Computer Creativity

Getting Started with Processing

Okanagan

Slides courtesy of Dr. Abdallah Mohamed.

Computer Creativity

Notes

Processing Basics

Okanagan

Slides courtesy of Dr. Abdallah Mohamed.

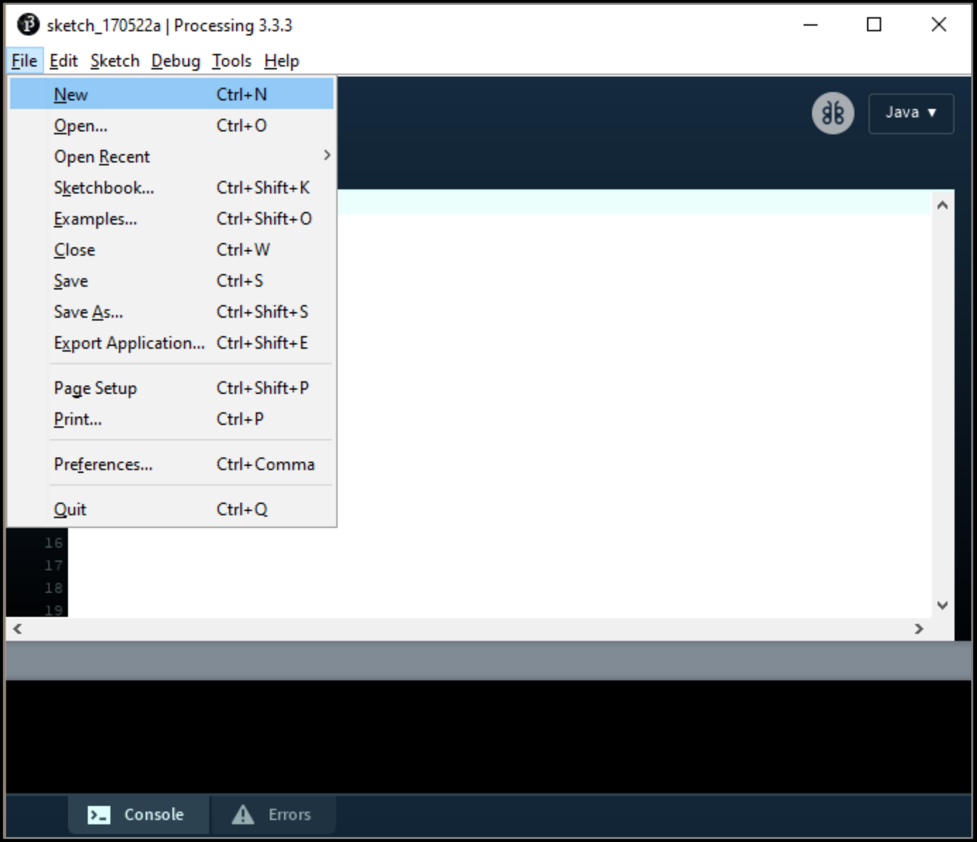
Objectives

- This are notes. After finishing reading these notes, you should be able to:
  - Create a new processing file
  - Draw four primitive shapes: point, line, rectangle, and oval
  - Set the sketch size.
  - Add comments to your code
  - Recognize that processing is case-sensitive and accepts free-form format.

Submit Lab 1 together

PDE: Creating and Running a Sketch

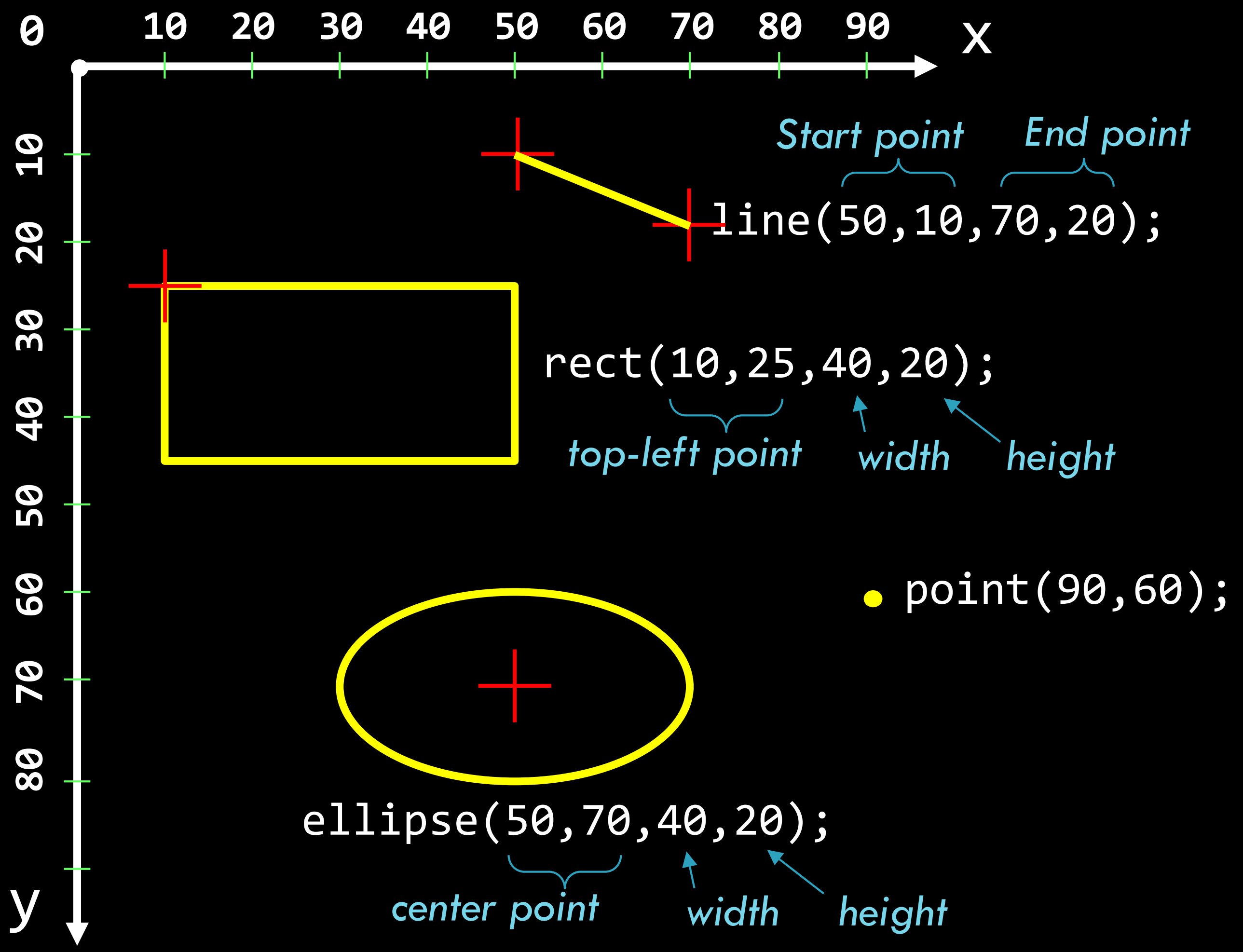
- To create a program code file, select `File->New` or
- Your new program is called a ***sketch*** in Processing. Sketches are saved in a folder on your computer called ***sketchbook***.
- To write your code, start typing in the Text Editor" area of the PDE.
- Use the buttons ***Run*** and ***Stop*** on the toolbar to run or terminate your program.



