COMS 1002
Computing in the Arts

Track overview - 3 modules

1) Visuals

Generative art and algorithmic creativity

2) Audio

“Performing your code” - making music with live coding

3) Physical Computing

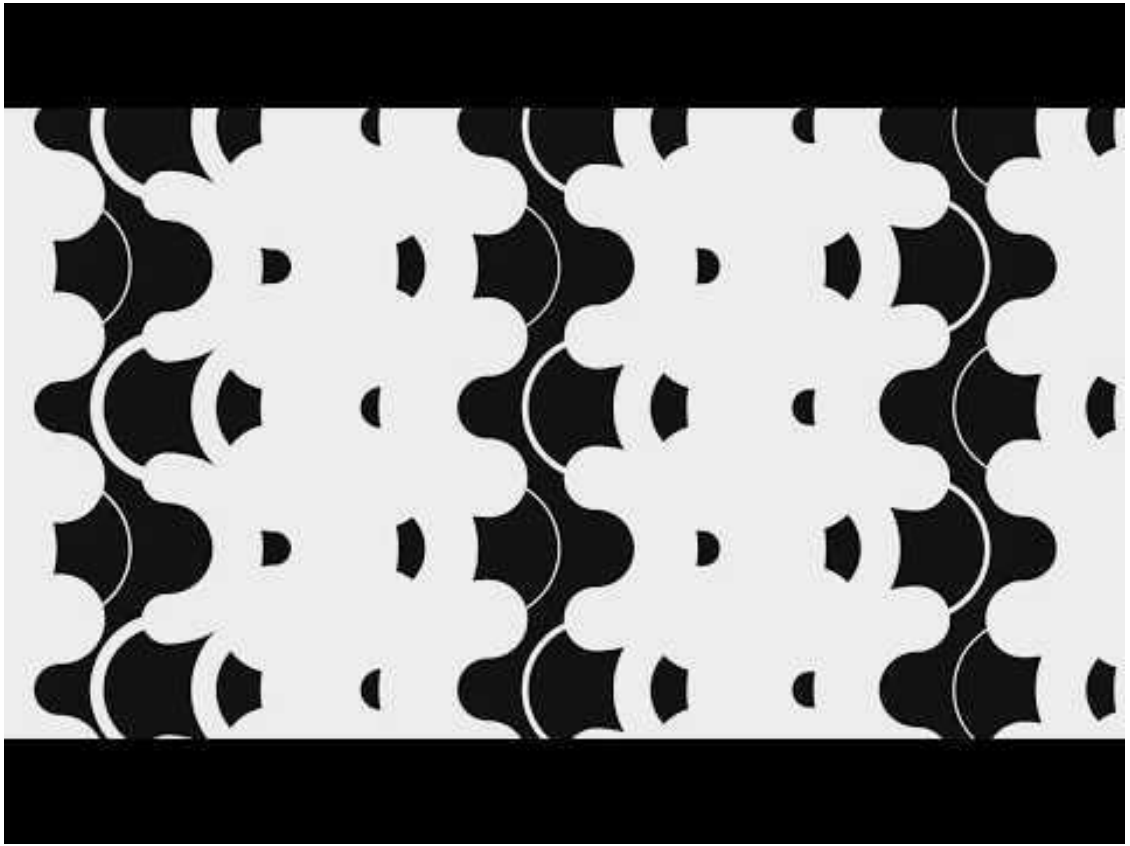
Connecting our code to the physical world

Module 1: Visuals

processing.py

Can computers be creative?

How do we use computation to guide/augment/inform our creative impulses?



Module 2: Audio

SonicPi

Live Coding: Play your laptop like an instrument

How can code be used beyond creating products?

How do we write code that controls data over time?

Module 3: Physical Computing

required purchase: BBC micro:bit (~$20 USD)

How are physical, digital devices created?

What are the challenges in connecting code to the physical world?

Track overview - 3 modules

1) Visuals

Generative art and algorithmic creativity

2) Audio

“Performing your code” - making music with live coding

3) Physical Computing (**Purchase a microbit now! Either version is fine)**

Connecting our code to the physical world

Track overview - lectures

Each module has two lectures.

Lecture 1 will be implementation focused and give you the tools you need to complete the associated project.

Lecture 2 will be theory focused and give us a chance to discuss issues in computing in the arts and computational creativity.

Quick breakout rooms of ~4

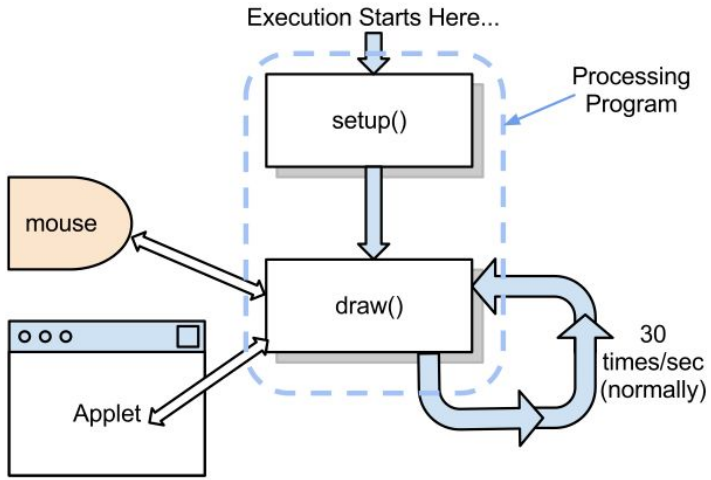
Introduce yourselves:

1) Which module are you most excited for and why?
2) Why did you choose this track?

