# Colours, Active Programs, and Coordinate Transforms

UBC Okanagan
Dr. Abdallah Mohamed
Computer Science

# Key Points

1) Color background, shapes, text
2) Control transparency
3) Understand two basic color modes: RGB vs HSB
4) Set color range

# *Color Representation*

- You can use different colors for your drawings and the background.

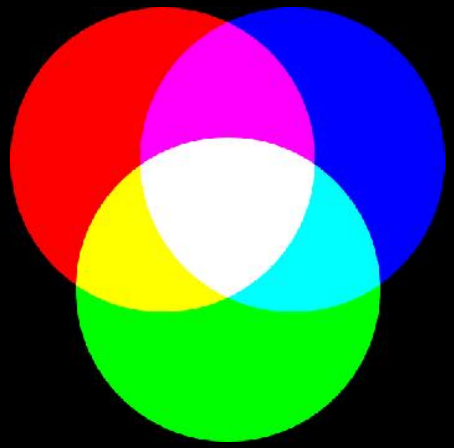
- You have two options:
  - ***Grayscale***: different shades of gray
    - A single digit (integer) ranging from 0 (black) to 255 (white)


  - ***Color*** : to represent a required color using a color model such as RGB or HSB (aka HSV).

# RGB Color Model

- RGB is a color that is a result of mixing three primary colors, Red, Green, and Blue.
  - The amount of each color is represented by a value from 0 (none) to 255 (max).

- Examples:



| | RED Component | GREEN Component | BLUE Component |
|---|---|---|---|
| Red | 255 | 0 | 0 |
| Green | 0 | 255 | 0 |
| Blue | 0 | 0 | 255 |
| White | 255 | 255 | 255 |
| Black | 0 | 0 | 0 |
| Yellow | 255 | 255 | 0 |
| Cyan | 0 | 255 | 255 |
| … | … | … | … |

- Note: when you have same amounts, you get a shade of gray

# Color Question

What is the best description of RGB color (210,0,190)?

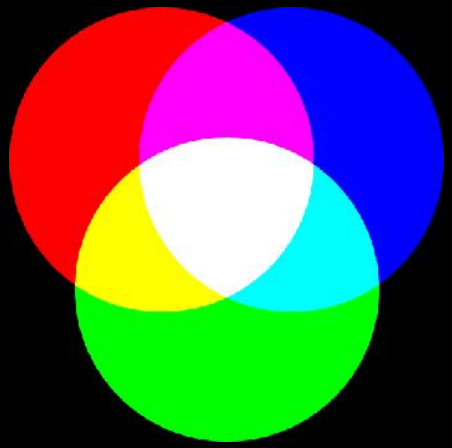
A. a shade of purple

B. a shade of yellow

C. a shade of blue

D. a shade of green

E. a shade of gray



# Color Question

What is the best description of RGB color (120,120,120)?

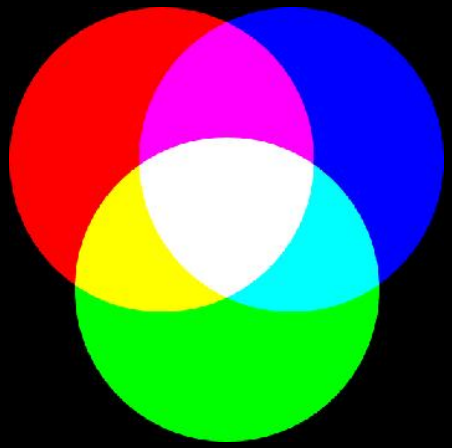
A. a shade of purple

B. a shade of yellow

C. a shade of blue

D. a shade of green

E. a shade of gray



# *How to Color?*

- You can color the following items:
  - ***background*** using `background()` function.
  - ***outline*** and ***fill*** of a shape using `stroke()` and `fill()` before drawing the shape.
- Use either
  - one argument for gray shades. e.g. `fill(0)` is *black* fill.
  - three arguments for RGB color. e.g. `fill(255,0,0)` is *red* fill
    - Note that RGB mode is used by default.
- Once you set a color, it applies to *all shapes drawn afterwards*.
- ***Default values*** are used if no colors are chosen.
  - ***background***: 204 (light gray), ***stroke***: 0(black), `fill`: 255 (white).
- You can use `noFill()` or `noStroke()` functions to disable filling or outlining a shape.

# *List of color functions so far…*

- `background()`
  - Set background color
- `stroke(), noStroke()`
  - Set stroke (line) color
- `fill(), noFill()`
  - Set filling or text color

# Colourful Shapes

```
background(128);

strokeWeight(3);

fill(255);
rect(0,0,90,40);

stroke(255, 0, 0);
fill(0);
rect(10,20,30,30);

stroke(255, 0, 255);
fill(255, 0, 0);
rect(50,20,30,30);
```

**Q:** Can you link each statement to one of the output shapes?

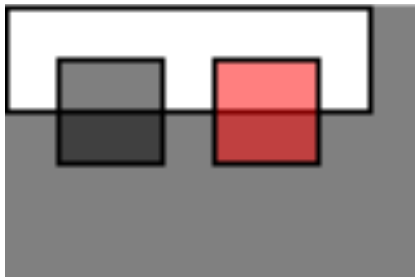
# Colourful Text

```
background(0);
size(140,120);

textAlign(CENTER);
textSize(28);
text("UBC", 70, 30);

textSize(18);
text("Okanagan", 70, 50);

fill(255,255,0);
textSize(12);
text("Computer Science", 70, 70);

fill(0,255,0);
textSize(10);
text("1177 Research Rd, Kelowna, BC V1V 1V7", 10,85,120,40);
```

# *Colour Transparency*

- An ***optional argument*** can be used for `fill()` and `stroke()` to ***control transparency***.
  - 0 completely transparent i.e. 0% opacity
  - 255 completely opaque i.e. 100% opacity
- Examples:
  - `fill(255)` is opaque white filling (default opacity is 100%)
  - `fill(0, 128)` is semi-transparent black filling
  - `fill(255, 0, 0, 128)` is semi-transparent red filling

```
background(128);
fill(255);
rect(0,0,70,20);
fill(0, 128);
rect(10,10,20,20);
fill(255, 0, 0, 128);
rect(40,10,20,20);
```



# Using Colours

What will be drawn on the screen?

A. A line, rectangle, and an ellipse

B. A rectangle and an ellipse

C. Only the ellipse

D. Nothing

E. This code has an error and won't run.

```
noStroke();
line(30,30,50,30);
noFill();
rect(10,10,20,20);
stroke(255,0);
ellipse(50,50,20,20);
```

# Using Colours

These two statements are exactly the same.

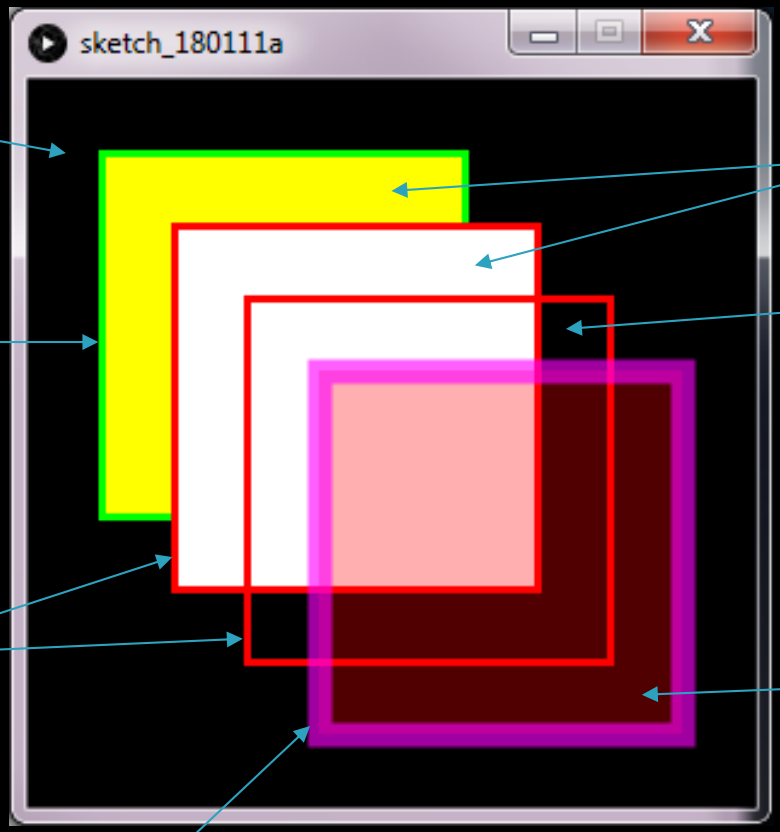
```
fill(255,255,255);
fill(255,255);
```

A. True

B. False

# Use Colours!

- Write a code to create the following sketch +2 points



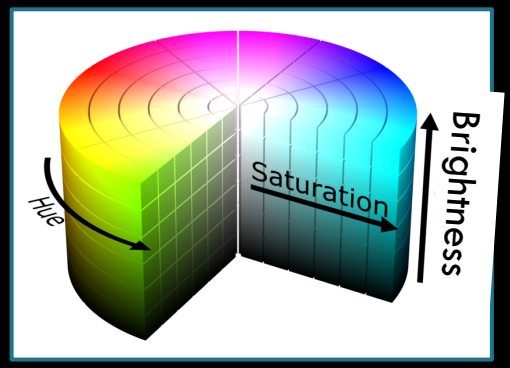
# *Aside: Hexadecimal Notation*

- RGB colours can be represented using Hexadecimal notation.
  - Syntax: #RRGGBB
    - The # denotes the hex notation
    - RR is a two-digit hex number representing red value from 0 to 255
    - GG is a two-digit hex number representing green value from 0 to 255
    - BB is a two-digit hex number representing blue value from 0 to 255
- Examples:
  - `fill(255,255,255)` equivalent to `fill(#FFFFFF)`
  - `fill(128,196,64)` equivalent to `fill(#80C440)`
  - `fill(0,0,255)` equivalent to `fill(#0000FF)`

# HSB Colour Model

- **In this mode, a colour is represented by three components**
  - ***Hue***
    - Dominant pure color.


  - ***Saturation***:
    - Vibrancy of the color
    - Range: 0 to 100
  - ***Brightness***
    - How bright the color is.
    - Range: 0 to 100



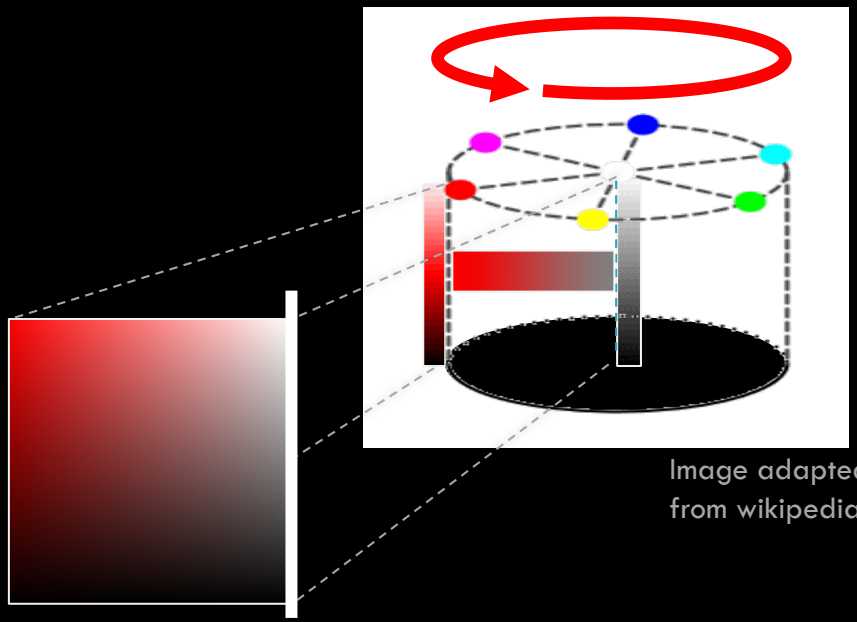



Image adapted from wikipedia

A plane with all S and B values for Hue=0 (red)

# Colour Question

What is the best description of HSB colour (350,90,95)?

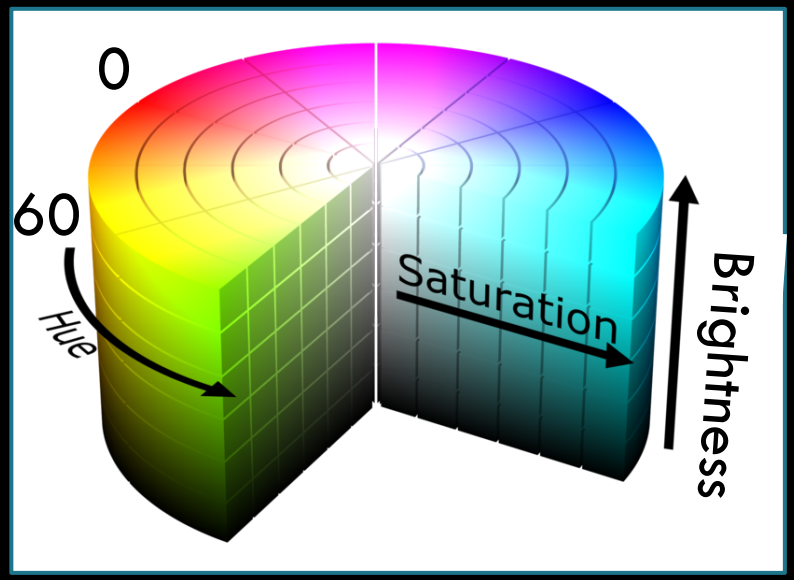
A. a shade of red

B. a shade of blue

C. Black

D. White

E. One of the ranges is invalid




***Note***: Assume the ranges (0-360,0-100,0-100)

# Colour Question

What is the best description of ***HSB*** colour (300,0, 50)?