Primitive Shapes

- Example of primitive shapes
  - Point: `point(90,60);`
  - Line: `line(50,10,70,20);`
  - Rectangle: `rect(10,25,40,20);`
  - Ellipse: $\underset{\substack{\uparrow \\ \textit{Function name}}}{\texttt{ellipse}}\texttt{(}\underbrace{\texttt{50,70,40,20}}_{\textit{Parameters}}\texttt{);}$

Drawing Primitive Shapes

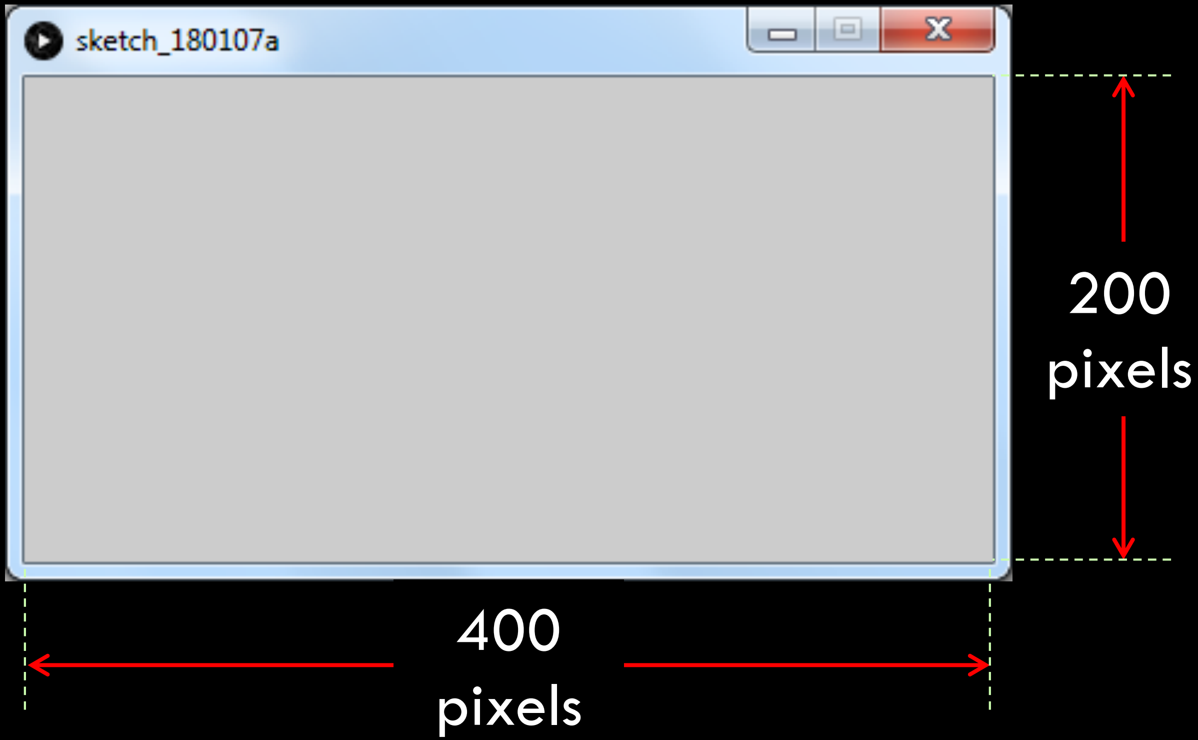


```
line(50,10,70,20);
rect(10,25,40,20);
point(90,60);
ellipse(50,70,40,20);
```

Sketch Size

- To set the size of your sketch, use the `size()` function. For example, the following line sets the sketch width and height to 400 and 200 pixels respectively.

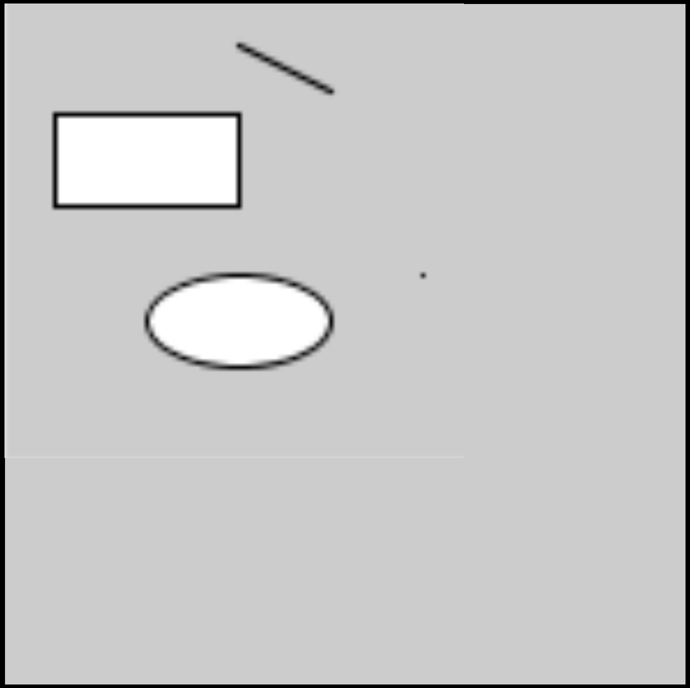
```
size(400,200);
```



Sketch Size: *Example*

- In the previous class, you wrote code to draw primitive shapes.
- The standard size of a sketch is 100x100 pixels
- The following program changes the size of the sketch to 150x150.

```
// set sketch size to 150x150
size(150,150);

// draw shapes
line(50,10,70,20);
rect(10,25,40,20);
point(90,60);
ellipse(50,70,40,20);
```



Sketch in Full Screen

- You can run your code in full screen using the function
  `fullScreen();`

- You can choose only ***ONE*** of the two functions `fullScreen()` and `size()` in any program.

```
// sketch in full screen
fullScreen();

// draw shapes
line(50,10,70,20);
rect(10,25,40,20);
point(90,60);
ellipse(50,70,40,20);
```

Syntax Rules

Syntax Rules: Comments

- Comments are used by the programmer to document and explain the code. Comments are ignored by the computer.
- There are two choices for commenting:
  - 1) One line comment: put "//" before the comment and any characters to the end of line are ignored by the computer.
  - 2) Multiple line comment: put "/*" at the start of the comment and "*/" at the end of the comment. The computer ignores everything between the start and end comment indicators.
- Example:

```
/* This is a multiple line
   comment.
With many lines. */
// Single line comment
// Single line comment again
line(10,10,20,20);      // Comment after code
```