Module 1: Visuals with Processing

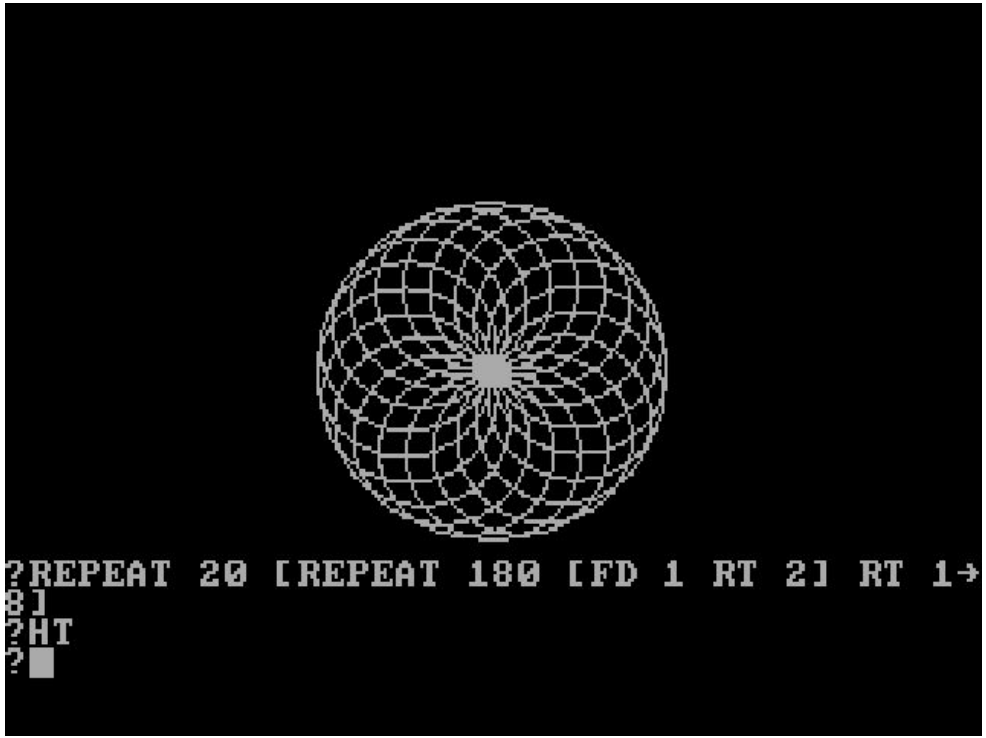
- programming environment
- visually oriented applications
- “for artists, by artists”

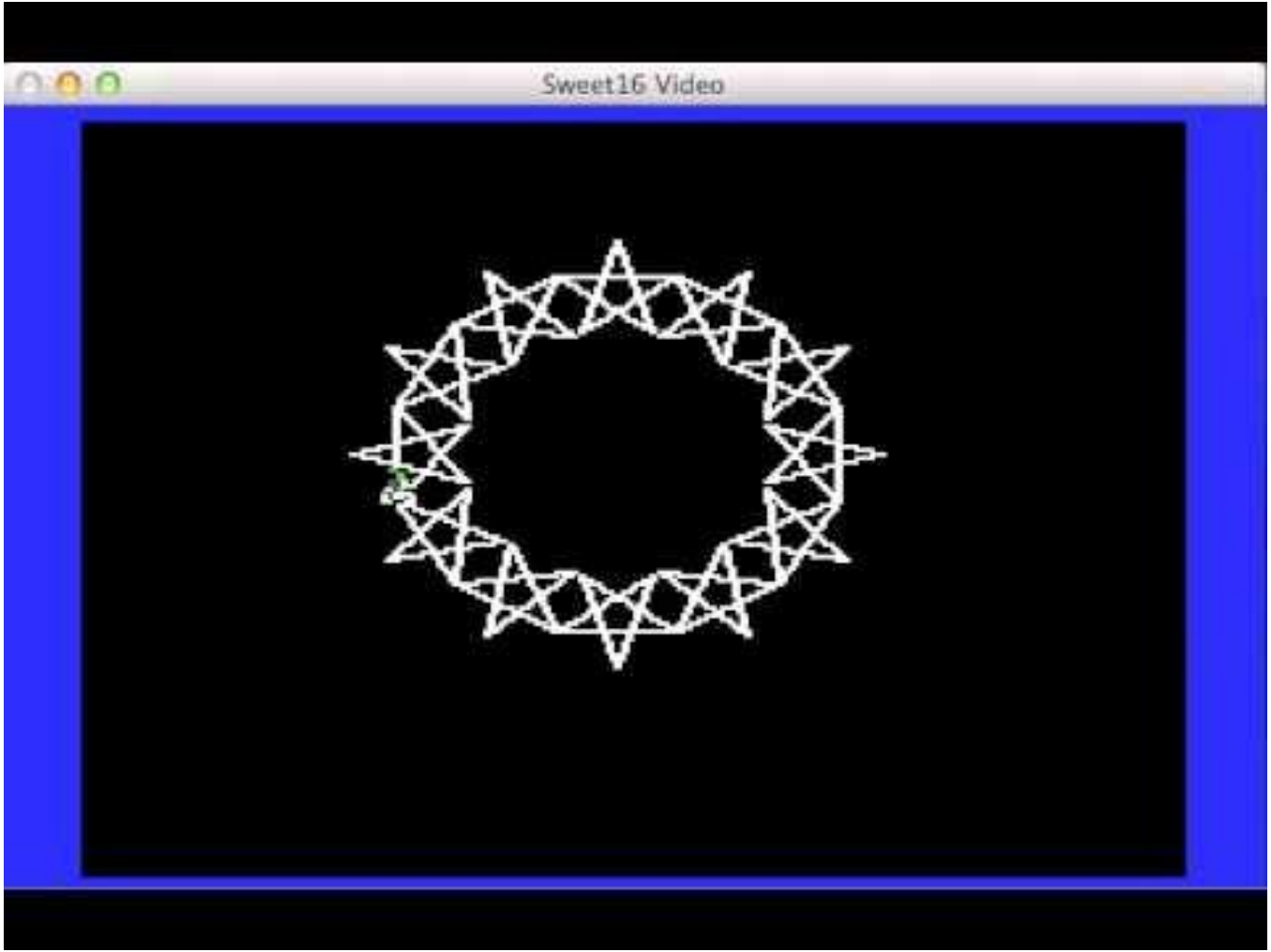
Own its origins to “Turtle Graphics”