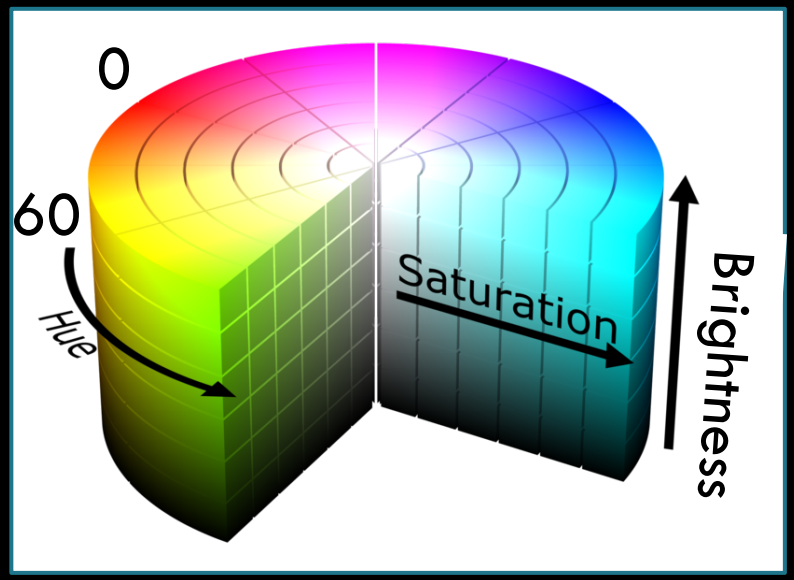
A. a shade of red

B. a shade of blue

C. a shade of gray

D. black

E. One of the ranges is invalid




***Note***: Assume the ranges (0-360,0-100,0-100)

# Colour Question

What is the best description of HSB colour (300, 99, 0)?

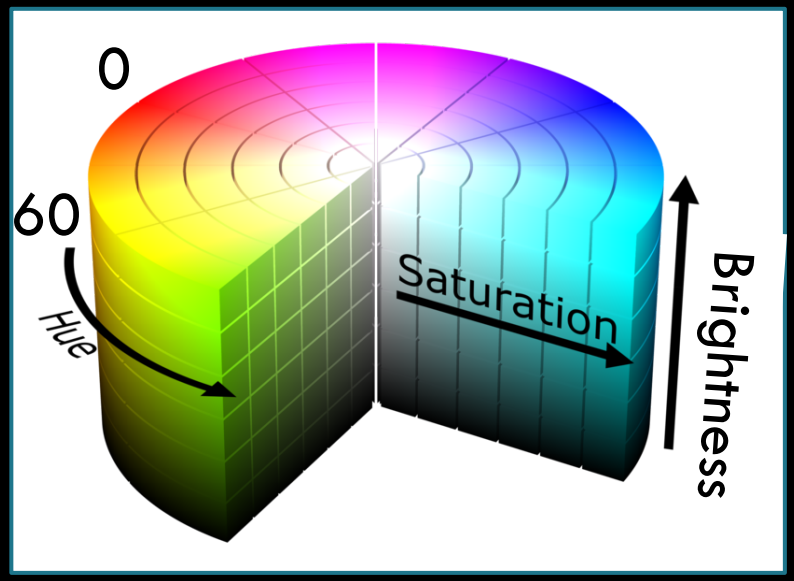
A. a shade of red

B. a shade of blue

C. a shade of gray

D. black

E. One of the ranges is invalid




***Note***: Assume the ranges (0-360,0-100,0-100)

# Colour Question

What is the best description of HSB colour (200,0,150)?

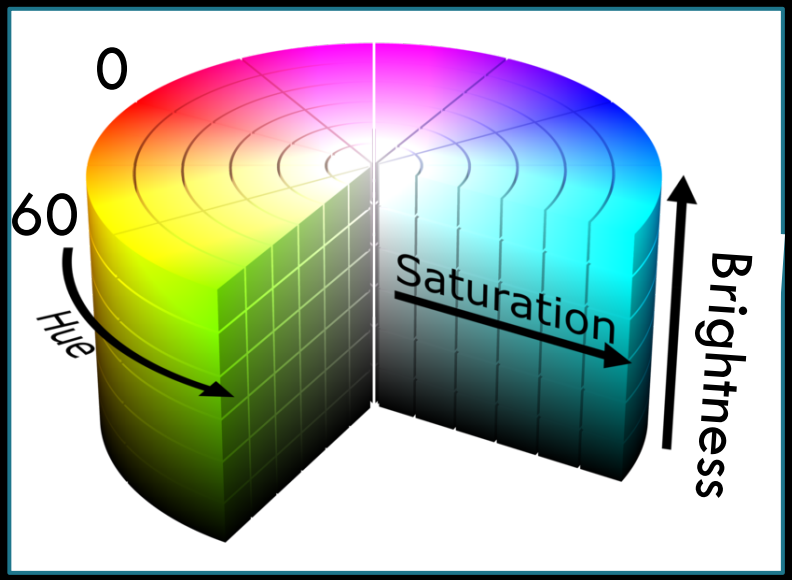
A. a shade of red

B. a shade of blue

C. Black

D. White

E. One of the ranges is invalid




***Note***: Assume the ranges (0-360,0-100,0-100)

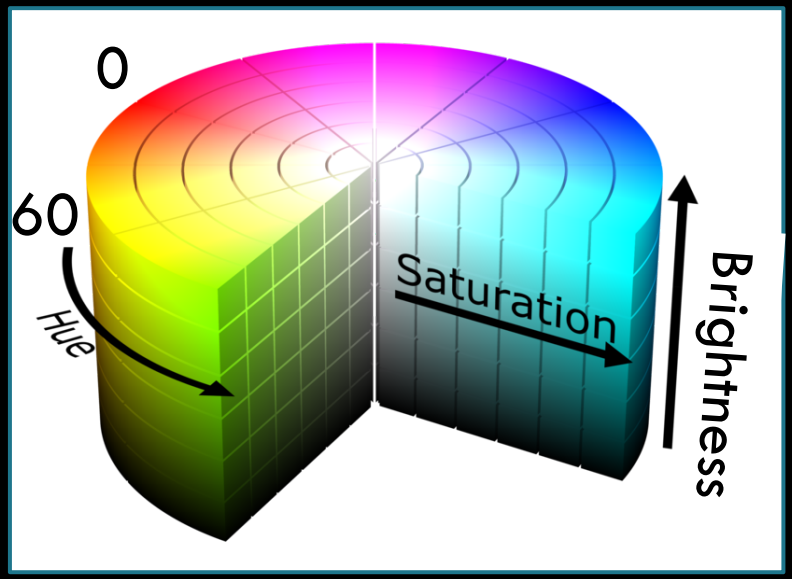
# Colour Question

These two HSB colours look the same on the screen:

(200,0,50) and (50,0,50)

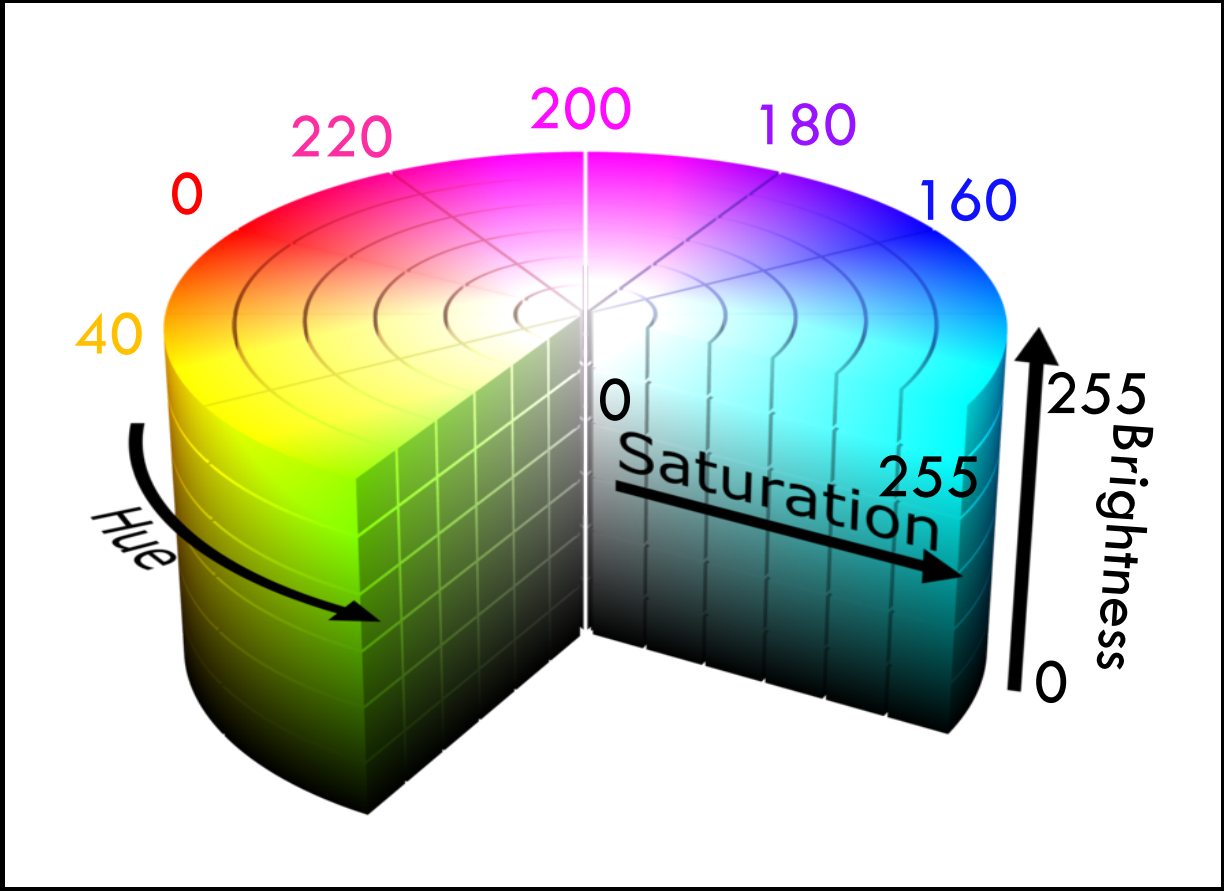
A. True

B. False



# *HSB Ranges in Processing*

- While above ranges (i.e. 360,100,100) are standard in image processing, the ***Processing language*** uses **255,255,255** by default.



# Changing the Color Mode

- By default, processing uses RGB mode with ranges from 0 to 255 for all color components R, G, and B.
- Defaults can be changed using `colorMode()` function.
- Syntax:

```
colorMode(mode)
colorMode(mode, max)
colorMode(mode, max1, max2, max3)
colorMode(mode, max1, max2, max3, maxA)
```

- Examples:
  - `colorMode(RGB)` RGB mode, use **default** ranges (0 to 255)
  - `colorMode(RGB,100)` RGB mode, ranges: 0 to 100 for all colors
  - `colorMode(HSB)` HSB mode, default ranges 0 to 255)
  - `colorMode(HSB,360,100,100)` change defaults to 360,100,100
  - `colorMode(HSB,1)` HSB, ranges 0 to 1.0 for all components
  - `colorMode(HSB,1,1,1,10)` same as above, opacity is 0 to 10.

# *Changing the Colour Mode, cont'd*

**Be careful**:

After changing the ranges with any of the statements above, those ranges will remain in use until they are explicitly changed again.

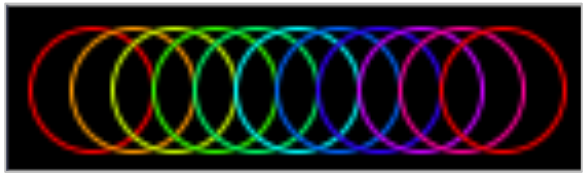
```
colorMode(RGB, 100, 100, 100);  //ranges are 100 for all R,G,B components
colorMode(HSB);                 //ranges are still 100 for all H,S,B components
```

# *List of colour functions we learned today*

- `background()`
  - Set background colour
- `stroke(), noStroke()`
  - Set stroke (line) colour and transparency
- `fill(), noFill()`
  - Set filling or text color and transparency
- `colorMode()`
  - Choose between RGB and HSB, and optionally set the range

# Examples

```
size(140, 40);
background(0);
noFill();
colourMode(HSB, 100);
for (int i = 0; i <= 100; i+=10) {
  stroke(i, 100, 100); //only change the hue in every iteration
  ellipse(i+20, 20, 30, 30);
}
```