More Syntax Rules

- To program in Processing you must follow a set of rules for specifying your commands. This set of rules is called a ***syntax***.
- Processing is case sensitive.
  - `Line()` is not the same as `line()`.
- Processing accepts ***free-form layout***.
  - Spaces and line breaks are not important except to separate words.
  - You can have as many words as you want on each line or spread them across multiple lines.
  - However, you should be consistent and follow the programming guidelines given for assignments.
    - It will be easier for you to program and easier for the marker to mark.
  - You can use "Auto Format" PDE feature to rearrange your code in a more readable form

Notes

Primitive Shapes, Text

Objectives

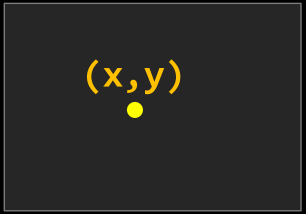
- These are some notes for you to work on outside of class. After finishing these notes, you should be able to:
  - Recognize and use primitive shape functions
    - point(), line(), rect(), ellipse(), quad(), triangle(), bezier()
  - Understand and specify shape ***coordinates***
    - i.e. specify the reference point or origin of a shape.
  - Specify the attributes of drawing ***stroke***.
  - Write ***text*** on your sketch

Drawing Primitive Shapes

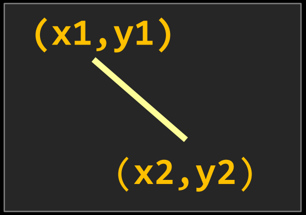
- You learned before how to draw some of the primitive shapes, namely: point, line, ellipse, and rectangle.

- There are other primitive shapes that we can also use such as: the quad, the triangle, and the Bezier line.

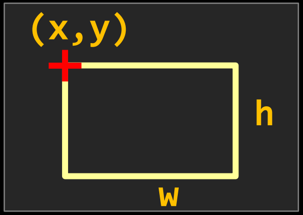
Primitive Shapes



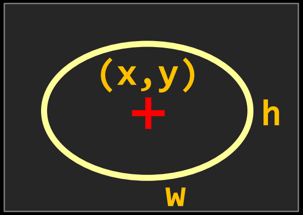
`point(x,y)`



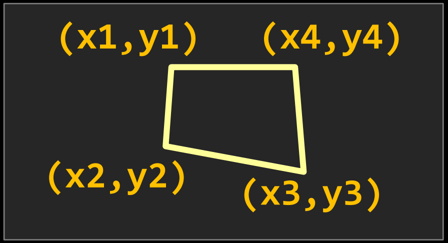
`line(x1,y1,x2,y2)`



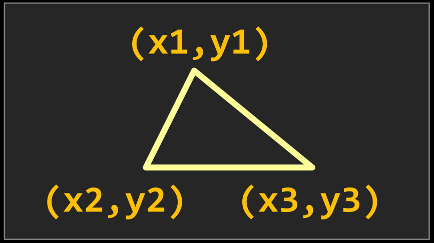
`rect(x,y,w,h)`



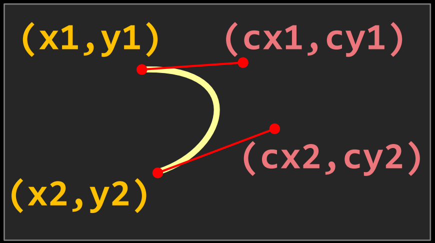
`ellipse(x,y,w,h)`



`quad(x1,y1,x2,y2,x3,y3,x4,y4)`



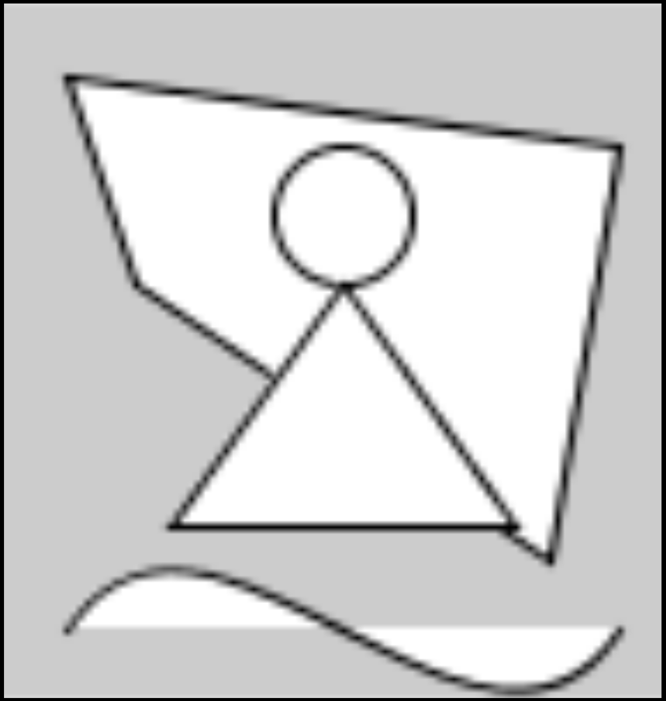
`triangle(x1,y1,x2,y2,x3,y3)`



`bezier(x1,y1,cx1,cy1,cx2,cy2,x2,y2)`

Primitive Shapes

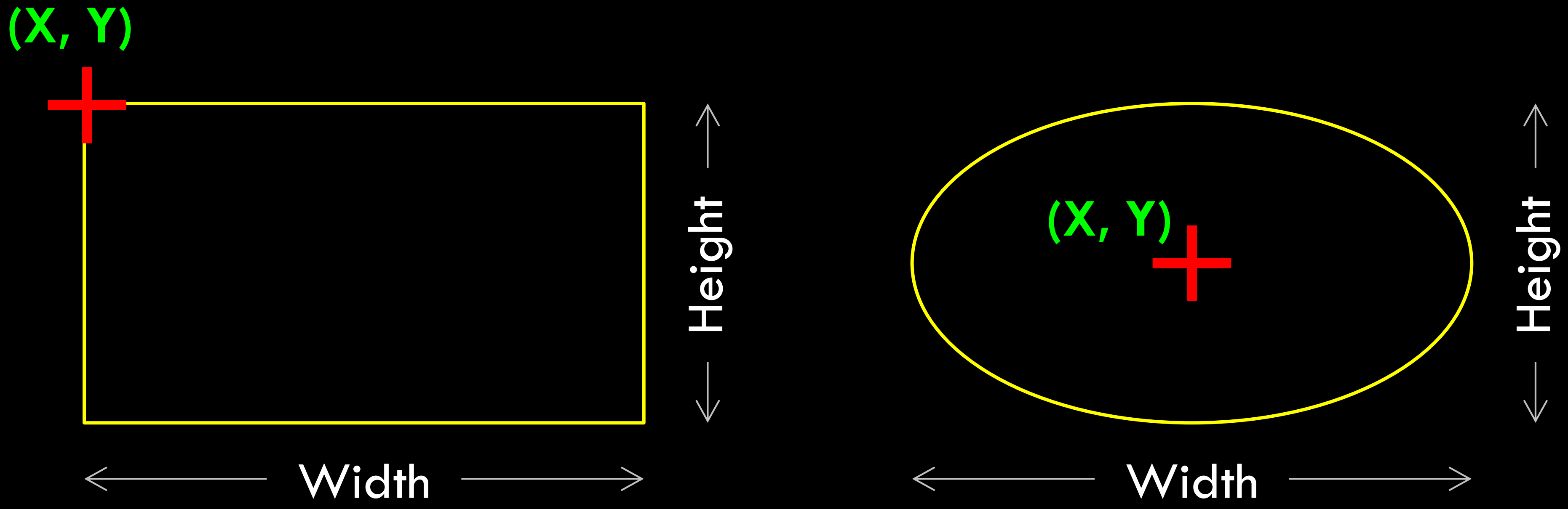
```
quad(10,10,20,40,80,80,90,20);

ellipse(50,30,20,20);

triangle(50,40,25,75,75,75);

bezier(10,90,30,60,70,120,90,90);
```