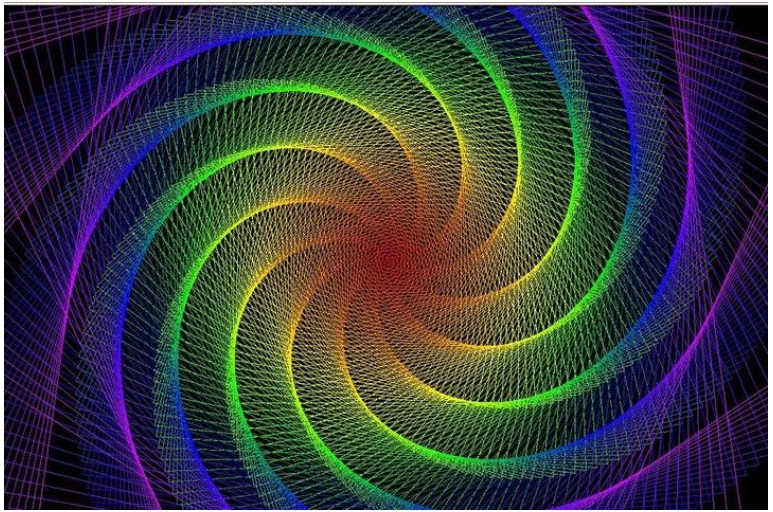


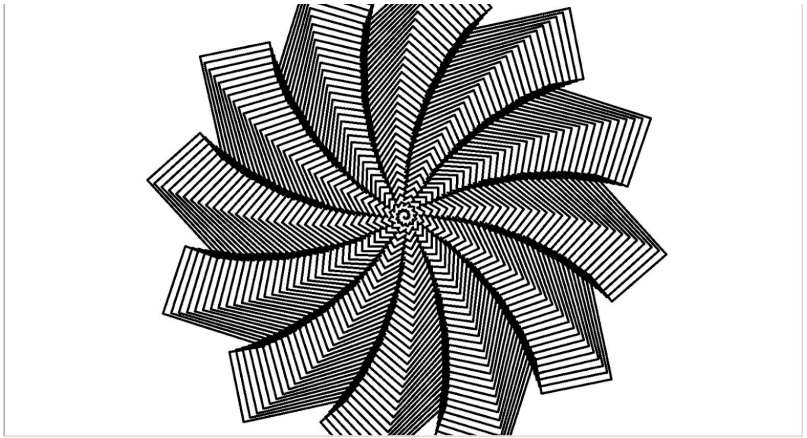
history of turtle graphics

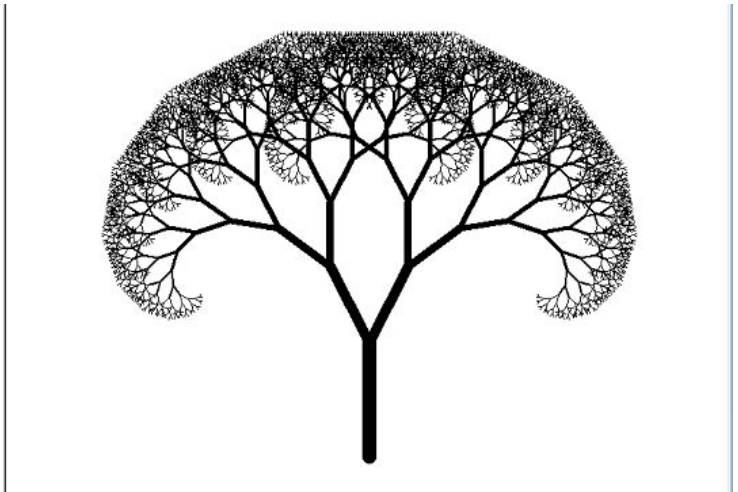
Turtle graphics was part of the original Logo programming language developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967.