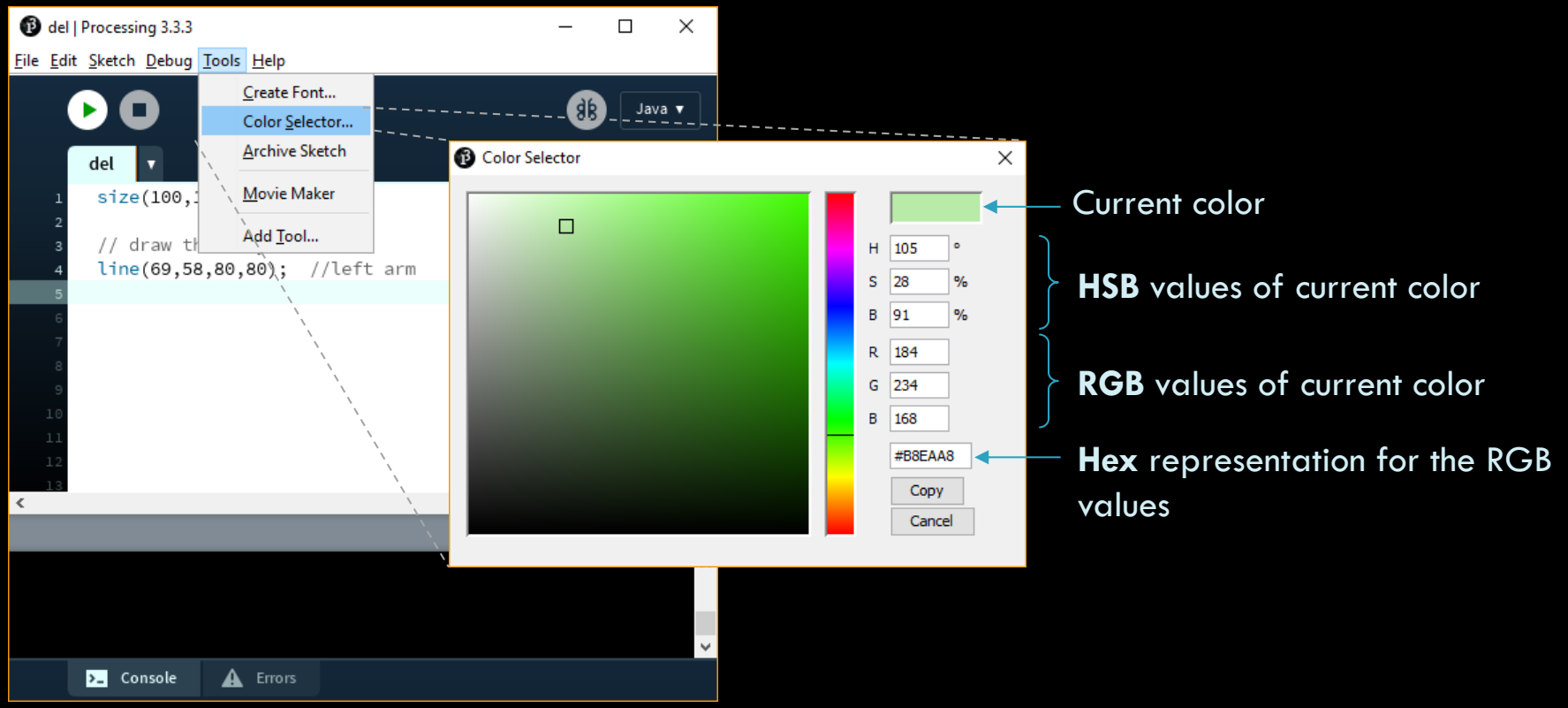


```
// this example is from Processing Documentation
colourMode(RGB, 100);
for (int i = 0; i < 100; i++)
  for (int j = 0; j < 100; j++) {
    stroke(i, j, 0);
    point(i, j);
  }
```

**Note**: don't worry so much if you don't remember for loops. We will go over it later.

# PDE Colour Selector Tool

- You can use the PDE colour selector tool from the tools menu (`Tools->Color Selector…`) to get the values of your chosen color.



# *Update Your Design*

- Add code to your character that you designed previously so that it has colours now ☺… remember, be creative with your colors and shapes.
- Here is my design, but yours should be different

# Active Programs

UBC Okanagan
Dr. Abdallah Mohamed
Computer Science

# Objectives

- This is a  pre-class reading materials.
- After finishing reading the materials, you should be able to:
  - Understand the difference between ***static*** and ***active*** modes.
  - Understand the order of execution of active sketches.
  - Create a simple animation using `setup()` and `draw()`.
  - Set the frame rate of an animation using `frameRate()`
  - Know where to place `size()` and `background()`
  - Use the system variables: `mouseX`, `mouseY`, `width`, `height`
  - Stop an animation using `noLoop()`

# *Static vs Active Modes*

- All programs you have been writing so far are ***static*** sketches.
  - A ***static sketch*** is a series of statements that aim to draw a ***single image***; i.e.no animation or interaction.

- An ***active*** sketch on the other hand aims to draw a ***series of images*** (each called a ***frame***) that represent an ***animation***.

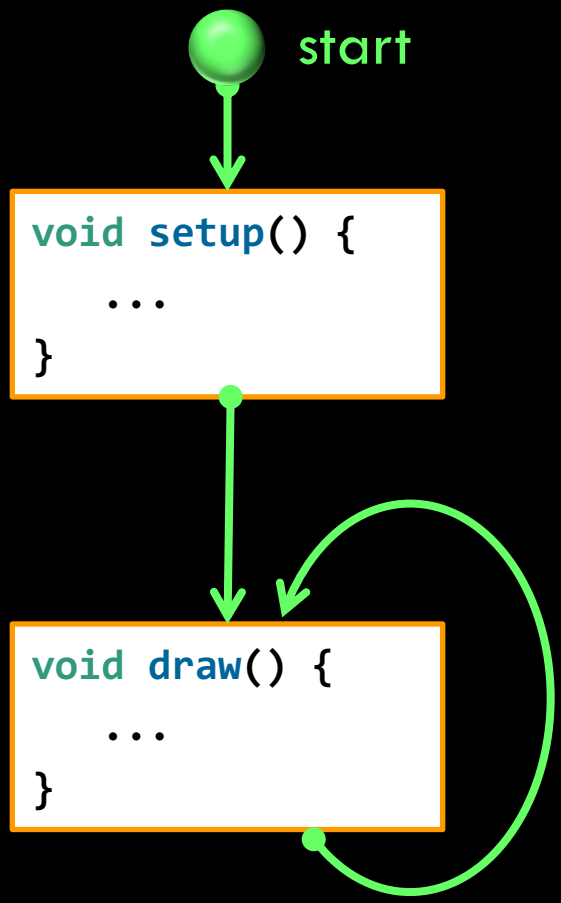
- Active sketches may be programmed to be ***interactive*** to user's actions.
  - Examples of actions: mouse movement, keyboard presses, etc.

# How to Create Active Sketches

- Two ***built-in*** functions: `setup()` and `draw()` are always ***called automatically***.
  - `setup()` runs once **at the beginning**
  - Then `draw()` runs repeatedly.

- The rate of running the `draw` method is called the ***framerate***.
  - The default is **60 fps**, but it can be changed using the `frameRate()` function.

- You can ***stop*** repeating `draw()` using `noLoop()` function.



# Active Program Structure

This part ***runs once*** and is used for ***initialization***

This part ***loops forever*** and is used for ***animation***

```
void setup() {
  // Step S1
  // Step S2
  // ...
  // Step Sn
}

void draw() {
  // Step D1
  // Step D2
  // ...
  // Step Dn
}
```

The two curly brackets { } are used to define the beginning and end of a block of code

**Order of execution:** $S_1$, $S_2$, .., $S_n$, **$D_1$**, $D_2$, .., $D_n$, **$D_1$**, $D_2$, .., $D_n$, **$D_1$**, ...etc

# Drawing a Static Sketch with setup/draw

- All these four programs produce the same output
  - *Justify?*

**without setup/draw**

```
size(200,200);
background(255);
rect(10,10,40,40);
```

**three different ways with setup/draw**

```
void setup(){
  size(200,200);
}
void draw(){
  background(255);
  rect(10,10,40,40);
}
```

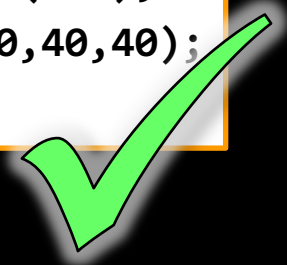
```
void setup(){
  size(200,200);
  background(255);
}
void draw(){
  rect(10,10,40,40);
}
```

```
void setup(){
  size(200,200);
  background(255);
  rect(10,10,40,40);
}
```

# Notes About Active Sketches

- You ***can't*** mix static and active modes!
  - Once you use active mode, you ***can't*** call any function, such as `rect()`, outside `setup()` and `draw()`.
- `size()` *can only be executed once*
  - *so it can't be part of draw()*

```
void setup(){
  size(200,200);
}
void draw(){
  background(255);
  rect(10,10,40,40);
}
```



mix static & active

```
rect(10,10,40,40);
void setup(){
  size(200,200);
}
void draw(){
  background(255);
}
```

wrong place for size()

```
void setup(){
}
void draw(){
  size(200,200);
  background(255);
  rect(10,10,40,40);
}
```

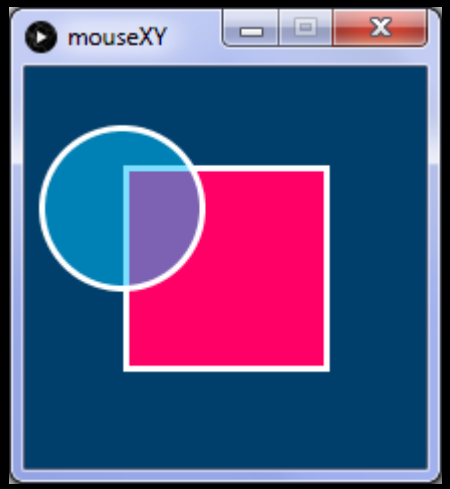
# *Mouse Location*

- Processing has two keywords (*system variables*) that will always contain the current coordinates of the mouse cursor.
  - `mouseX` and `mouseY` contain mouse location (x,y) in current frame.
  - ***Default value*** is 0 for both variables.

# A Shape Following the Mouse

- In this example, the ball follows the mouse position.

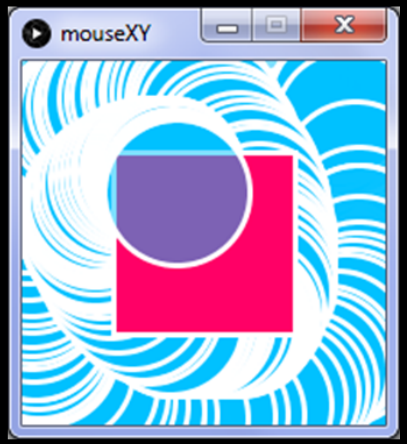
```
void setup() {
  size(200, 200);
  stroke(255);
  strokeWeight(3);
}
void draw() {
  background(0,63,107);
  fill(255,0,102);
  rect(50,50,100,100);
  fill(0,192,255,130);
  ellipse(mouseX, mouseY, 80, 80);
}
```



# Where to put background()

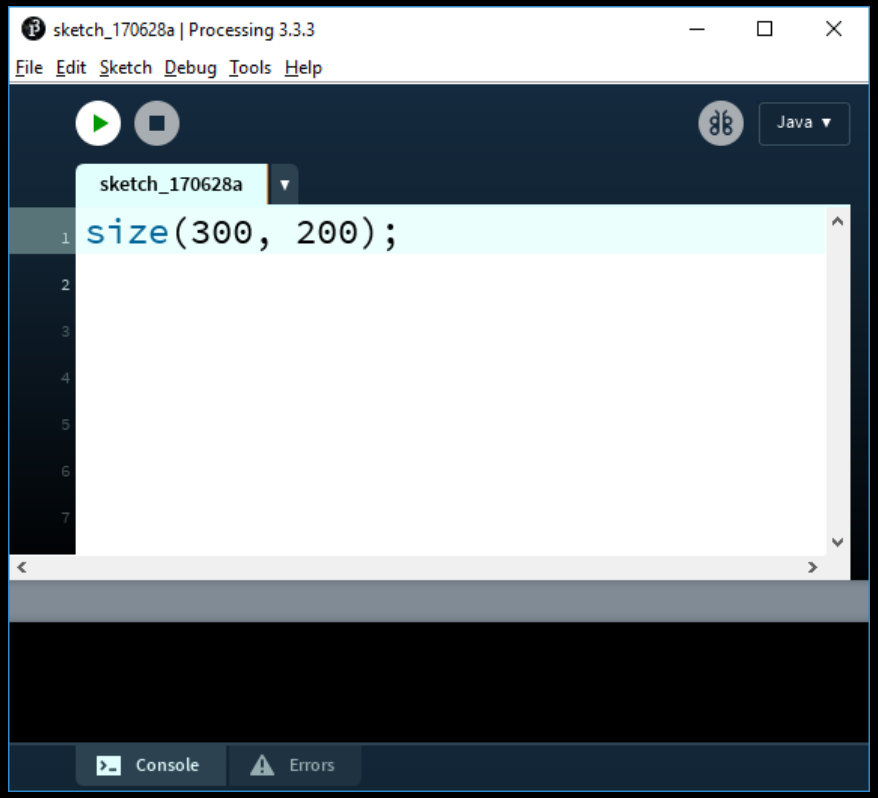
- *If placed in draw()*, it clears the sketch at beginning of every frame
  - i.e. it flood the sketch with some color.
- *If placed in setup()*, it sets the background of first frame only and doesn't clear subsequent frames.
- If you move `background()` to **`setup()`**, this would be the output from the previous example.

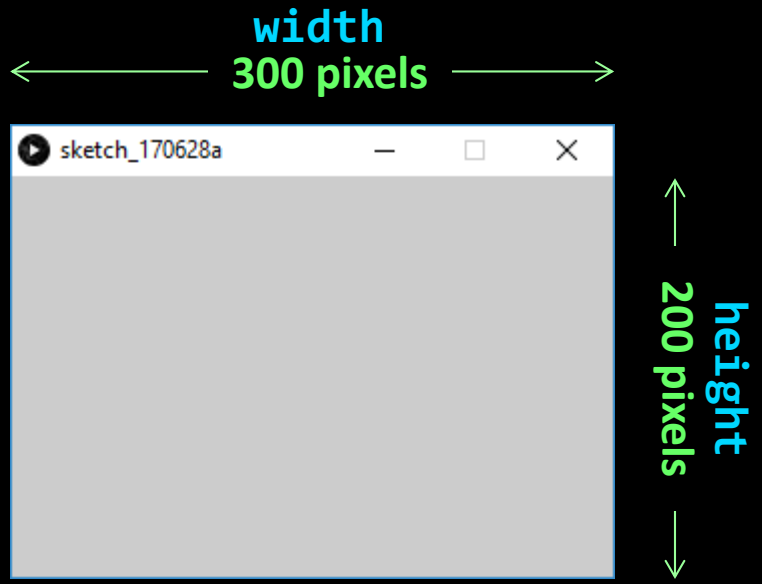
```
void setup() {
  size(200, 200);
  stroke(255);
  strokeWeight(3);
  background(0,63,107);
}
void draw() {
  fill(255,0,102);
  rect(50,50,100,100);
  fill(0,192,255,130);
  ellipse(mouseX, mouseY, 80, 80);
}
```



# Window's width and height

- There are to more useful system variables: `width` and `height` that contain the size of the current display window.
  - We set these two values using the `size()` function.
  - Default value is 100 if `size()` is not used.