Defualt Shape Coordinates

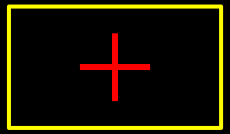
- The *default* coordinates for **`rect`** and **`ellipse`** are:

```
rect(Top_Left_X, Top_Left_Y, Width, Height)  → CORNER
ellipse(Center_X, Center_Y, Width, Height)   → CENTER
```

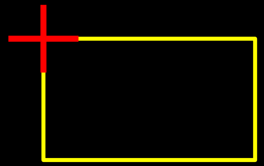


Specifying Shape Coordinates

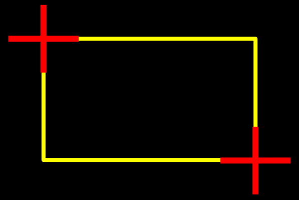
- Default coordinates can be explicitly set to one of three modes:
  - CENTER
    `(Center_X, Center_Y, Width, Height)`



  - CORNER
    `(Top_Left_X, Top_Left_Y, Width, Height)`



  - CORNERS
    `(Top_Left_X, Top_Left_Y, Bottom_Right_X, Bottom_Right_Y)`

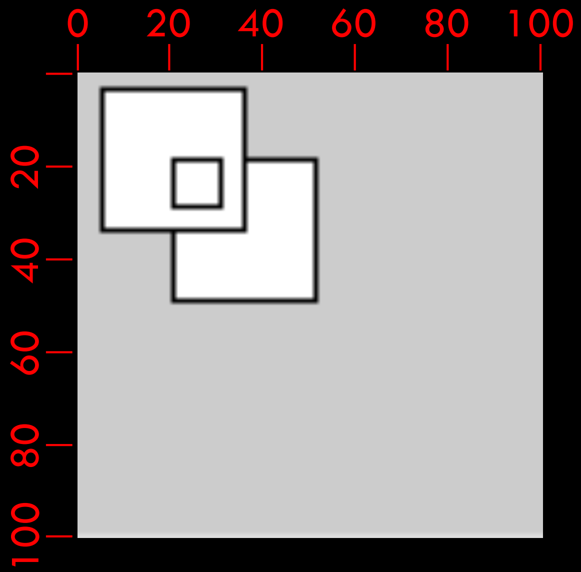


- The above applies to **`rect`** and **`ellipse`** but not necessarily to all shapes

Specifying Shape Coordinates, cont'd

- You can change the mode using `rectMode` and `ellipseMode` functions.

```
// set the sketch size
size(100,100);
// draw
rectMode(CORNER);     //this is the default mode
rect(20,20,30,30);
rectMode(CENTER);      //default is CORNER
rect(20,20,30,30);
rectMode(CORNERS);      //default is CORNER
rect(20,20,30,30);
```

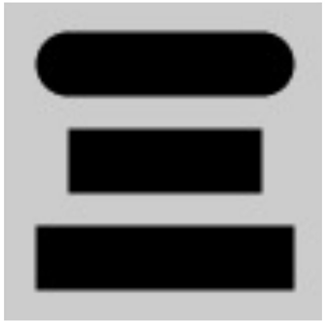


- *Question: Can you link each statement to the right shape?*

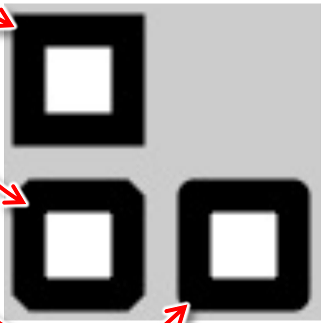
Stroke Attributes

- Stroke attributes are controlled by:
  - `strokeWeight()`: *Sets the width of the stroke in pixels*. Takes one number (the width). Default is 1 pixel.
  - `strokeCap()`: *Sets the endpoints*. Takes one parameter that can be **ROUND**, **SQUARE**, or **PROJECT**. Default is **ROUND**.
  - `strokeJoin()`: Determines how line segments connect including the corners of any shape. Takes one parameter that can be **MITER**, **BEVEL**, or **ROUND**. Default is **MITER**.

```
strokeWeight(20);
strokeCap(ROUND);
line(20, 20, 80, 20);
strokeCap(SQUARE);
line(20, 50, 80, 50);
strokeCap(PROJECT);
line(20, 80, 80, 80);
```



```
strokeWeight(10);
strokeJoin(MITER);
rect(10, 10, 30, 30);
strokeJoin(BEVEL);
rect(10, 60, 30, 30);
strokeJoin(ROUND);
rect(60, 60, 30, 30);
```



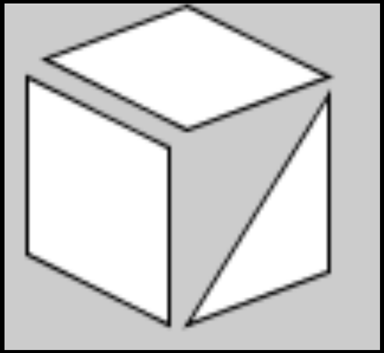
Lecture Activity

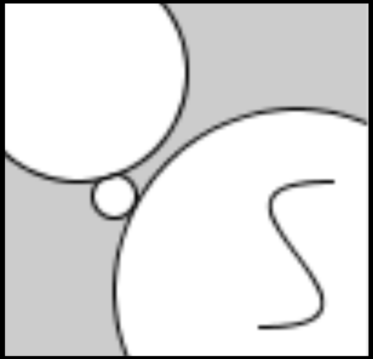
Accept the GH Classroom Link on the course website:

Canvas > Course Content > GitHub Classroom Links

Draw Primitive Shapes

- Write code to draw the following sketches. Assume reasonable dimensions.