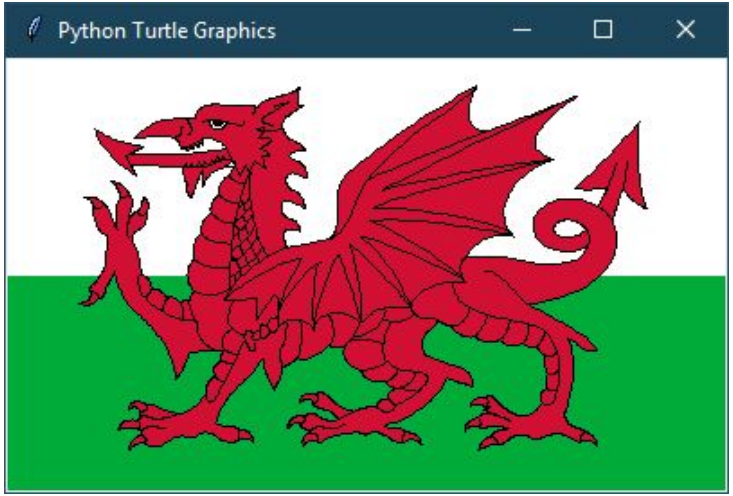



Sweet16 Video








Python Turtle Graphics

Turtle Graphics vs Processing

Turtle Graphics:

- Low-level commands, drawing lines, simple commands (left, forward, etc)
- Used to teach basics of programming/computational processes

Processing:

- High-level language, lines/shapes/images/etc, large set of API commands,
- Used to teach basics of programming and to create art

Processing in practice

https://vimeo.com/320513063 (play from 0:53 to 1:58)

Processing example code

1) Setting up canvas
2) Drawing shapes
3) Input

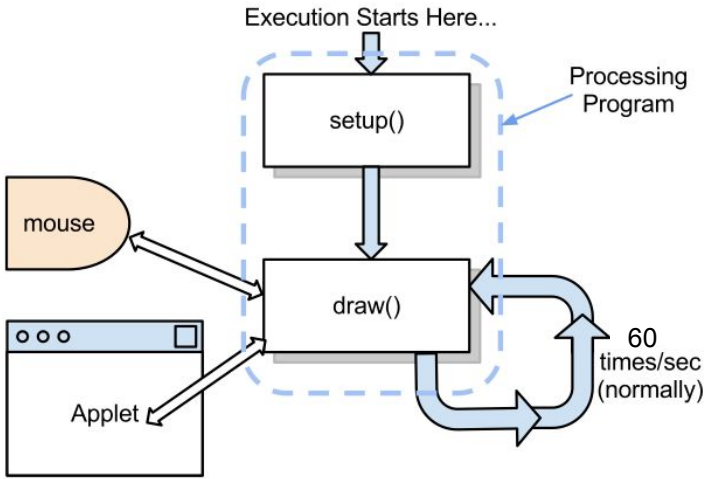
Demo time... (Mouse2D)

```
"""
Mouse 2D.

Moving the mouse changes the position of the box.
"""


def setup():
    size(640, 360)
    noStroke()
    rectMode(CENTER)


def draw():
    background(51)
    fill(255, 204)
    rect(mouseX, height / 2, mouseY / 2 + 10, mouseY / 2 + 10)
```

Basic flow of a Processing program

Setup your “canvas” with the setup() function

Animate the “canvas” with draw()

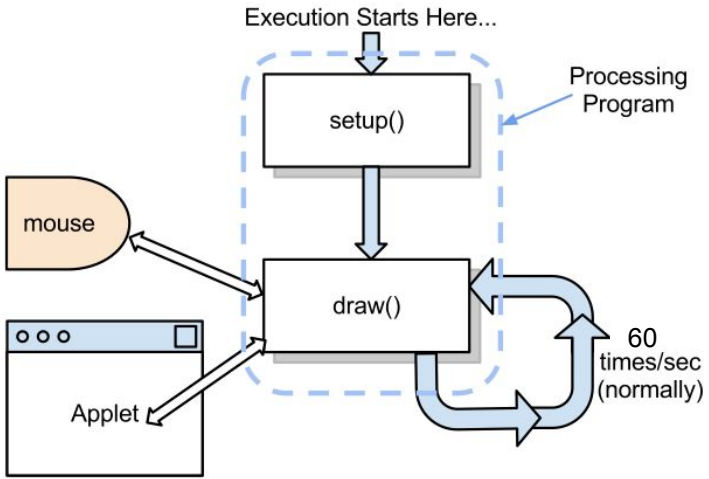


setup()

This allows you to set global properties for your canvas.

Only runs once at the beginning of execution.