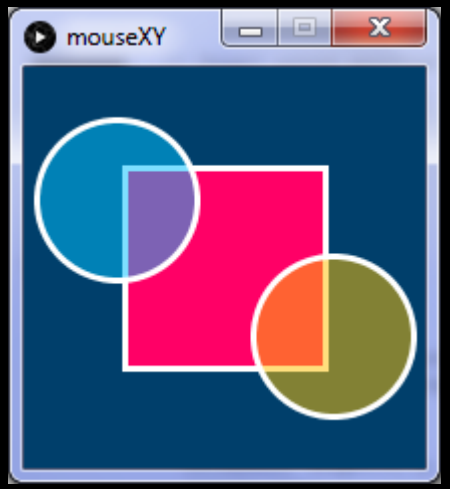


# Two Shapes Controlled by the Mouse

In this example, we create a second ball positioned at the inverse of the cursor position.

```
void setup() {
  size(200, 200);
  stroke(255);
  strokeWeight(3);
}

void draw() {
  background(0,63,107);
  fill(255,0,102);
  rect(50,50,100,100);
  fill(0,192,255,130);
  ellipse(mouseX, mouseY, 80, 80);
  fill(255,192,0,130);
  ellipse(width-mouseX, height-mouseY,80,80);
}
```



# Function Automatically Called

Which of these functions is *automatically called* by the system once we run the program?

A. size(200,200);

B. setup() and draw()

C. noLoop()

D. rect(0,0,width,height);

E. ellipse(0,0,width,height);

Question

# Frame Rate

The default frame rate is _____ and it can be changed using the function _____

A. 15, frameRate()

B. 60, frameRate()

C. 15, setFrameRate()

D. 60, setFrameRate()

E. None of the above

Question

# Where to write code?

Which code is valid?

A.

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

B.

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

C.

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

D. None of the above.

# Where to write code?

Which code clears a the display window at the beginning of each frame?

A.

```
void setup(){
  size(200,200);
}
void draw(){
  background(255);
  rect(5,5,90,90);
}
```

B.

```
void setup(){
  size(200,200);
  background(255);
}
void draw(){
  rect(5,5,90,90);
}
```

C.

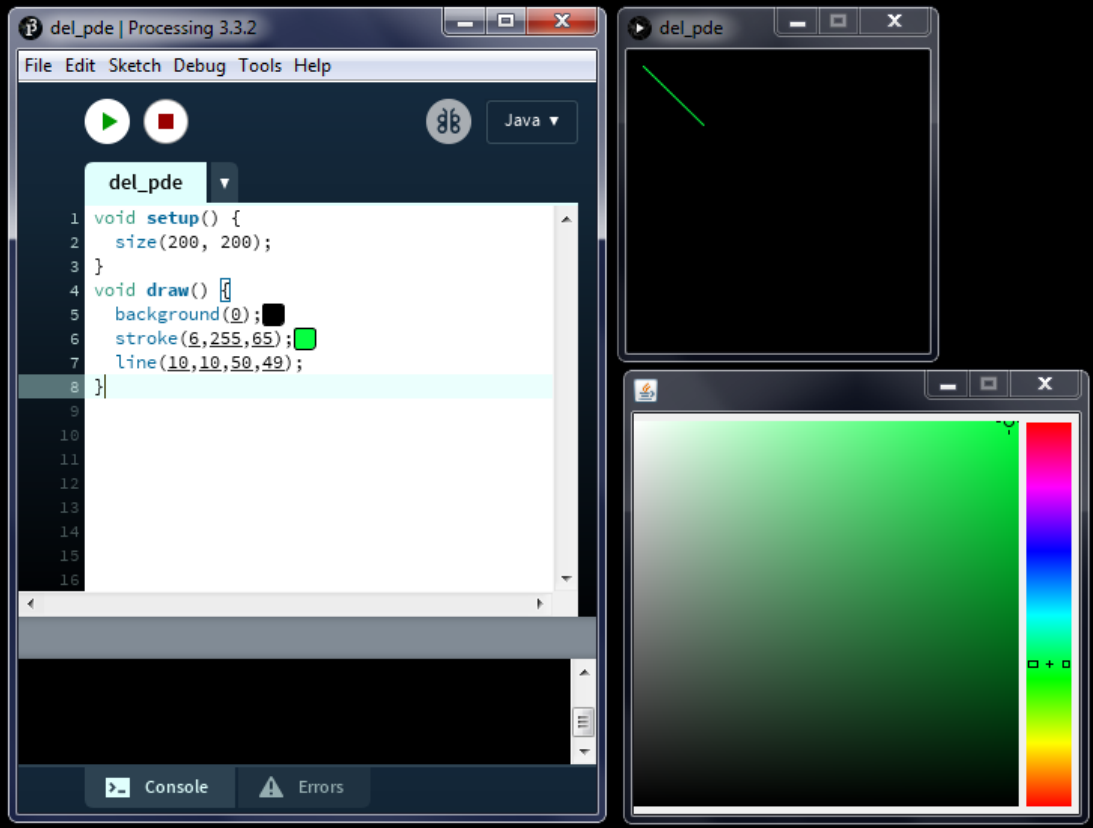
```
background(255);
void setup(){
  size(200,200);
}
void draw(){
  rect(5,5,90,90);
}
```

D. None of the above; I have a better answer.

***Question:*** *what is the difference between A, B, and C?*

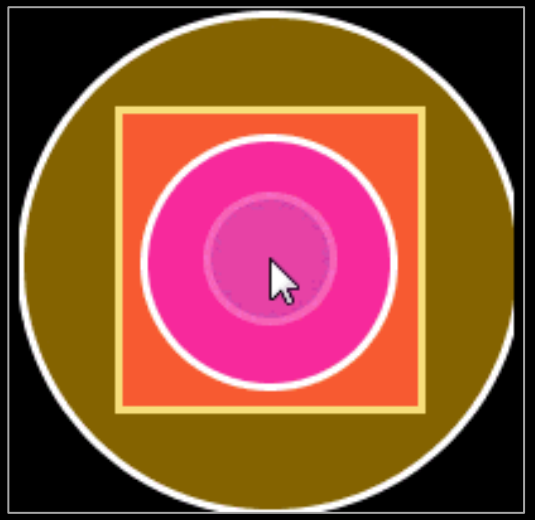
# *Tweaking Your Sketch At the Runtime*

- Tweak Mode (**`Sketch-> Tweak`**) runs the code so that you can change some color and variable values while the code is running and see instant feedback.
- Notes:
  - This only applies to ***active mode***.
  - You need to save your program before you can tweak it.



# Animation based on Mouse Location

- Build on the code in the pre-class materials and use the mouse coordinates (`mouseX`, `mouseY`) to control other attributes in the animation, e.g. size, transparency, color, background, etc.
- Be creative! For example, in this animation →
  - I added a third circle, then had the size of each shape change differently:
    - Circle1: radius = **`mouseX + mouseY`**
    - Circle2: radius = **`mouseX/2`**
    - Circle3: radius = **`mouseY*2`**
    - `Box:` size depends on mouseX and mouseY
  - Controlled shapes' location with mouse location.
  - Changed the background color based on the combined size of all circle.



+3 points

- Your interactive animation doesn't have to have any purpose for now, just try to make it look cool and have fun ☺

# *Moving YOUR Character*

- Referring to the character you designed previously, add code to your program so that the character moves with your mouse cursor.
- Hint: the location of all shapes of your character should depend on mouseX and mouseY
- Here is the output (Your character could be different):

+3 points

Computer Creativity

*Pre-Class Reading*

# *Active Programs (2)*

UBC Okanagan

Dr. Abdallah Mohamed
Computer Science

# Objectives

- This is a  pre-class reading materials. After reading them, you should be able to:
  - Use mouse location from previous frame (`pmouseX`, `pmouseY`)
  - Generate random numbers using `random()`
  - Write programs that are driven by mouse and key events
    - 1) Using mouse functions:

      `mousePressed()`, `mouseReleased()`, `mouseClicked()`, `mouseMoved()`, `mouseDragged()`
    - 2) Using key functions:

      `keyPressed()`, `keyReleased()`

# *Mouse Location… revisited!*

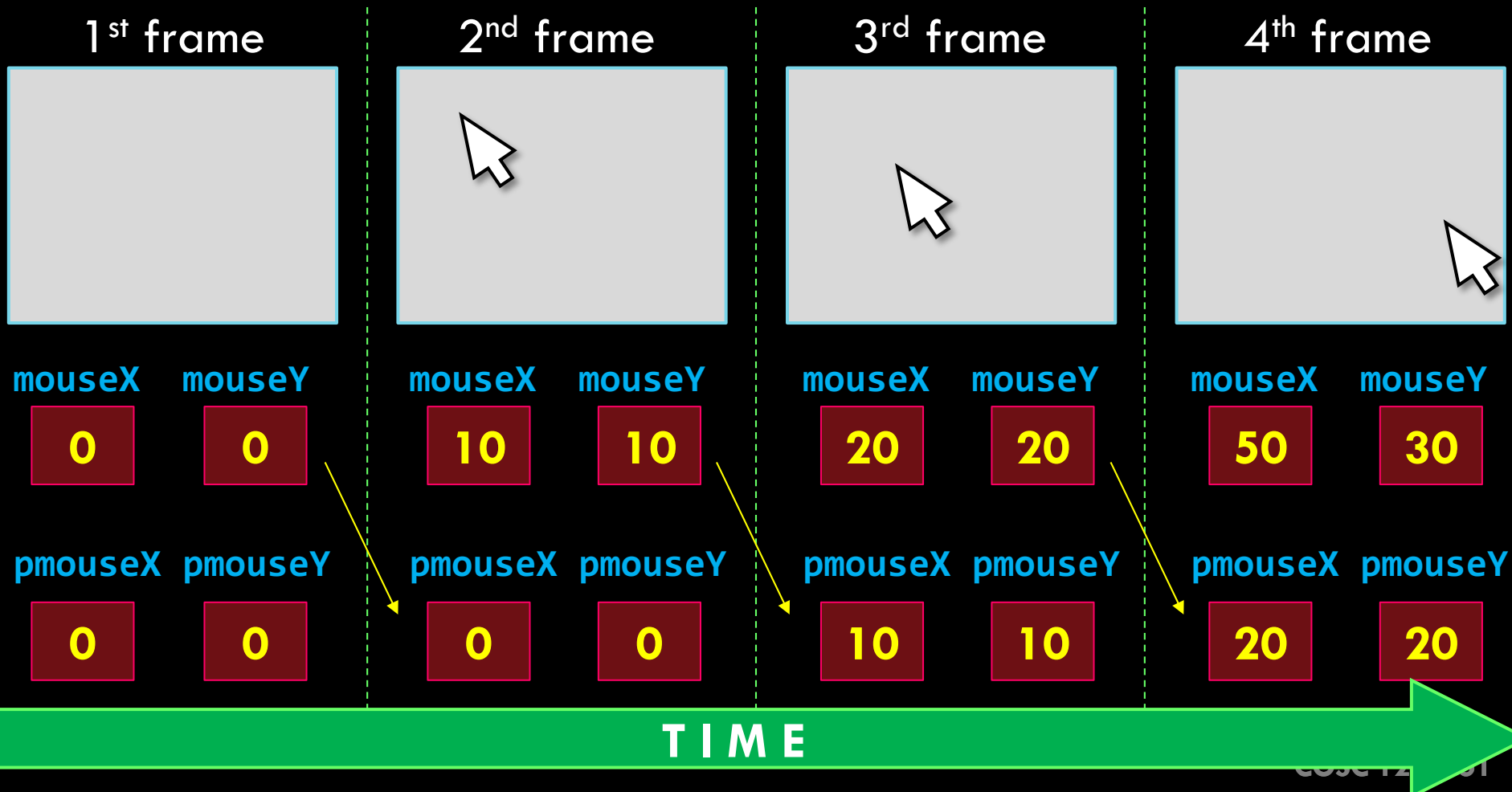
- You have seen before that Processing has two *system variables* that hold the current coordinates of the mouse cursor
  - `mouseX` and `mouseY` contain mouse location (x,y) in current frame.

- Furthermore, processing has two more system variables that will always hold the previous coordinates of the mouse cursor.
  - `pmouseX` and `pmouseY` contain (x,y) from the frame previous to the current frame (if used inside the `draw()` function).

- ***Default value*** is `0` for all four variables.

# pmouseX and pmouseY

- You can use pmouseX and pmouseY whenever you want to use the mouse location in the previous frame.