(a)


(b)

- Hint: sketch your drawing on paper first, try to figure out the coordinates, then write code

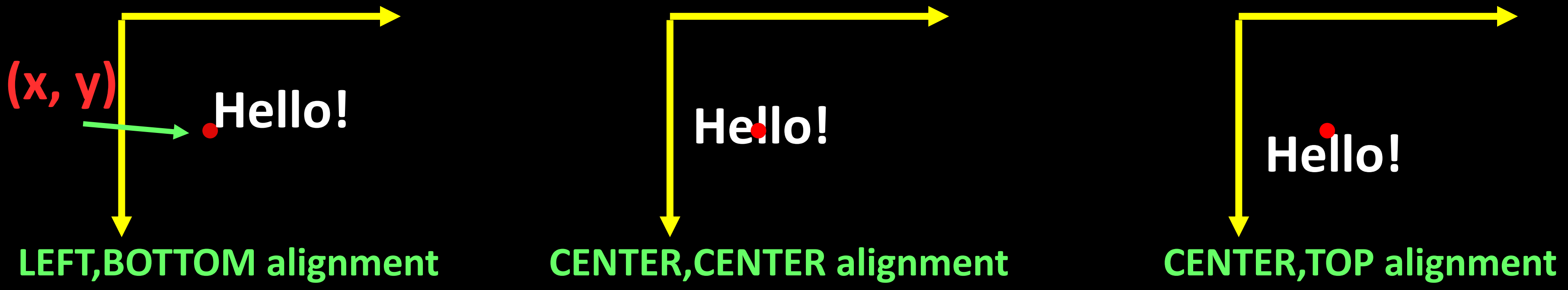
See you on Friday!

Getting Started with Processing

Drawing Text

Drawing Text

- You can add text to your sketch using the following functions:
  - `textSize(20)` changes the text size to 20
  - `text("Hello!", x, y)` writes "Hello!" at (x,y)
- Use `textAlign()` to align the text.
  - ***Default*** is "left-bottom.



Drawing Text

- You can also define a textbox so that text wraps inside it using the syntax

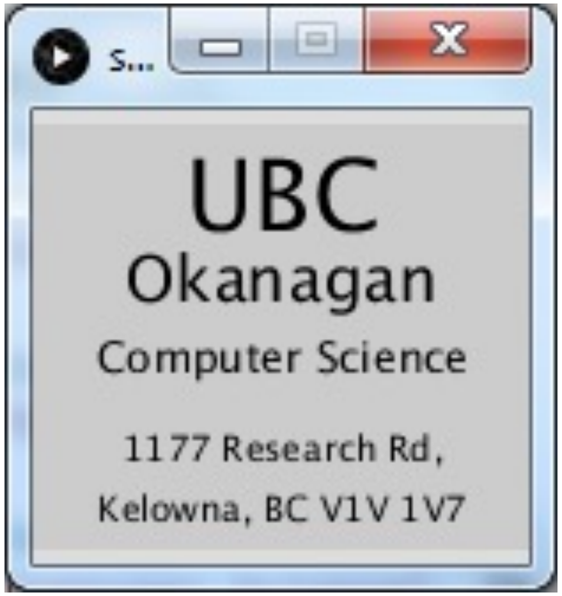
```
text("long text here",x,y,width,height)
```

  - ***Note:*** `width` ***and*** `height` ***parameters are optional***

- **FONT**
  - To change the font, you need two functions: `loadFont()` and `textFont()`.
    - More about this later
- **COLOR**
  - To change the text color, use the `fill` function
    - More about this later