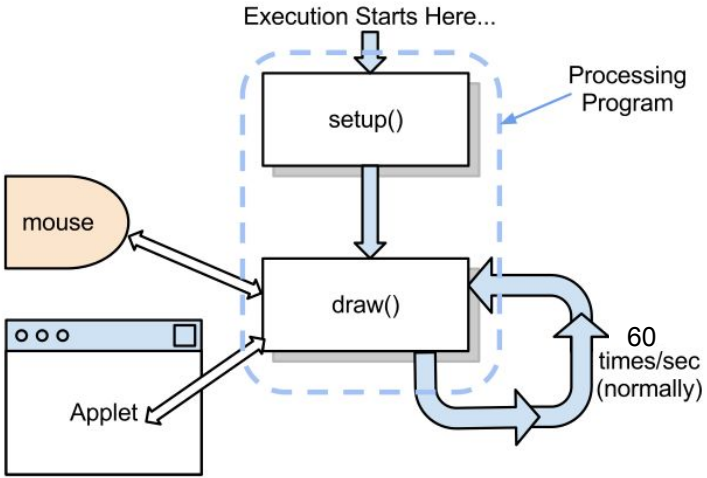
Variable declared in the scope of setup do not exist outside that scope!



setup()

Examples

```
a = 0

def setup():
        size(200, 200)
        background(0)
        noStroke()
        fill(102)

def draw():
        background(0)
        global a
        a = (a + 1) % width
        rect(a, 10, 2, 80)
```



Description

The `setup()` function is called once when the program starts. It's used to define initial enviroment properties such as screen size and background color and to load media such as images and fonts as the program starts. There can only be one `setup()` function for each program and it shouldn't be called again after its initial execution. Note: Variables declared within `setup()` are not accessible within other functions, including `draw()`.

a few key functions

size() - set the size of the display window

background() - set the background color of the window. Can also be set to an image.

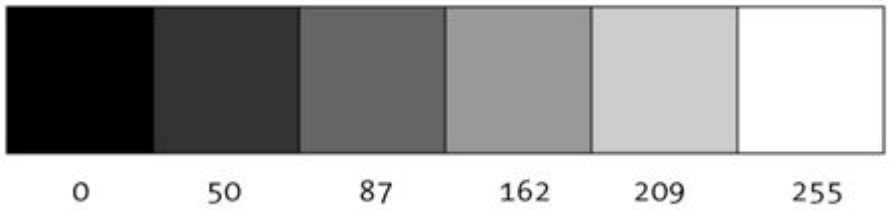
fill()/noFill() - the default color to use when filling shapes (like rect)

stroke/noStroke() - the default color to use for the outline of shapes (like rect)

colorMode() - the color mode to use. Defaults to RGB

colors on the computer

values **0-255** (2^8, 8-bit color)



colorMode()

RGB - Red, Green, Blue

24-bit color, 8-bits for R, G, and B

RGBA - Red, Green, Blue, Alpha

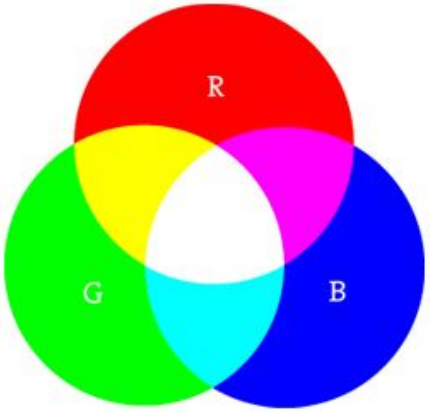
HSB - Hue, Saturation, Brightness

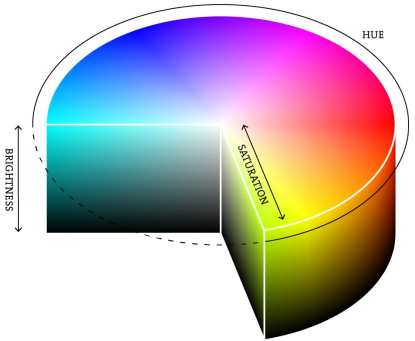
RGB and HSB express the same set of colors

On the computer screen,

we are mixing light, not paint.

Demo (LinearGradient)

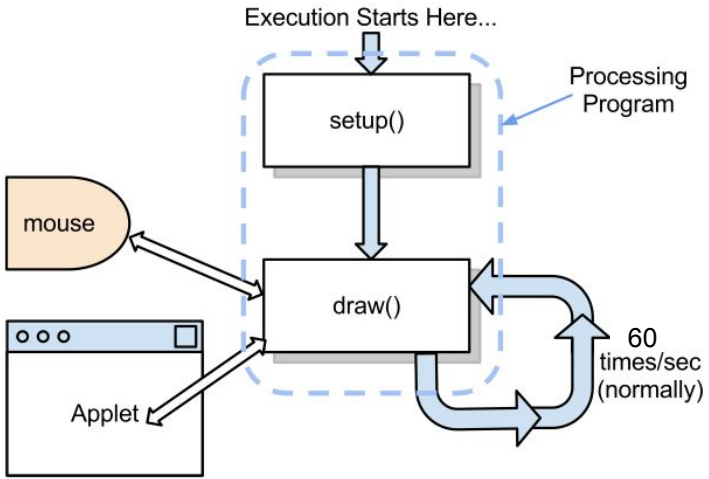




draw()