# pmouseX and pmouseY

- We can use previous mouse coordinates is to draw a ***continuous line***.
- Note where we placed `background`.

```
void setup() {
  size(200, 200);
  background(255); // don't clear previous frame
}
void draw() {
  line(pmouseX, pmouseY, mouseX, mouseY );
}
```


- Task: change the ***framerate*** to 4 fps and check the output

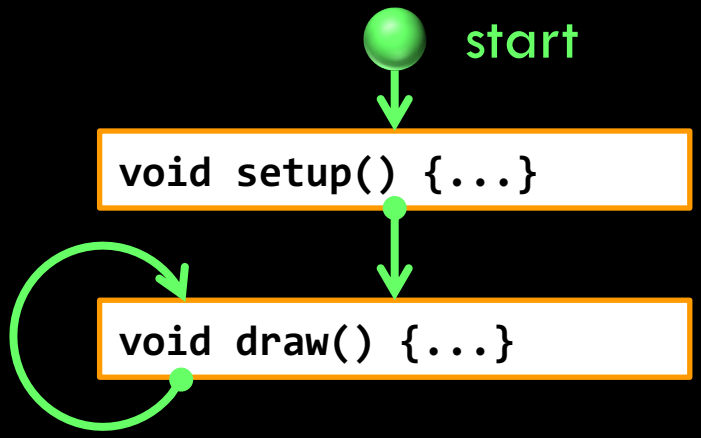
# *Mouse and Key Events*

- While `setup()` & `draw()` are ***always*** invoked automatically, there are functions that are invoked ***based on users input.***

- Functions that are called based MOUSE events:
  - `mousePressed()`: called whenever a mouse button is clicked
  - `mouseReleased()`: called whenever a mouse button is released
  - `mouseClicked()`: called after a mouse button is pressed then released.
  - `mouseMoved()`: called whenever the mouse moves and the mouse button ***is not*** clicked
  - `mouseDragged()`: called whenever the mouse moves and the mouse button ***is*** clicked

- Functions that are called based KEY events:
  - `keyPressed()`: called whenever a key is pressed.
  - `keyReleased()`: called whenever a key is released.

# Overall Structure of Active Programs




setup runs once

***then***

draw runs many times per second

```
void mouseClicked(){...}
void mousePressed() {...}
void mouseReleased(){...}
void mouseDragged() {...}
void mouseMoved(){...}
```

mouse functions are ***automatically*** called when a mouse event happens

```
void keyPressed() {...}
void keyReleased() {...}
```

Key functions are ***automatically*** called only when a key event happens

# Event Driven Program

- In this example, a new circle is drawn ***wherever the mouse is clicked***. The color of the circle is random.
- Also, ***whenever a key is pressed***, the sketch is cleared!

```
void setup() {
  size(200, 200);
  colorMode(HSB,360,100,100);     //HSB mode is used
  background(360,0,100);          //white background
  noStroke();
}
void draw() {// nothing here}

void mousePressed() {
  fill(random(360),100,100,128); //random color
  ellipse(mouseX, mouseY, 40, 40);
}
void keyPressed() {
  background(360,0,100);          //clear sketch
}
```

***Question***: what happens if we add background() to draw()?

Computer Creativity

# Active Programs (2)

UBC Okanagan

Dr. Abdallah Mohamed
Computer Science

# *The Pre-Class Reading*

- The pre-class materials covered the following:
  - ***New*** keywords: `pmouseX`, `pmouseY`
  - ***New*** functions : `random()`
  - ***New*** event-driven functions
    - automatically invoked based on MOUSE events:

      `mousePressed()`, `mouseReleased()`, `mouseClicked()`,

      `mouseMoved()`, `mouseDragged()`
    - automatically invoked based on KEY events:

      `keyPressed()`, `keyReleased()`

# Mouse Location

Which of the following keeps track of mouse location from previous frame?

A. mouseX , mouseY

B. pmouseX , pmouseY

C. pFrame.x, pFrame.y

D. pFrame.mouseX, pFrame.mouseY

E. none of the above

# Drawing a continuous line

Which of the following can be used to draw a continuous line?

A.
```
void setup(){  background(255); }
void draw() {
  line(pmouseX, pmouseY , mouseX, mouseY);
}
```

B.
```
void setup(){  background(255); }
void draw() {
  line(mouseX, mouseY, pmouseX, pmouseY );
}
```

C.
```
void setup(){ }
void draw() {
  background(255);
  line(mouseX, mouseY, pmouseX, pmouseY );
}
```

D. *Either A or B*

E. *All of them*

Question

# Event Based Programming

Which of these functions is automatically called whenever the user presses the mouse button and moves the mouse at the same time.

A. mouseReleased

B. mousePressed

C. mouseDragged

D. mouseMoved

E. Both B and C

# Framerate

Which framerate has most probably produced this output?

A. 60

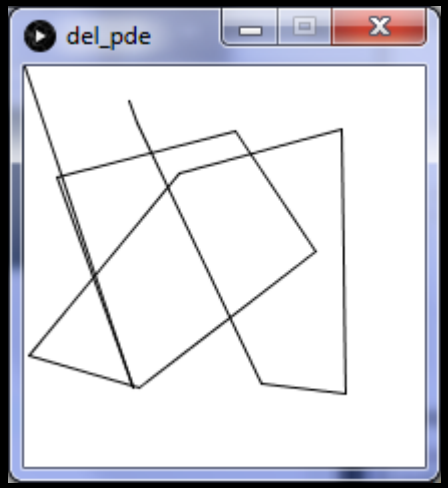
B. 45

C. 30

D. 25

E. 5

```
void setup() {
  size(200, 200);
  background(255);
  stroke(0);
  framerate(????);
}
void draw() {
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

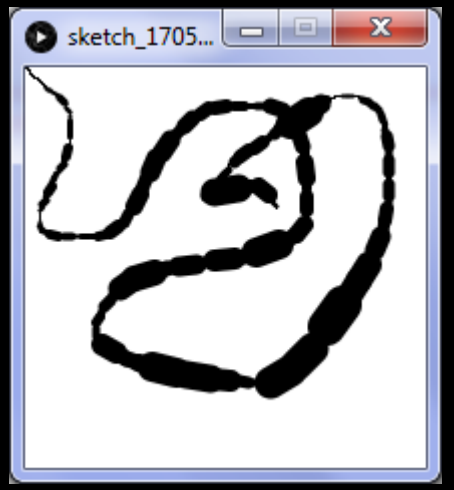


# Mouse Speed

- this code is from the pre-class readings and is used to draw a continuous line.

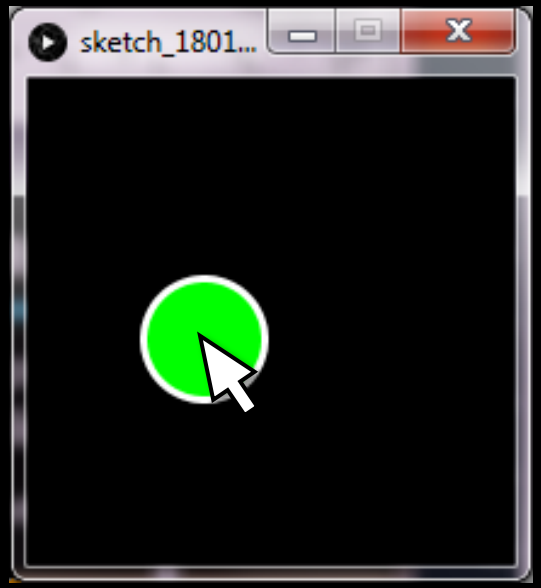
```
void setup() {
  size(200, 200);
  background(255);
  stroke(0);}
void draw() {
  //... add code here ...
  line(pmouseX, pmouseY, mouseX, mouseY );
}
```

- Modify the code so that the thickness of the line is controlled by the ***mouse speed***. Here are some hints:
  - Mouse Speed is the distance the mouse travel per unit of time. Therefore, speed can be computed in terms of the distance the mouse travels in each new frame.
    - i.e. difference between current mouse position and previous one
  - Use `abs()` function to avoid negative values.
  - Don't worry too much about having accurate calculations.

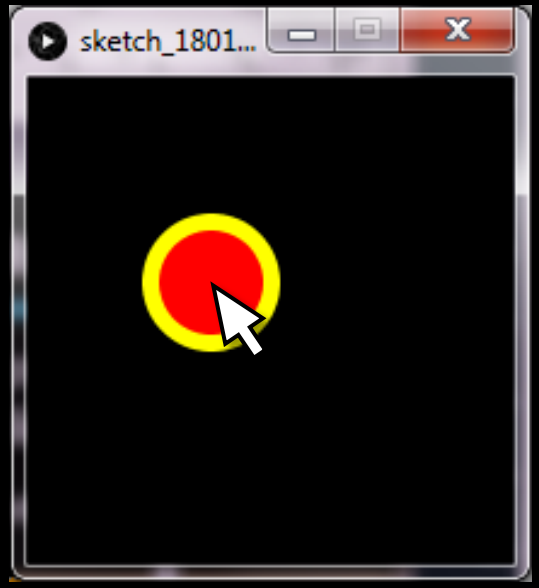


# Mouse Events

- Create a program that draws a circle which follows the mouse (same location as the mouse)
- The circle should be:
  - red with thick, yellow outline as long as the mouse is pressed.
  - green with thin, white outline as long as the mouse is not pressed.
- **Don't** use variables or **conditional** statements



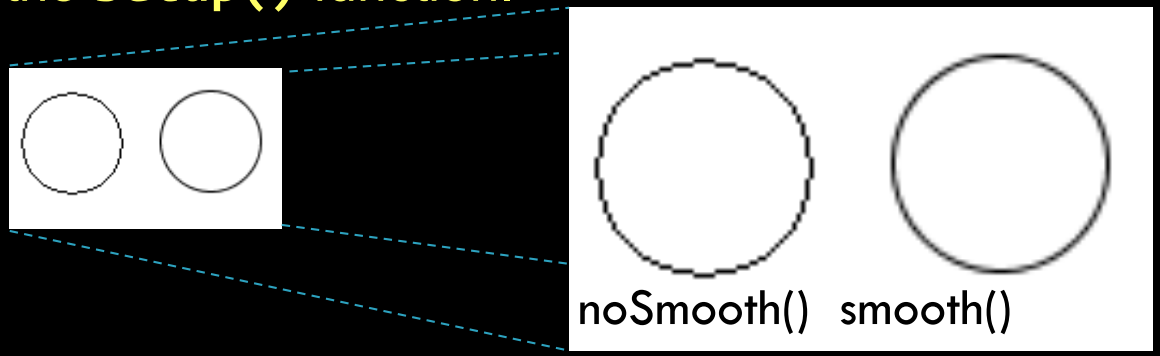

Mouse key is not pressed




Mouse key is pressed

# Aside: `smooth()` and `noSmooth()`

- By default, all geometry is drawn with smooth (anti-aliased) edges. However, you can control this behaviour using **`smooth()`** function to enable this feature, and **`noSmooth()`** function to disable smoothing.
- Notes:
  - You don't need to run **`smooth()`** as it is the default behaviour.
    - You may use it if you want to change the anti-aliasing level (1,2,4,8) – the default level 2; i.e. **`smooth(2)`**
  - The maximum anti-aliasing level is determined by the hardware of the machine running the software
    - i.e. no guarantee that **`smooth(4)`** and **`smooth(8)`** will work on your computer.
  - Use both functions inside the **`setup()`** function.



# Coordinates Transformation

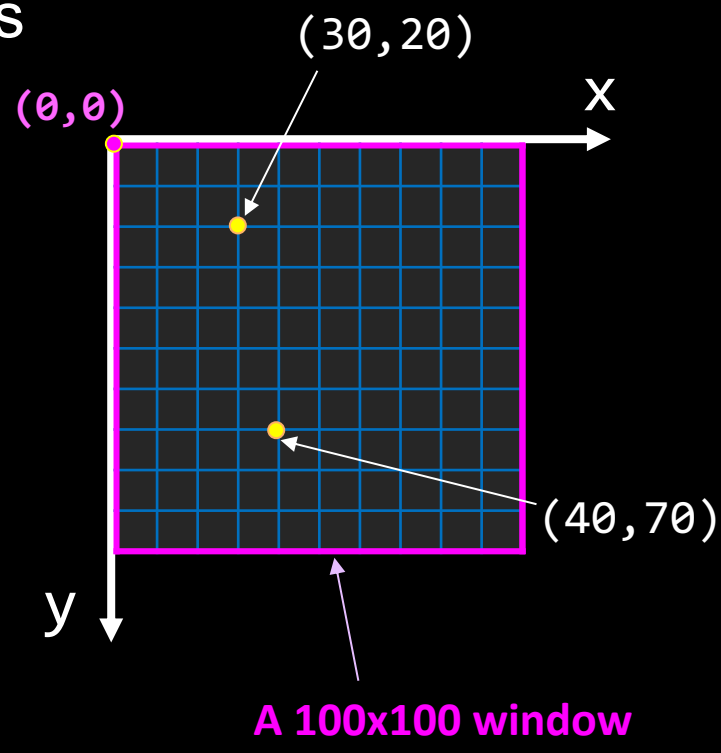
UBC Okanagan
Dr. Abdallah Mohamed
Computer Science

# Key Points

1) How to translate, rotate, and scale the coordinates
2) Coordinates are reset before every new frame.
3) Transformation is cumulative ***within*** each frame.
4) Order is important when combining more than one transformation.
5) Storing and restoring coordinate systems.
6) How to use transformed coordinates in static and dynamic programs.

# *The Default Coordinate System*

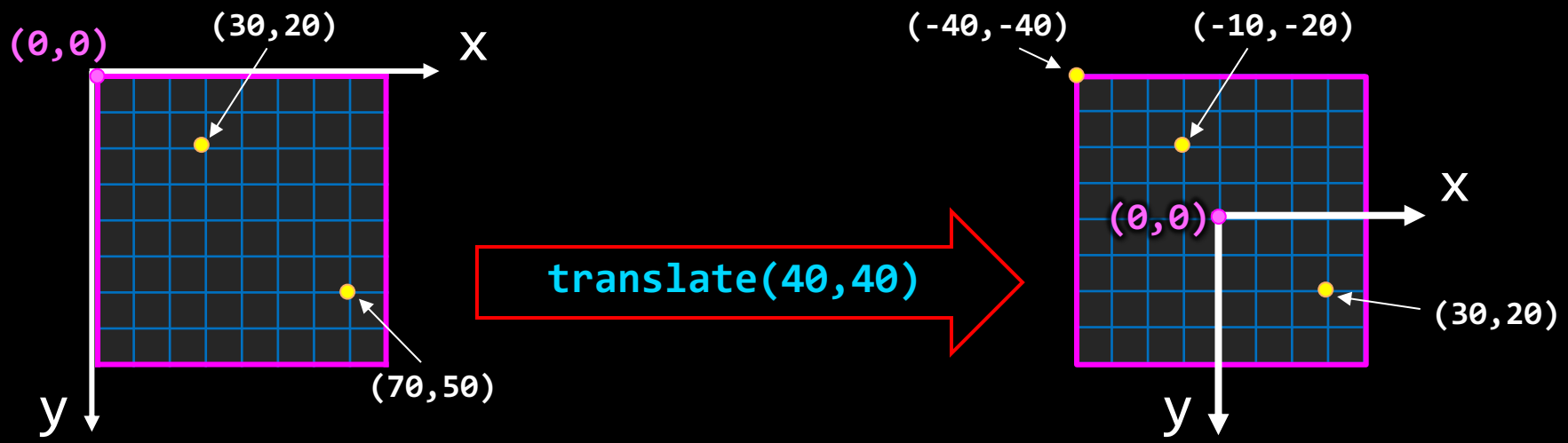
- By default, the coordinate system has its ***origin*** at the upper-left corner of the window, with x and y coordinates as shown in the figure.




- This default representation can be transformed, i.e. ***translated***, ***rotated***, and scaled using built-in functions:
  - `translate()`, `rotate()`, `scale()`
- Only shapes drawn ***after*** the transformation use the new coordinates.
- Coordinates are ***reset at the beginning*** of each new frame (inside `draw()`)

# Coordinate Translation - `translate()`

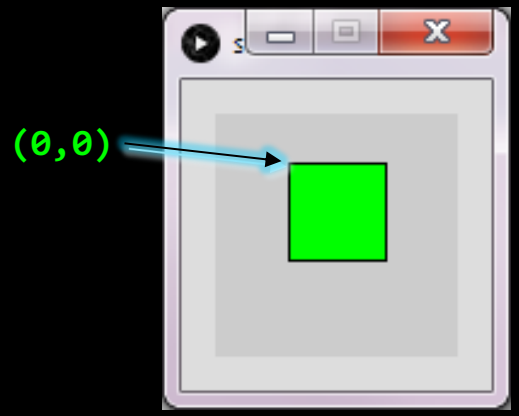
- The `translate()` function moves the origin to a new location.




- `translate()` applies only to shapes drawn after the function call
- You can think of translate is if you are ***adding its arguments*** to all shapes that come after it. i.e. `(30,10) + (40,40) = (70,50)`

# translate() in static mode

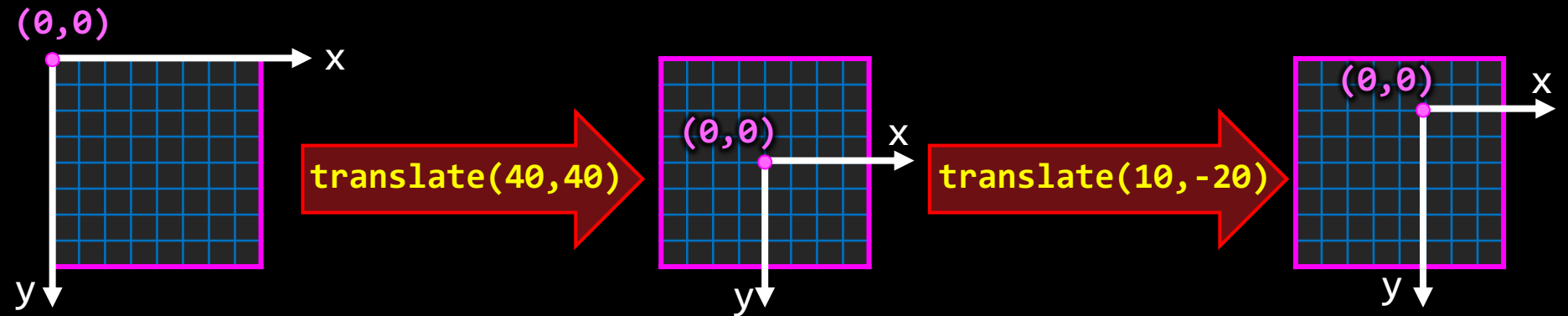
```
//  move origin 30 px right and 20 px down
translate(30, 20);
fill(0,255,0);
rect(0, 0, 40, 40);   // Draw at new origin
```



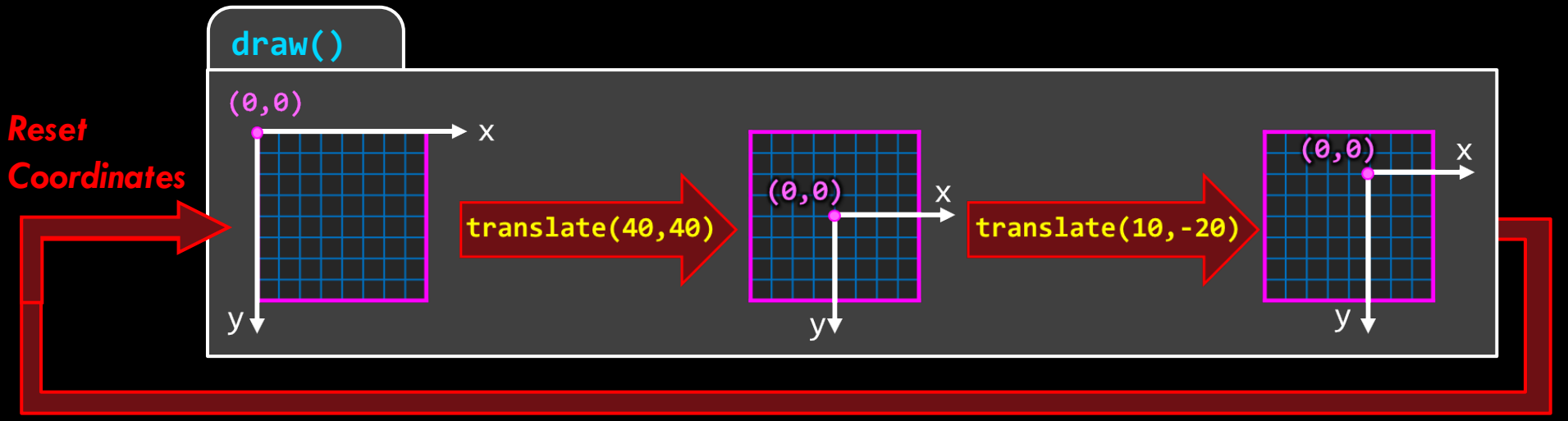

- ***Remember*** that we can think of the output as if we **add the translation** value to the **location of rectangle (assuming the original is still at top-left corner)**. That is,
  - The (x,y) of the green rectangle is (**0**+30,**0**+20) if the original is still at top-left corner.

# Coordinate Translation - `translate()`

- `translate()` is **cumulative** **within** each frame.

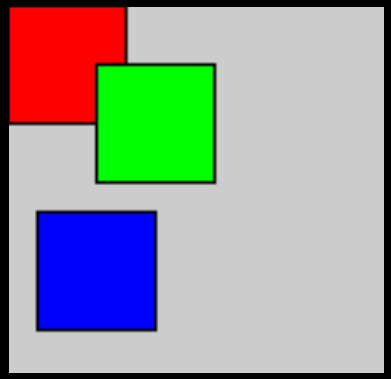



- However, the coordinates are **reset** for each new frame.
  - i.e. if you use transform the coordinates within draw() method, the next frame assumes default origins and then translate again.



# translate() in static mode

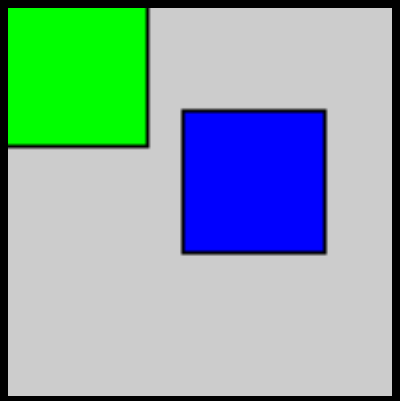
```
// Draw rect at default origin
fill(255,0,0);          // Red
rect(0, 0, 40, 40);