Example

```
size(140,120);
fill(0);   // write in black

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

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

textSize(12);
text("Computer Science", 70, 70);

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



UBC
Okanagan
Computer Science
1177 Research Rd,
Kelowna, BC V1V 1V7

Git and GitHub Demo

Notes

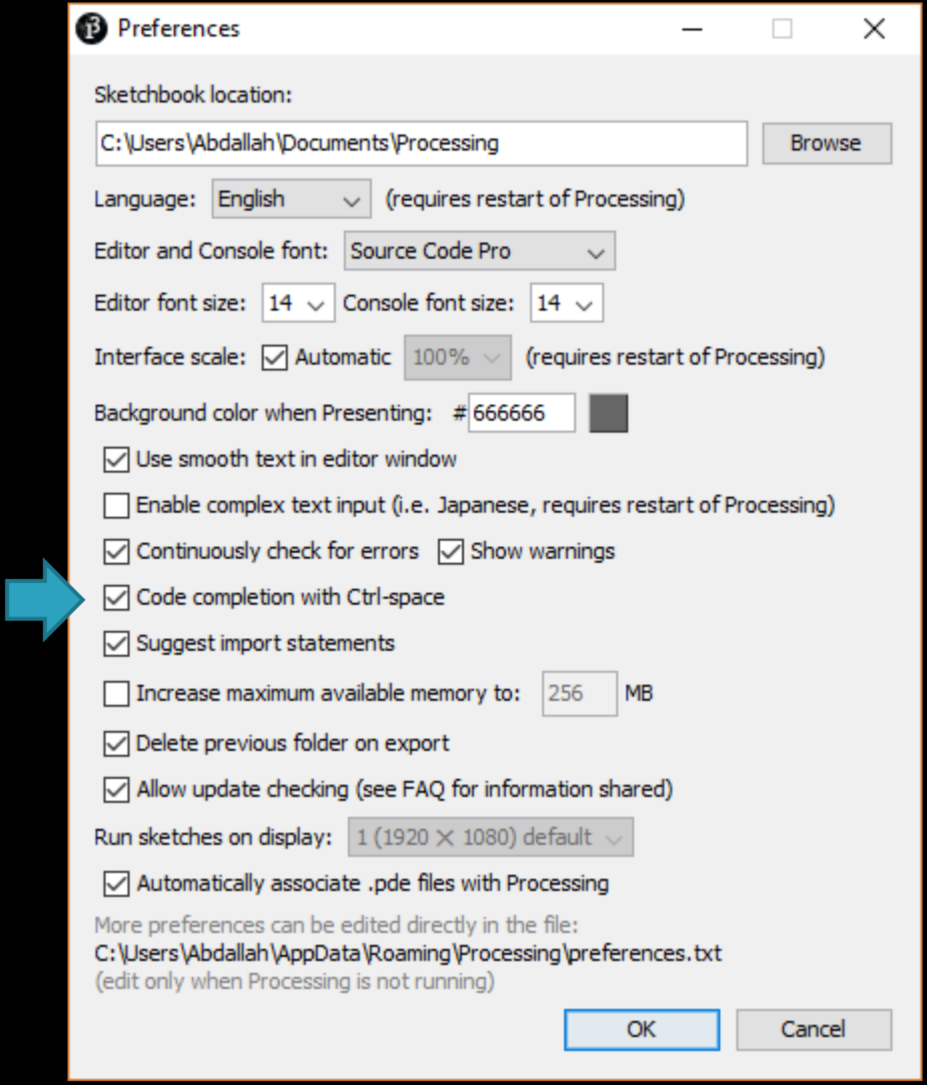
PDE Features

PDE Useful features

- Use **`Edit->Auto Format`** (or **`Ctrl+T`**) to automatically adjust code format to be more readable (i.e. indentation, spacing, etc.).
- Use **`Ctrl+/`** to comment/uncomment a selected section of code.
- Use Auto Compete (**`Ctrl+Space`**) to get code suggestions
  - enable from **`File->Preferences`** (next slide)
- Use the Color Selector (**`Tools->Color Selector…`**) to get the value of a color of your choice.
- You can view many examples that demonstrate the different capabilities of Processing by going to **`File->Examples…`**
- You can add other files (images, fonts, documents, etc.) to use in your sketch from **`Sketch->Add File…`**
  - This can also be done using your preferred file manager (e.g. Windows Explorer), but we will discuss this later.

Code Completion

- It is recommended to use Code Completion feature. You can enable it by going to `File->Preferences` and check that option as shown in the figure.


Preferences

Sketchbook location: C:\Users\Abdallah\Documents\Processing [Browse]

Language: English (requires restart of Processing)

Editor and Console font: Source Code Pro

Editor font size: 14 Console font size: 14

Interface scale: ✓ Automatic 100% (requires restart of Processing)

Background color when Presenting: # 666666

- ✓ Use smooth text in editor window
- ☐ Enable complex text input (i.e. Japanese, requires restart of Processing)
- ✓ Continuously check for errors ✓ Show warnings
- ✓ Code completion with Ctrl-space
- ✓ Suggest import statements
- ☐ Increase maximum available memory to: 256 MB
- ✓ Delete previous folder on export
- ✓ Allow update checking (see FAQ for information shared)

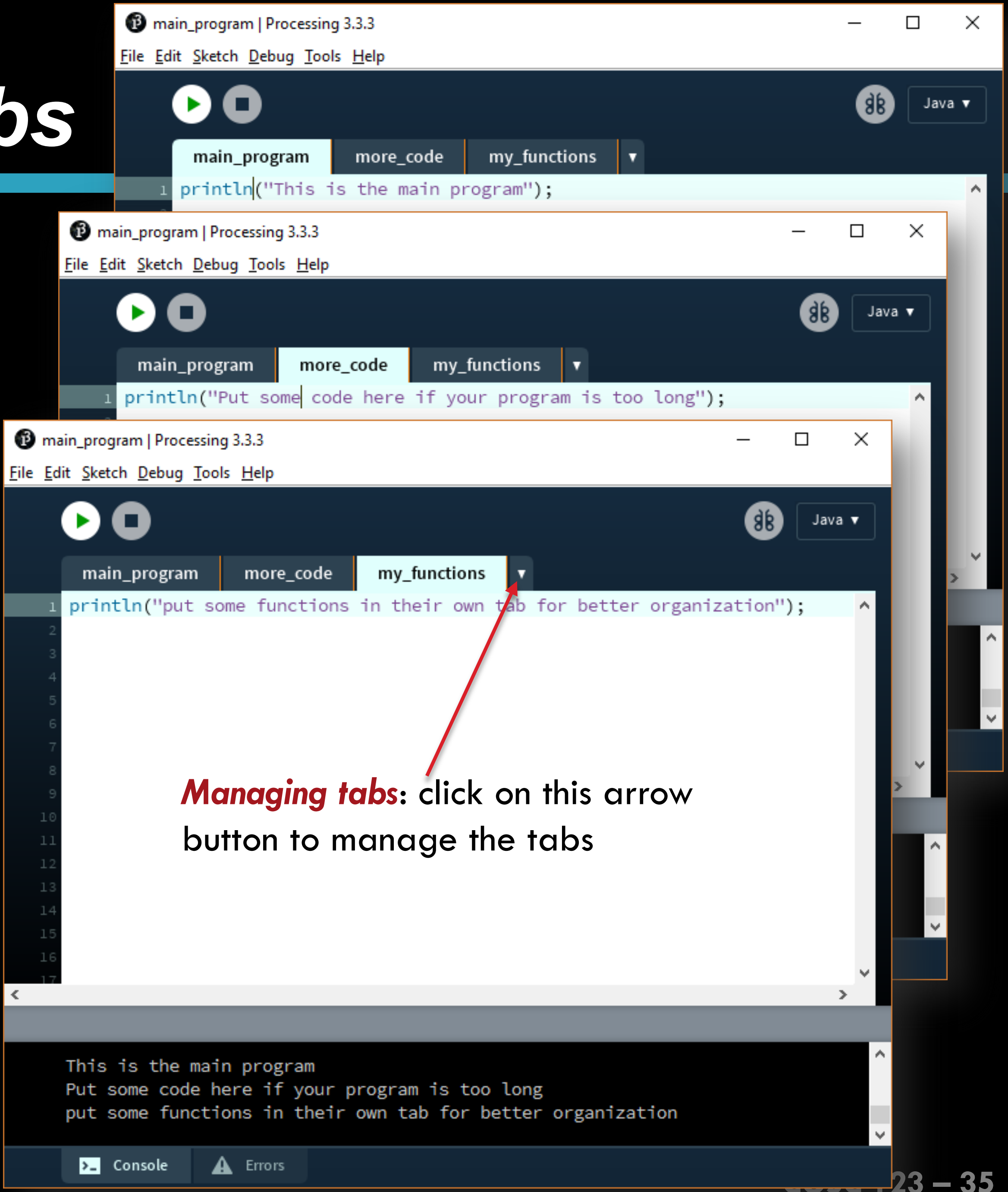
Run sketches on display: 1 (1920 × 1080) default

- ✓ Automatically associate .pde files with Processing

More preferences can be edited directly in the file:
C:\Users\Abdallah\AppData\Roaming\Processing\preferences.txt
(edit only when Processing is not running)

Sketchbook Tabs

- You can divide your code into several files managed by tabs for better structuring.
  - PDE arranges tabs alphabetically by their names.
- The code in all tabs will run as if it is in the same file.
  - Tabs run from left to right.
- Examples:
  - Put classes in tabs.
  - Put your new functions in tabs.


main_program | Processing 3.3.3
File Edit Sketch Debug Tools Help
Java
main_program
more_code
my_functions
Console
Errors