Setup your “canvas” with the setup() function

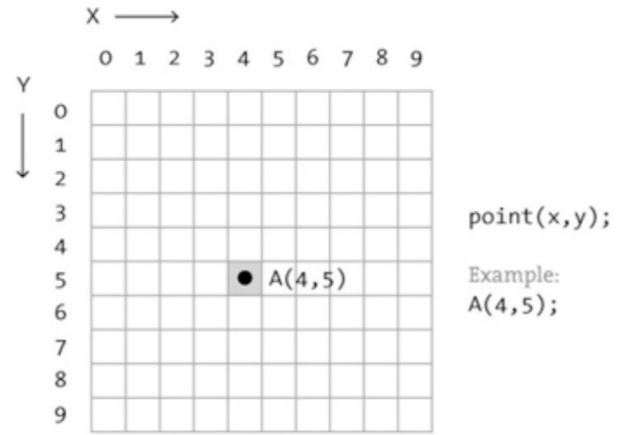
Animate the “canvas” with draw()



drawing shapes

drawing in processing users a grid system, much like Turtle Graphics

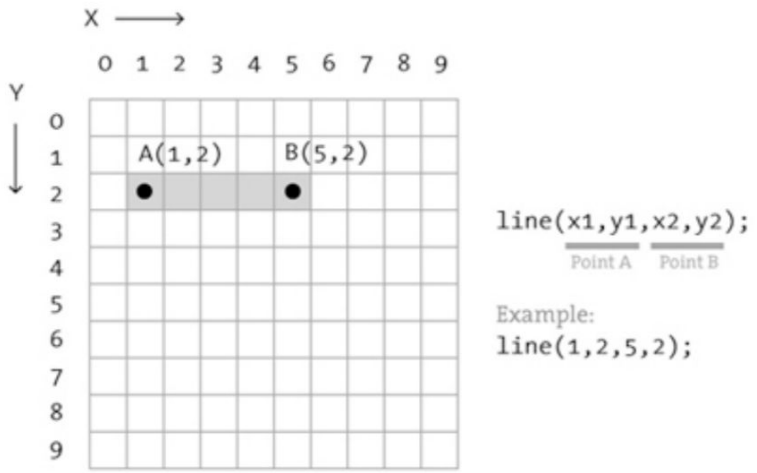
```
point(x, y);
```



drawing shapes

drawing in processing users a grid system, much like Turtle Graphics

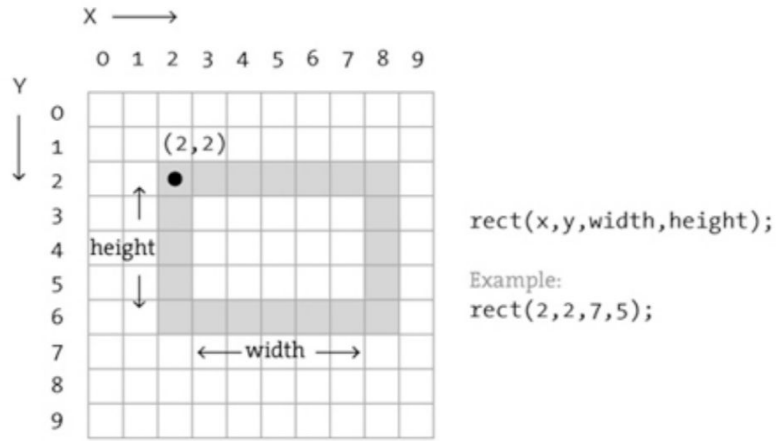
```
line(x1, y1, x2, y2);
```



drawing shapes

drawing in processing users a grid system, much like Turtle Graphics

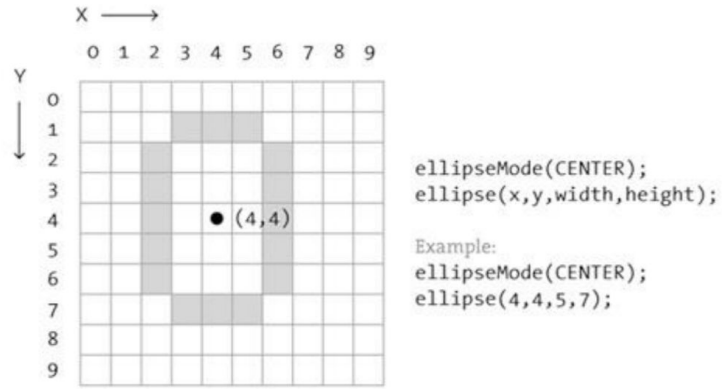
```
rect(x, y, width, height);
```



drawing shapes

drawing in processing users a grid system, much like Turtle Graphics

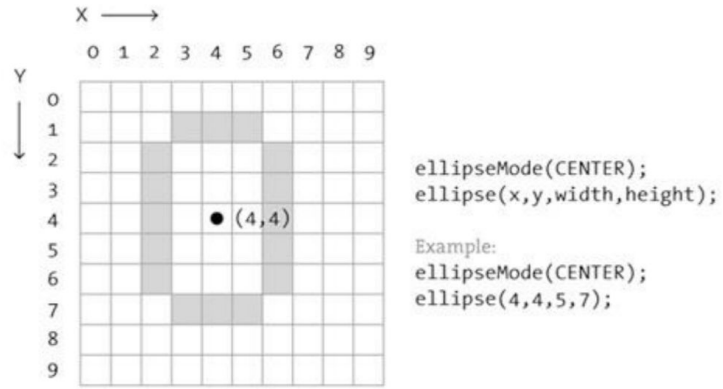
```
ellipseMode(CENTER);
ellipse(x, y, width, height);
```



drawing shapes

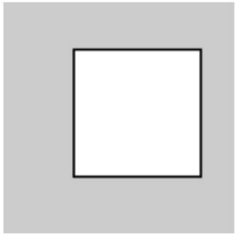
drawing in processing users a grid system, much like Turtle Graphics

```
ellipseMode(CENTER);
ellipse(x, y, width, height);
```