// move origin 30 px right and 20 px down
translate(30, 20);
fill(0,255,0);          // Green
rect(0, 0, 40, 40);     // Draw at new origin

// move origin again 20 px left and 50 px down
translate(-20, 50);
fill(0,0,255);          // Blue
rect(0, 0, 40, 40);     // Draw rect at new
```



- ***Another way*** of thinking of the output is that we **add the translation** value to the location of shapes drawn after the function call. That is,
  - The (x,y) of the green rectangle is (**0**+30,**0**+20)
  - The (x,y) of the blue rectangle is (**0**+30-20,**0**+20+50)

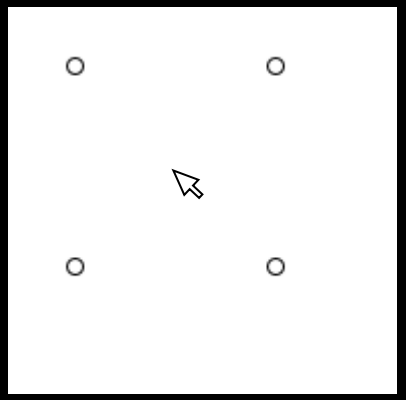
# translate() in dynamic mode

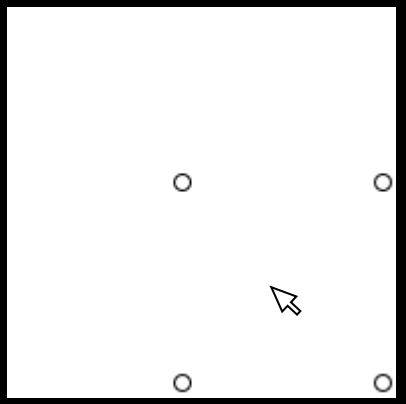
```
void draw() {
  // every beginning of new frame, default origin at (0,0) is used
  fill(0,255,0);          // Green
  rect(0, 0, 40, 40); // Draw rect at new origin

  translate(50, 30);      // move origin to (50, 30)
  fill(0,0,255);          // Blue
  rect(0, 0, 40, 40); // Draw rect at new origin
}
```



# Moving all items with the mouse

