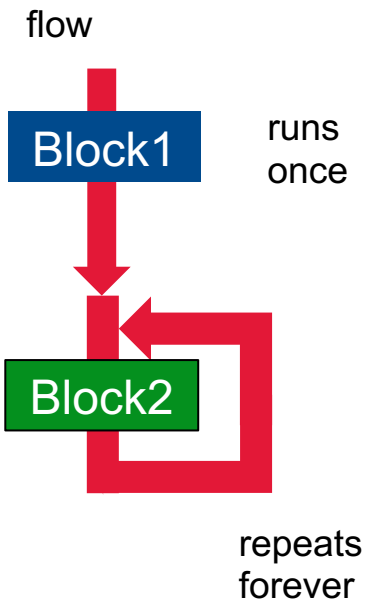
`arc()`	Draws an arc in the display window
`circle()`	Draws a circle to the screen
`ellipse()`	Draws an ellipse (oval) in the display window
`line()`	Draws a line (a direct path between two points) to the screen
`point()`	Draws a point, a coordinate in space at the dimension of one pixel
`quad()`	A quad is a quadrilateral, a four sided polygon
`rect()`	Draws a rectangle to the screen
`square()`	Draws a square to the screen
`triangle()`	A triangle is a plane created by connecting three points

https://processing.org/reference/#shape

YORK UNIVERSITÉ UNIVERSITY

Tracing DynamicSketch.pde?

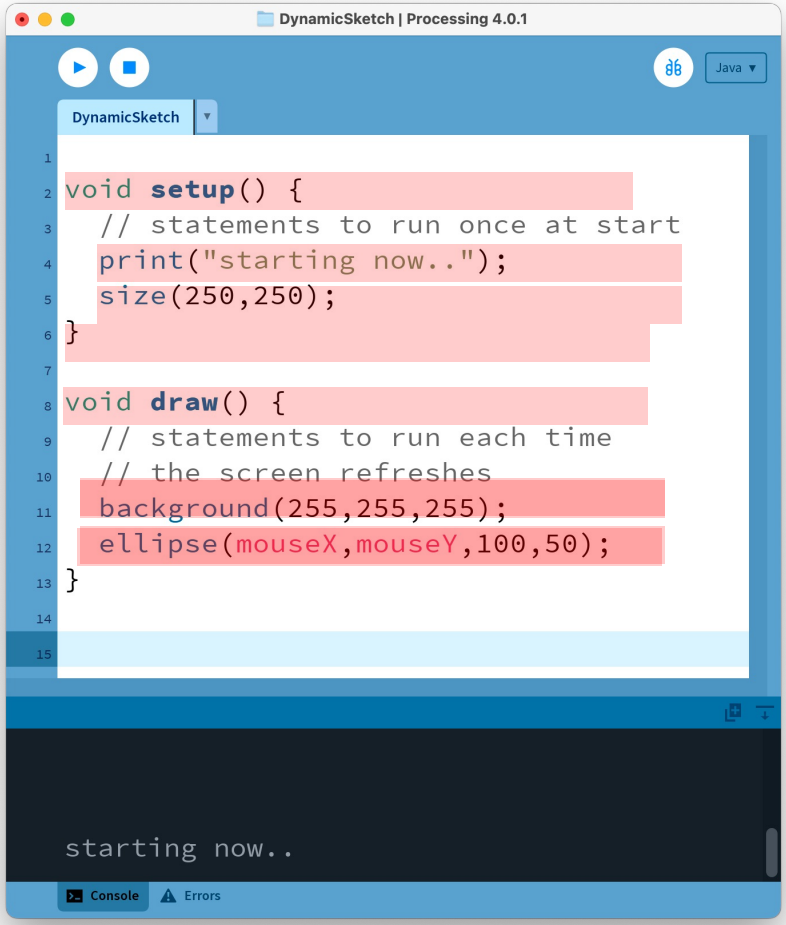


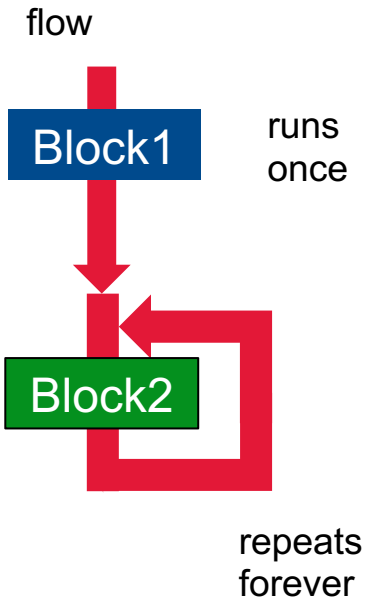


Variables

- Variables are identifiers we create (names) for containers that will store certain types of data values
- We can create our own, or utilize some *pre-defined* variables that processing provides for us
- (mouseX, mouseY) are *pre-defined* variables that hold the current mouse position → i.e. the (x,y) position of the cursor on the application window (in image coordinates)
- This is useful as we can cause changes in our drawings by moving the mouse!

Tracing DynamicSketch.pde?





Next Lecture

- Defining **variables** to hold/store data values
  - Declaration
    - built-in (primitive) data types
  - Assignment
  - How data types encode/represent data
  - Naming Conventions