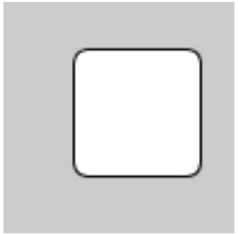
drawing shapes

Usually, we draw much larger shapes, so we cannot see the pixels

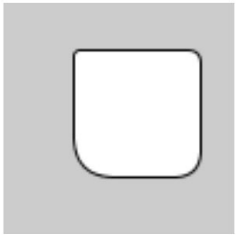


```
rect(30, 20, 55, 55)
```

Note: shapes are rendered one at a time, and will be layered based on code order



```
rect(30, 20, 55, 55, 7)
```



```
rect(30, 20, 55, 55, 3, 6, 12, 18)
```

manipulating shapes

translate()

rotate()

scale()

Demo time (myRotate)

user input/interactivity

Allows your animations to be interactive

mouse input

Allows your animations to be interactive

Demo:

- Mouse2D,
- MousePress

```
void mouseClicked(){

  if(mouseButton == LEFT)
    fill(0);

  else if(mouseButton == RIGHT)
    fill(255);

  else
    fill(126);
}
```

```
void mousePressed()

void mouseReleased()

void mouseClicked()

void mouseDragged()

void mouseMoved()

void mouseWheel()
```

```
mouseX
mouseY
pmouseX
pmouseY
```

keyboard input

Allows your animations to be interactive

Demo:

Keyboard

```
void keyTyped(){

  if(key == 'b')
    fill(0);

  else if(key == 'w')
    fill(255);

  else
    fill(126);
}
```

```
void keyPressed()

void keyReleased()

void keyTyped()


keyPressed
key
keyCode
```

Understanding Frame Rate

How does the xrange value change the framerate?

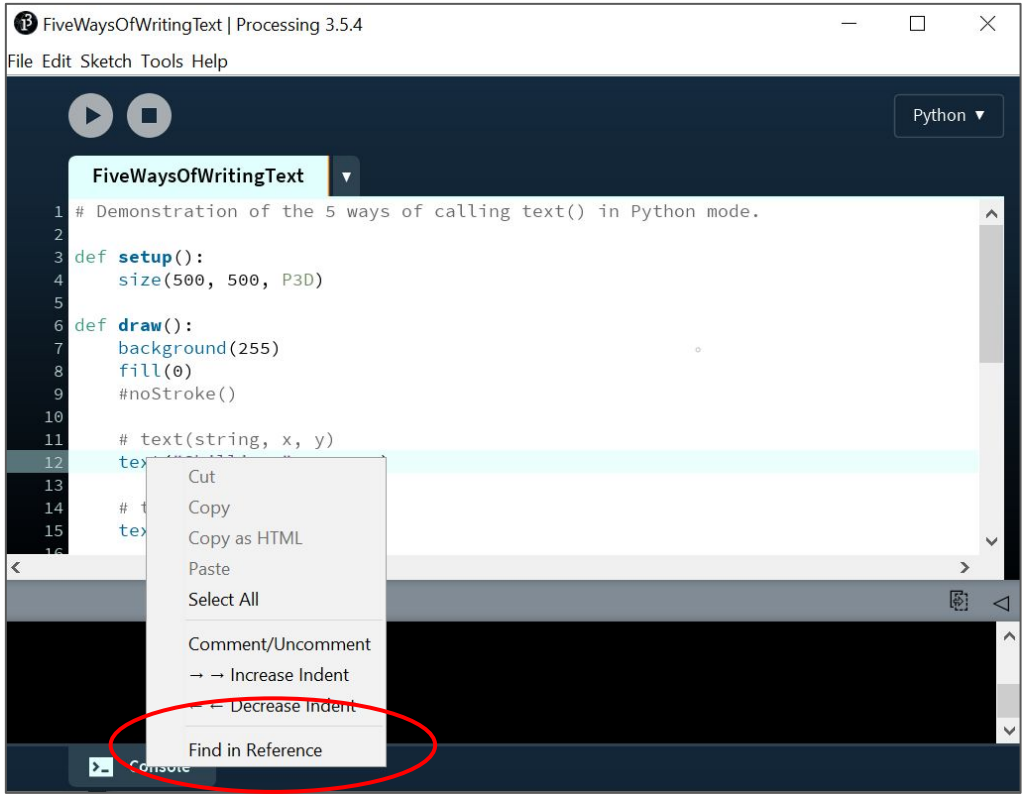
Demo time (quadRendering)

```
"""
Performance demo using quad rendering
"""


def setup():
    size(800, 600, P2D)
    noStroke()
    fill(0, 1)


def draw():
    background(255)
    for i in xrange(500):
        x = random(width)
        y = random(height)
        rect(x, y, 30, 30)

    if frameCount % 10 == 0:
        print frameRate
```

Learning Processing

Read the reference docs!