```
println("This is the main program");
```

```
println("Put some code here if your program is too long");
```

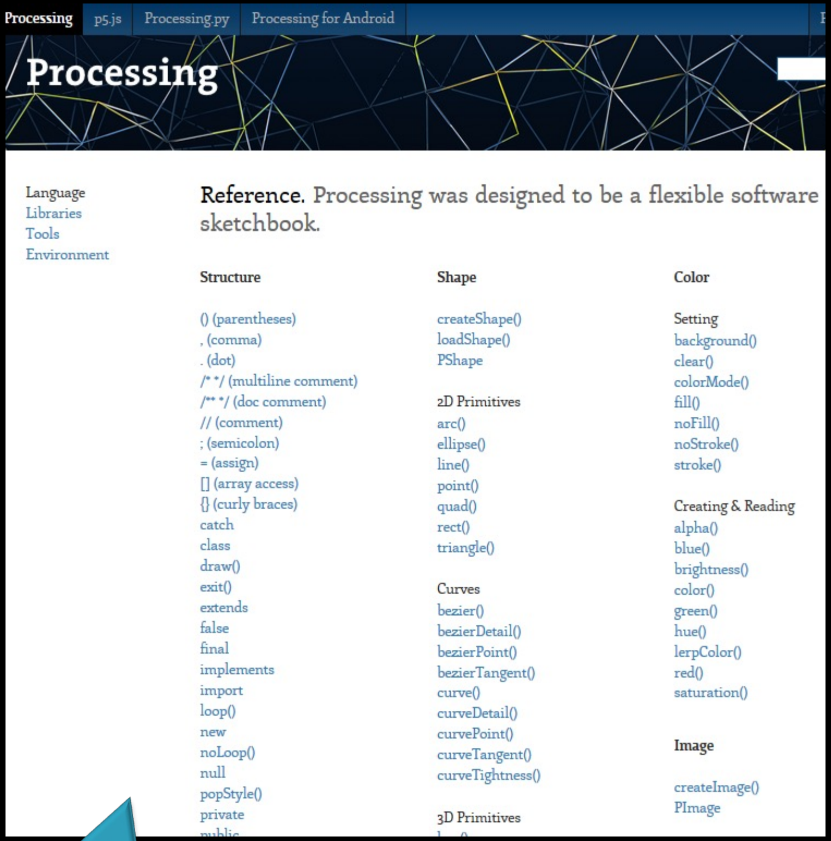
```
println("put some functions in their own tab for better organization");
```

Managing tabs: click on this arrow button to manage the tabs

```
This is the main program
Put some code here if your program is too long
put some functions in their own tab for better organization
```

Processing Language Reference

- You can view a complete reference for the language using **`Help->Reference`**
- If you need help with specific keyword, highlight it then choose **`Find in Reference`** From the **`Help`** menu or the context menu.



Processing | p5.js | Processing.py | Processing for Android

Processing

Language
Libraries
Tools
Environment

Reference. Processing was designed to be a flexible software sketchbook.

Structure

() (parentheses)
, (comma)
. (dot)
/* */ (multiline comment)
/** */ (doc comment)
// (comment)
; (semicolon)
= (assign)
[] (array access)
{} (curly braces)
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
popStyle()
private

Shape

createShape()
loadShape()
PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives

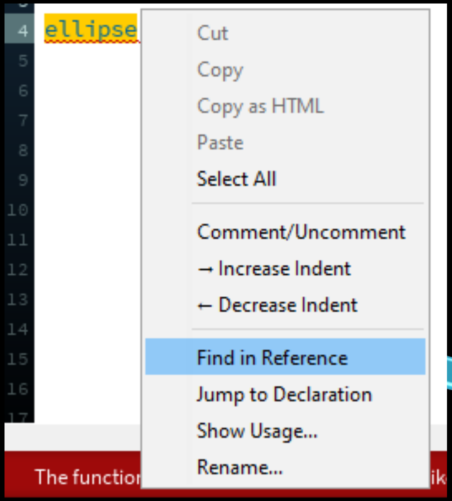
Color

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

Image

createImage()
PImage

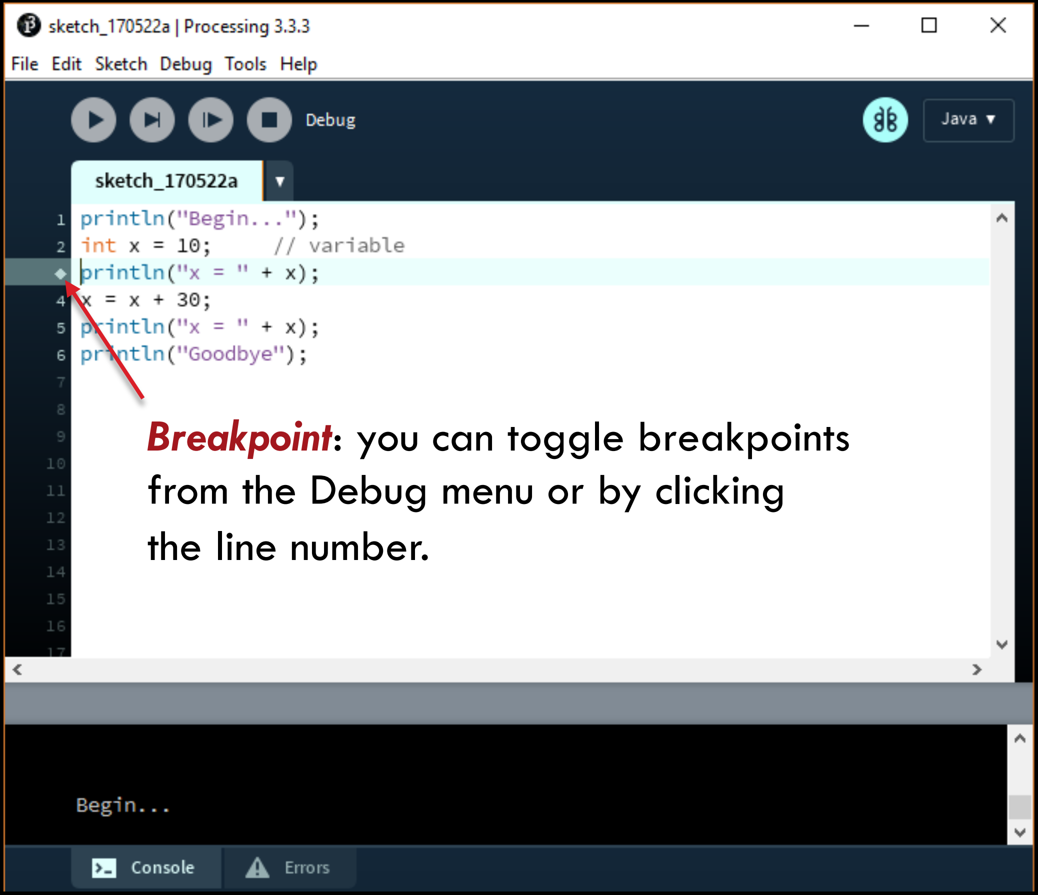


PDE Debugger

You can enable the debugging mode from the debug menu or by clicking the Debugger icon.

Debugger functions:

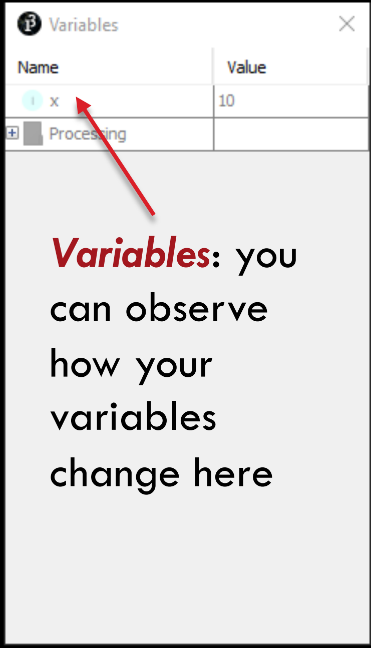
- **Debug**: run till the first breakpoint
- **Continue**: advance the code till the next breakpoint.
- **Step**: advance the code one line.
- **Step Into**: advance the debugger into the a function call.
- **Step Out**: advance the debugger outside a function to the calling statement.


sketch_170522a | Processing 3.3.3
File Edit Sketch Debug Tools Help
Debug
Java
sketch_170522a
Console
Errors