```
void draw() {
   background(255);
   // Translate to the mouse location
   translate(mouseX, mouseY);
   ellipse( 30,  30, 6, 6);
   ellipse(-30,  30, 6, 6);
   ellipse( 30, -30, 6, 6);
   ellipse(-30, -30, 6, 6);
 }
```





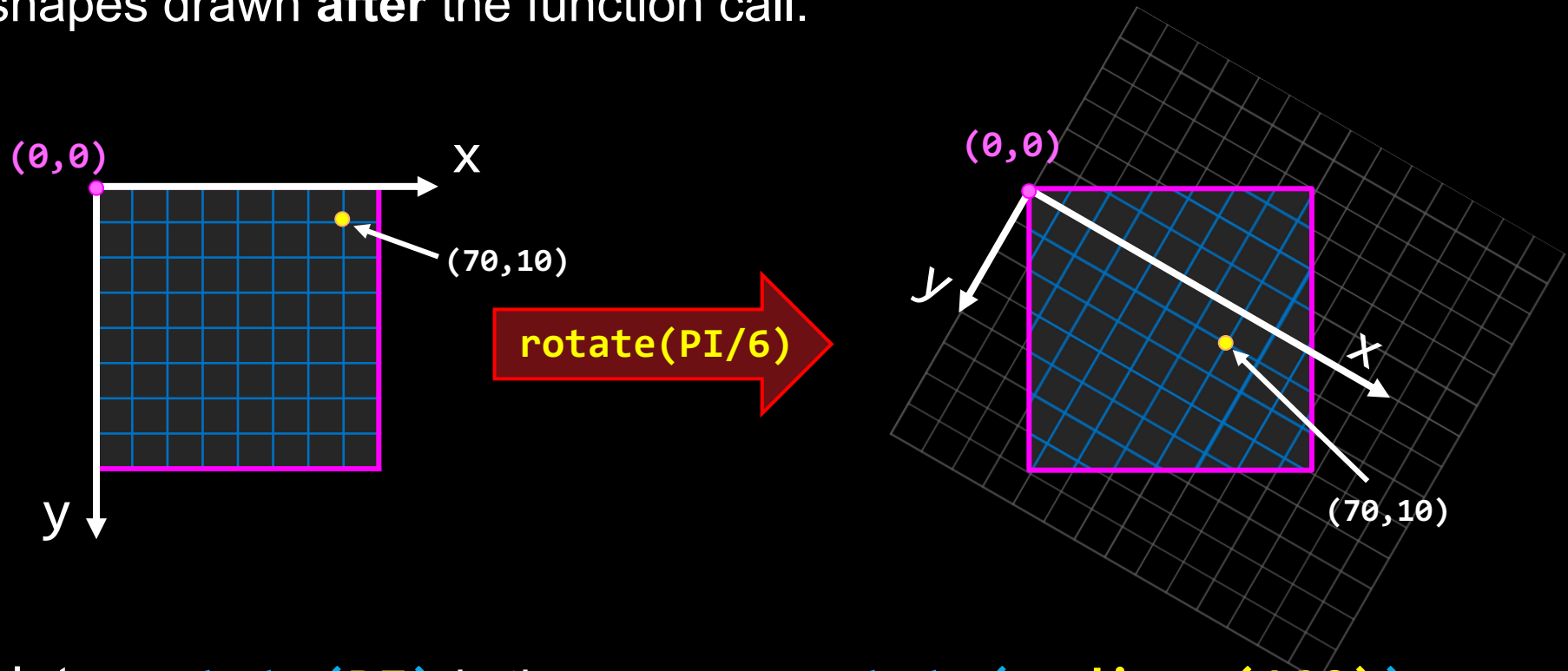
**Q1:** is there another way to write the code **without translate()**?

**Q2:** what is the ***benefit*** of using translate() over the other method?

*(remember the exercise of moving your character with the mouse)*

# Coordinate Rotation - `rotate()`

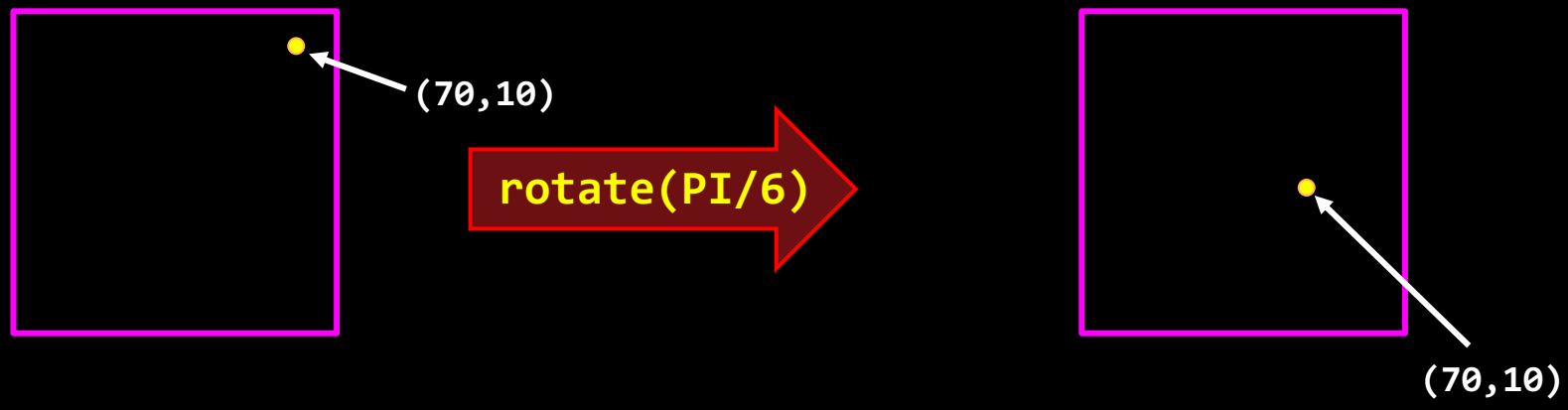
- The `rotate()` function rotates the axes to a new angle.
- It has one parameters, the angle specified in **radians**.
- Similar to `translate()`, rotate() is cumulative and applies only to shapes drawn **after** the function call.




- Note: `rotate(PI)` is the same as `rotate(radians(180))`

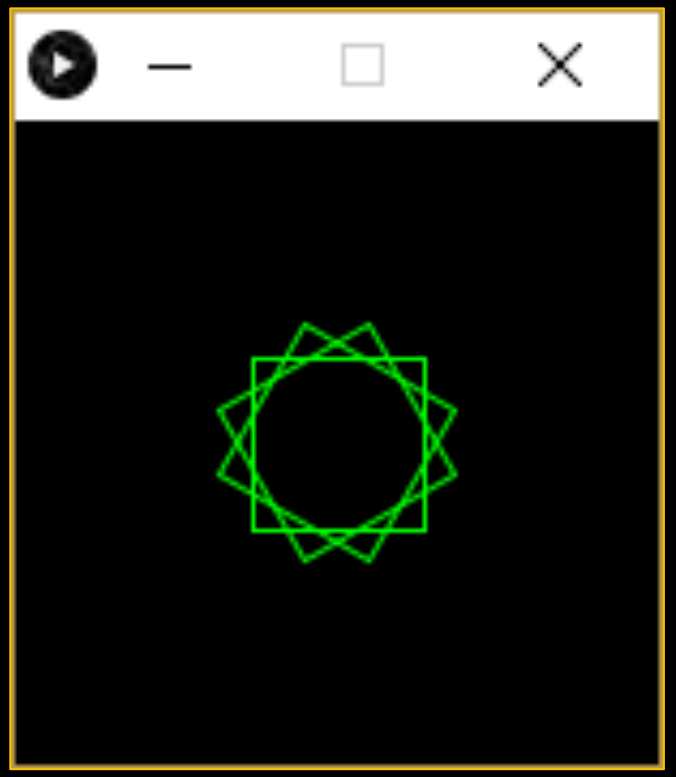
# Coordinate Rotation - `rotate()`

- The `rotate()` function rotates the axes to a new angle.
- It has one parameters, the angle specified in **radians**.
- Similar to `translate()`, rotate() is cumulative and applies only to shapes drawn **after** the function call



# rotate() Example

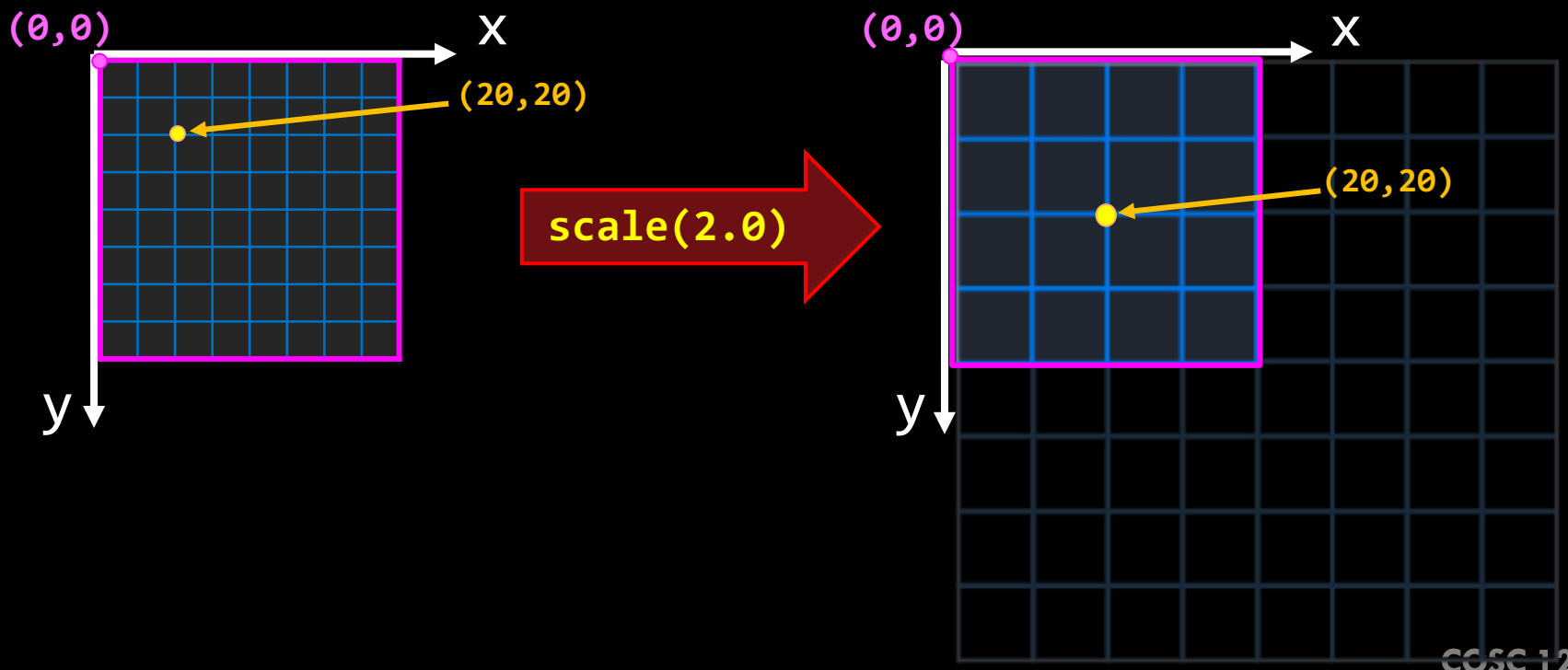
```
size(150, 150);
background(0);
noFill();
stroke(0, 255, 0); // green outline
rectMode(CENTER);

translate(75, 75); // origin at sketch center

rotate(PI/6);      // rotate 30 degrees
rect(0, 0, 40, 40);

rotate(PI/6);      // rotate 30 degrees more
rect(0, 0, 40, 40);

rotate(PI/6);      // rotate 30 degrees more
rect(0, 0, 40, 40);
```



Q. Link statements to shapes in sketch.

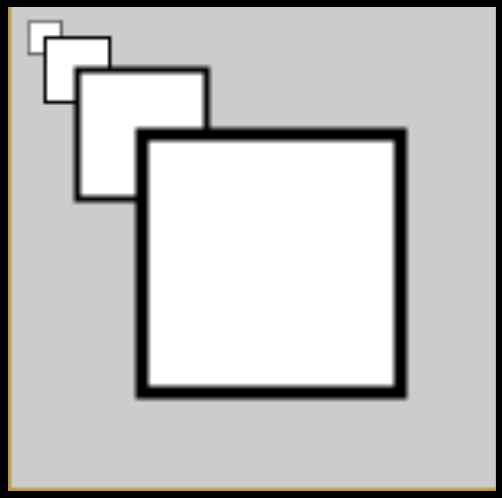
# Coordinate Scaling - `scale()`

- The `scale()` function scales the coordinate system so that shapes are drawn in a different scale (this also affects pixel and border size).
  - Two functions: scale(size) and scale(xsize, ysize)
- Similar to other transforms, rotate() is cumulative and applies only to shapes drawn after the function call



# scale() Example

```
size(150, 150);

scale(0.5);  // scale is 50%
rect(10, 10, 20, 20);

scale(2);    // now scale is back to 100%
rect(10, 10, 20, 20);

scale(2);    // scale is 200%
rect(10, 10, 20, 20);

scale(2);    // scale is 400%
rect(10, 10, 20, 20);
```



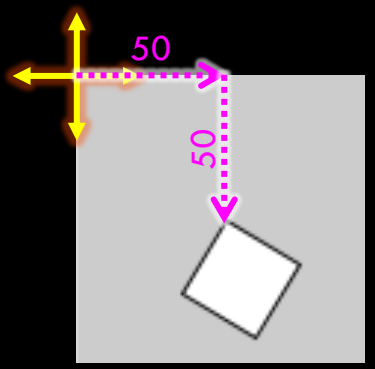
Q. Link statements to shapes in sketch.

# *Order Matters!*

***Order is important*** when combining more than one transformation.

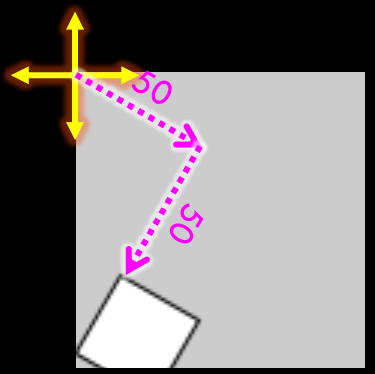
- In the first example, the coordinates are translated first then rotated

```
// translate then rotate
translate(50, 50);
rotate(PI/6);
rect(0, 0, 30, 30);
```




- In the second example, the coordinates are rotated first then translated.

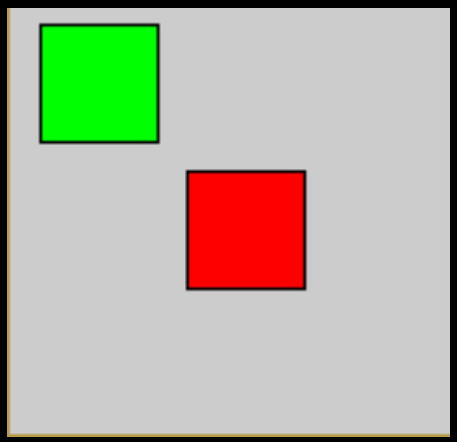
```
// rotate and translate
rotate(PI/6);
translate(50, 50);
rect(0, 0, 30, 30);
```



# Storing and Restoring Coordinates

- The coordinate system is saved as a transformation matrix.
- You can use **pushMatrix()** and **popMatrix()** to store and restore the current coordinate system.
- Example:

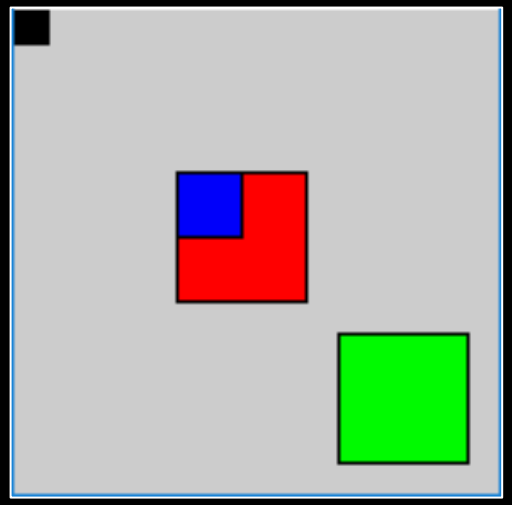
```
size(150,150);

pushMatrix();       // save current origin

translate(50, 50); //origin at (50,50)
fill(255,0,0);      //red
rect(10,10,40,40);

popMatrix();       //retrieve last stored origin

fill(0,255,0);     //green
rect(10,10,40,40);
```



# Aside: stacking transformations

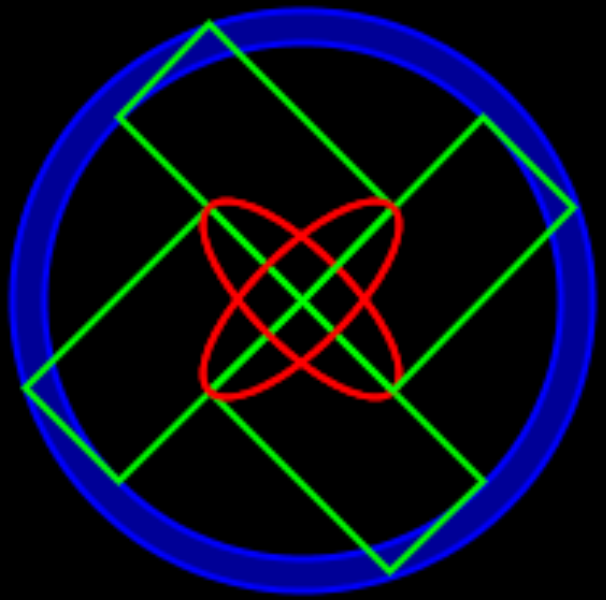
- You can use **pushMatrix()** and **pushMatrix()** multiple times
  - This case, the will be added/removed using the “matrix stack”.