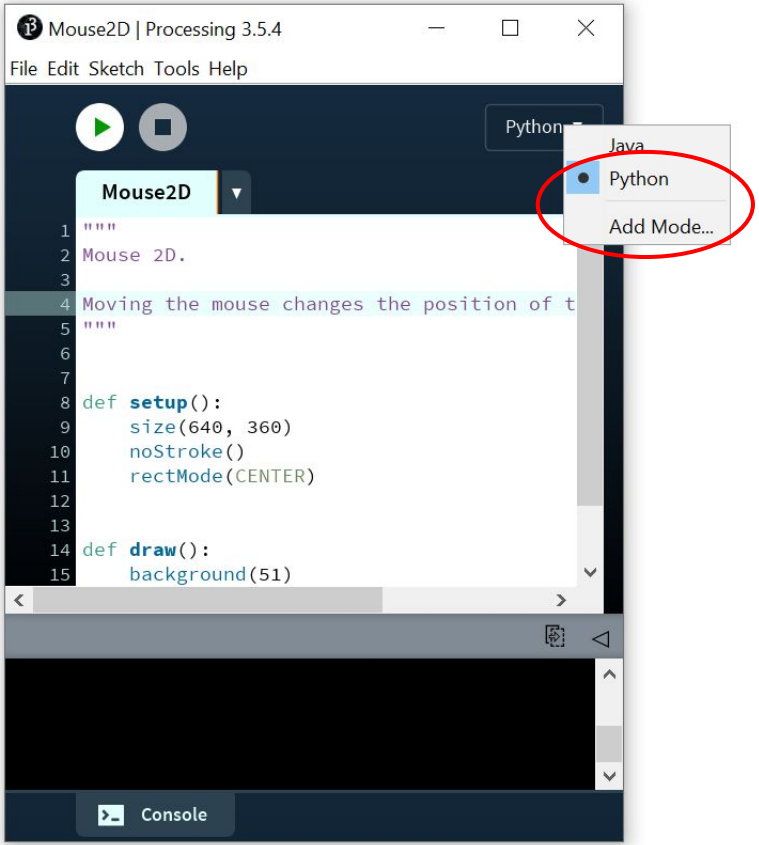
https://py.processing.org/reference/



Processing in Python

Processing was originally written in Java.

Lucky for us, there is also a Python mode

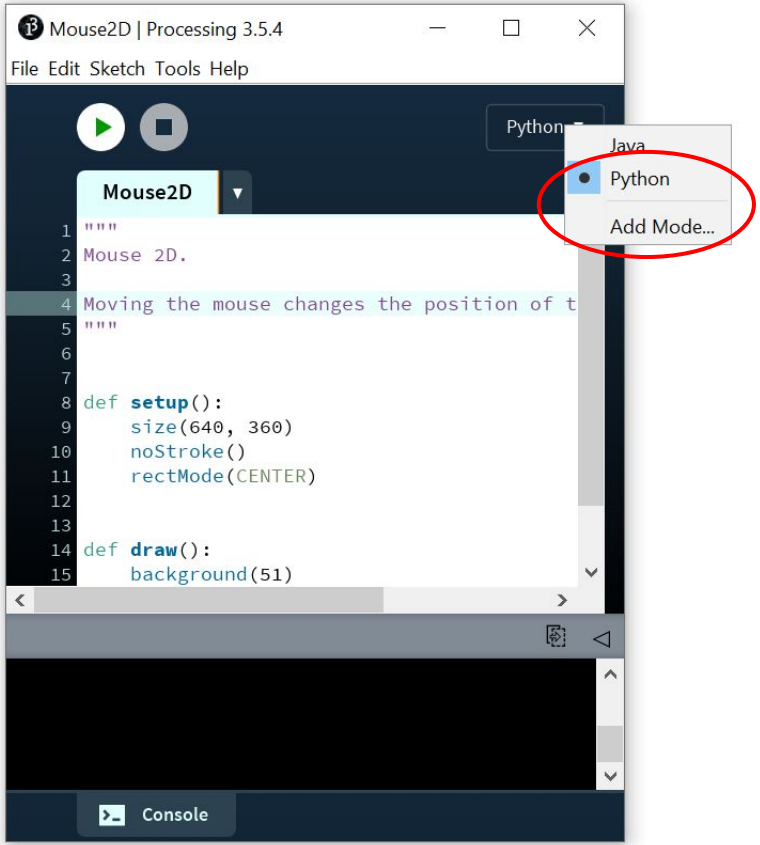


Processing in Python

Processing was originally written in Java.

Lucky for us, there is also a Python mode

Note: there are some limitations



More resources

A tutorial from a ***really*** energetic instructor https://hello.processing.org/

https://py.processing.org/tutorials/

endless code examples: *File > Examples (Ctrl+Shift+o)*

Project 1

Making Art with Processing

This is an open-ended project. You are *required* to be creative.

More details in lab.

Next week lecture

Generative processes to “automatically” create “art”