```
size(150,150);

pushMatrix();       // save default origin
translate(50, 50); //origin at (50,50)
fill(255,0,0); rect(0,0,40,40); //red

pushMatrix();       // save current transformation
translate(50,50);  // origin at (100,100)
fill(0,250,0); rect(0,0,40,40); //green

popMatrix();       // restore prev origin - (50,50)
fill(0,0,250); rect(0,0,20,20);  //blue

popMatrix();       // restore original origin (0,0)
fill(0); rect(0,0,10,10);         //black
```



# Coordinate Transformation

- [3 points] Write code to produce the output below. You can only use `rect()` and `ellipse()` functions to draw the shapes. All shapes must be located at (x,y) = (0,0), i.e. the origin of the shape is (0,0) – use coordinate transformation to place the shapes.



# *Moving YOUR Character using transform()*

[1 point] ***Previously***, you moved your character by adding mouseX and mouseY to every (x,y) of all shapes in your character.

***Today***, we will move the character using a simpler technique.

- 1) copy your character code from Exercise2 in the "Color" slides
- 2) add one statement at the beginning to move (translate) your character.

Computer Creativity

*Pre-Class Reading*

# *Variables, Data Types, and a bit of Math*

*Part(A)*

UBC Okanagan

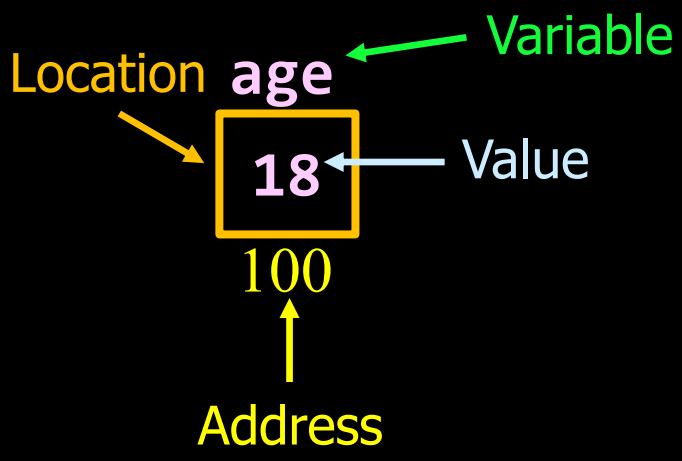
Dr. Abdallah Mohamed
Computer Science

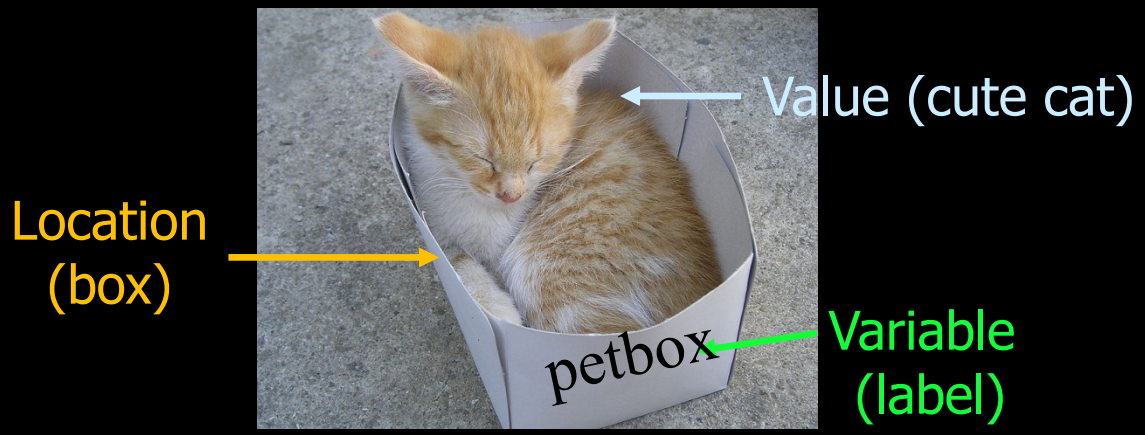
# Objectives

- This is a  pre-class reading materials. After reading them, you should be able to:
  - Define value, variable, and memory location
  - Create and use variables of different data types
  - recognize the naming rules and guidelines for variables.
  - List and compare the data types in processing.
  - Define and use the “color” type.
  - Creating and using constants.
  - Properly use math operators.
  - Evaluate math expressions.

# Values, Variables, and Locations

- A ***value*** is a data item that is manipulated by the computer.
- A ***variable*** is the name that the programmer uses to refer to a location in memory.
- A ***location*** has an address in memory and stores a value.



# *Values, Variables, and Locations*

- Let's say we want to store a number that represents the age.
  - **Step #1**: ***Declare*** variable by giving it a ***NAME*** and a ***TYPE***.

```
int age;   // age can only store integers
```

    - The computer allocates space for the variable in memory (at some memory address). Every time we give the name **age**, the computer knows what data item we mean.

  - **Step #2**: ***Initialize*** the variable to have a starting value. E.g.,

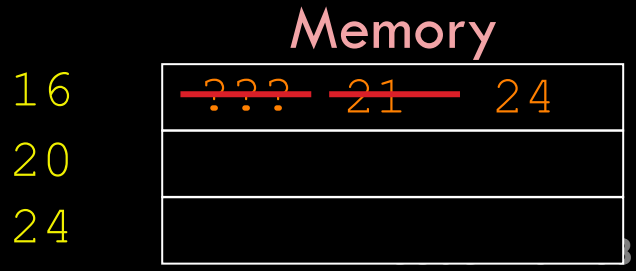
```
age = 21;
```

  - **Step #3**: Value stored in a location can be changed throughout the program to whatever we want using ***assignment*** ("**=**" symbol).

```
age = age + 3;
```

Variable Name Lookup Table

| Name | Location | Type |
|---|---|---|
| age | 16 | int |



# *Variable Name*

- A variable must have a ***NAME*** and a ***TYPE***.

- Names (aka ***identifiers***):
  - ***are case sensitive*** (*B* is not the same as *b*)
  - can be a sequence of characters that include only *letters*, *digits*, *underscores* ( _ ), and *dollar signs* ($).
  - must start with a letter, an underscore (_), or a dollar sign ($).
    - cannot start with a digit.
  - can ***not*** be a reserved word.
    - E.g. cannot be called double, true, false, or null.
  - Naming guidelines
    - can be of any length, but reasonable (readable) length is preferred.
    - should start with a lower case letter.
    - if more than one word, remove the spaces and capitalize all words after the first one (e.g. my first car → `myFirstCar`)

# *Variable Type*

- A variable must have a ***name (identifier)*** and a ***type***. Each type has a valid range of values and uses a different amount of memory space.

| | Type | Size in memory | Range |
|---|---|---|---|
| whole numbers | `byte` | 8 bits | $-2^7$ to $2^7-1$ (-128 to 127) |
| | `short` | 2 bytes | $-2^{15}$ to $2^{15}-1$ (-32768 to 32767) |
| | `int` | 4 bytes | $-2^{31}$ to $2^{31}-1$ |
| | `long` | 8 bytes | $-2^{63}$ to $2^{63}-1$ |
| real numbers | `float` | 4 bytes | e.g. 17.345f |
| | `double` | 8 bytes | e.g. 12452.212 (more accurate) |
| characters | `char` | 2 bytes | e.g. 'a', '1'and '?' |
| boolean | `boolean` | 1 byte | true or false |

- ***Note***: Unlike JavaScript, where you don't specify a type (i.e. just use `var`), in Java (and Processing) you must specify the variable type.

# The String Type

- ***Strings*** are sequences of characters inside double quotes (i.e. text in double quotes).
- Example:

```
String personName = "Abdallah Mohamed";
personName = "John Smith";
```

  - The first statement creates (defines) a variable and initializes its value to “Abdallah Mohamed”.
  - The second statement is assigns a new value to existing variable.

- The ***concatenation operator*** is used to combine two strings into a single string. The notation is a plus sign '+'.

```
String firstName = "Abdallah", lastName = "Mohamed";
String fullName = firstName + lastName;
```

# The Color Type

- Processing introduces a new data type called **color** which stores color information. The value of a color variable can be set by the **color()** function.

```
color red = color(255,0,0);       // red in RGB mode
color navy = color(#443F76);      // navy in hex notation (RGB)

colorMode(HSB,360,100,100);

color green = color(128,100,100);// green in HSB mode
color blue = color(#0011FF);      // blue in hex notation (RGB)

background (navy);     // navy background

fill(red);
rect(10,10,40,40);     // red square

fill(green);
rect(50,50,40,40);     // green square

fill(blue);
ellipse(75,25,30,30); // green square
```

**red** uses RGB as it was defined before changing the color mode

**blue** uses RGB even though it was defined after setting the color mode because **blue** was defined using hex notation

Example 1

# Declaring and Initializing Variables

```
// Declaring Variables
double a;      // Declare a to be a double variable
int x, y;      // Declare x and y as integer variables

// Assignment Statements
a = 7.1;       // Assign 7.1 to a;
x = 1 + 3;     // assign 4 to x;
y = x + 2;     // assign 6 to y;

// Declaring and Initializing in ONE Step
double a2 = 7.1;
int x2 = 1, y2 = 2;
```

# Using Variables

- Here are two more examples of two variables x and y

```
int x;                  // declare a variable
x = 5;                  // initialize a variable – what is assignment '='?
int y = 10;             // declare and initialize a variable
println(x);             // print value of x
x = 10;                 // overwrite old value
println("x " + x);      //what is the output?
```

```
int x = 10, y;          // y has no values yet
y = x;                  // y is 10 now
y = y + 1;              // = does not mean equal, it means assignment.
println("x + y = " + (x + y));  //notice the output
```

# Constants

- Constants are similar to variables except that once initialized they cannot change.
- To create a constant, use the keyword **final** before your variable declaration.

```
final double PI = 3.14159;
final int SIZE = 3;
```

- Naming Convention:
  - Capitalize all letters in constants
    - e.g. MAX, PI, SIZE
  - Use underscores for multiple words.
    - e.g. MAX_VALUE

# *Expressions*

# The Assignment Statement

- An ***assignment statement*** changes the value of a variable.
  - The variable on the left-hand side of the = is assigned the value from the right-hand side.
  - The value may be changed to a constant, to the result of an expression, or to be the same as another variable.
  - The values of any variables used in the expression are always their values before the start of the execution of the assignment.
- Example:

```
int A, B;
A = 5;
B = 10;
A = 10 + 6 / 2;
B = A;
A = 2*B + A - 5;
```

**Question**: What are the values of A and B?

# Expressions

- An ***expression*** is a sequence of operands and operators that yield a result. An expression contains:
  - ***operands*** - the data items being manipulated in the calculation
    - e.g. `5, "Hello, World", myDouble`
  - ***operators*** - the operations performed on the operands
    - e.g. `+, -, /, *, %` (modulus or remainder after integer division)

- An operator can be:
  - ***unary*** - applies to only one operand
    - e.g. `d = -3.5;` // "-" is a unary operator, 3.5 is the operand
  - ***binary*** - applies to two operands
    - e.g. `d = 3 * 5.0;` // "*" is binary operator, 3 and 5.0 are operands

- ***Integer Division:***
  - 5 / 2 if both operands are integers, the output is an integer 2
  - 5.0 / 2 if at leas one operand is float, output is float 2.5

# *Division Operator*

- What is the result of 25 / 4?
- How would you rewrite the expression if you wished the result to be a floating-point number?
- Are the following statements correct? If so, show the output.

```
println("25 / 4 is " + 25 / 4);
println("25 / 4.0 is " + 25 / 4.0);
println("3 * 2 / 4 is " + 3 * 2 / 4);
println("3.0 * 2 / 4 is " + 3.0 * 2 / 4);
```

# *The Remainder Operator (%)*

- The % operator returns the remainder of two numbers.
- Examples:

| Operation | Result |
|---|---|
| a) 14 % 6 | 2 |
| b) -34 % 5 | - 4 (matches numerator sign) |
| c) -34 % -5 | - 4 |
| d) 34 % -5 | 4 |
| e) 5 % 1 | 0 |
| f) 1 % 5 | 1 |
| g) 3 % 0 | runtime error. Can't divide by zero |

# *Operator Precedence*

- Each operator has its own priority similar to their priority in regular math expressions:
  1. Any expression in parentheses is evaluated first starting with the inner most nesting of parentheses.
  2. Unary + and unary - have the next highest priorities.
  3. Multiplication and division (*, /, %) are next.
  4. Addition and subtraction (+,-) are then evaluated.

# *The ++ and -- Operators*

- It is very common to subtract 1 or add 1 from the current value of an integer variable.
- There are two operators which abbreviate these operations:
  - `++` add one to the current integer variable
  - `--` subtract one from the current integer variable
- Example:

```
var j=0;
j++;      // j = 1;  Equivalent to j = j + 1;
j--;      // j = 0;  Equivalent to j = j - 1;
```

# *Augmented Assignment*

- The operators +, -, *, /, and % can be combined with the assignment operator = to form augmented operators.

```
x += 5;   //Equivalent to   x = x + 5
x -= 5;   //Equivalent to   x = x - 5
x *= 5;   //Equivalent to   x = x * 5
x /= 5;   //Equivalent to   x = x / 5
x %= 5;   //Equivalent to   x = x % 5
```