```
println("Begin...");
int x = 10;      // variable
println("x = " + x);
x = x + 30;
println("x = " + x);
println("Goodbye");
```

```
Begin...
```

Breakpoint: you can toggle breakpoints from the Debug menu or by clicking the line number.



Name	Value
x	10
Processing	

Variables: you can observe how your variables change here

Exercise 2

Using PDE Debugger to Trace Code

- Use the PDE Debugger to trace the following code. Notice the change in the x and y values.
  - Step 1: Switch to Debugging mode
  - Step 2: Put a breakpoint at the first line.
  - Step 3: Click Run (Debug).
  - Step 4: Step through your code and observe the change in x,y and in the console

```
int x, y = 20;
x = 10;
println("x: " + x);
println("y: " + y);
x = x + 3;
y = y + x;
println("x: " + x);
println("y: " + y);
println("The End!");
```

No need to submit this to Canvas!!

Tips for Debugging Your Code

- Here are some tips that you may want to try when debugging your code:
  - Trace changes in your variables.
    - If you are not using the PDE Debugger, you can programmatically display the values of those variables related to your problem after they change.
      - You can use println() or text() functions to display the values.
  - Simplify your code.
    - …using comments. Test segments of your code individually and see if they run as expected.
  - Take a break!
    - Do something else or even go to sleep. When you come back, you might see what you weren't able to see before.
  - Get "*someone*" to look at your code.
    - Fresh eyes might catch obvious mistakes that you aren't able to see.

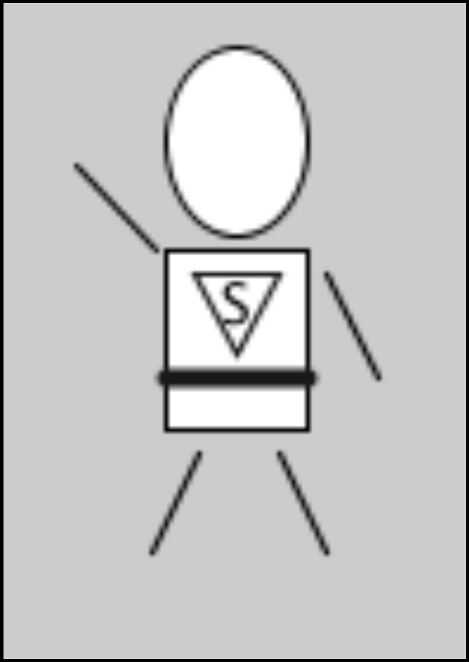
Lecture Activity

Accept the GH Classroom Link on the course website:

Canvas > Course Content > GitHub Classroom Links

Create A Character

- Write code to design a simple character:
  - We will use this character throughout the semester in other exercises. So, try to be creative!
  - No need to worry about the color at this point.
  - Use the easiest drawing mode for aligning your body parts.
    - For example, it would be easier if we use the CENTER drawing mode for the torso.
  - Include the following items:
    - **A belt** (stroke with larger width)
    - A **logo** on the character chest.
  - The design must have at least <u>**one character**</u> of text.
- Hint: sketch your drawing on paper first, try to figure out the coordinates, then write code



Review of Primitive Shapes, Text

The Notes

- Your notes for this week included discussion of:
  - Primitive shape functions
    - point(), line(), rect(), ellipse(), quad(), triangle(), bezier()
  - Shape ***coordinates*** (origin)
  - Stroke attributes
  - Text

Key Points

First:

- self-assess your understanding of the pre-class readings

Then:

1) Practice on primitive shapes and text

Shape Coordinates

The default coordinates for rectangles and ellipses are:

A. CORNER for both

B. CENTER for both

C. CENTER for rectangles, and CORNER for ellipses

D. CORNER for rectangles, and CENTER for ellipses

E. None of the above

Shape Coordinates

We can change the coordinates of a rectangle to CENTER using the statement:

A. coordinate(CENTER);

B. center();

C. rectMode(CENTER);

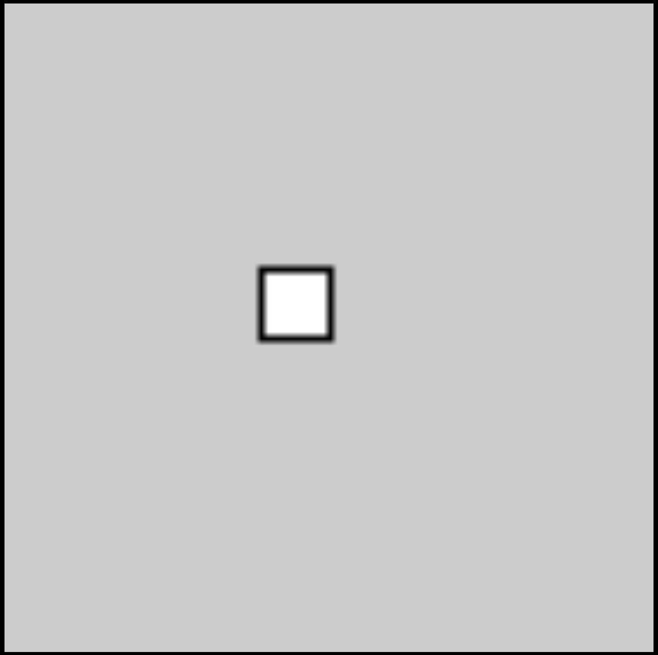
D. mode(CENTER);

E. CENTER;

Specifying Shape Coordinates

Which coordinate mode did we use here?

```
size(100,100);
rectMode(?????);
rect(40,40,50,50);
```



A. CORNER

B. CORNERS

C. CENTER

D. CENTERS

E. Other

Stroke Attributes

We can change the width of the drawing stroke using the function:

A. width()

B. strokeWdith()

C. weight()

D. strokeWeight()

E. stroke()

Text

The statement to write “Hi” on the sketch at (50,50) is

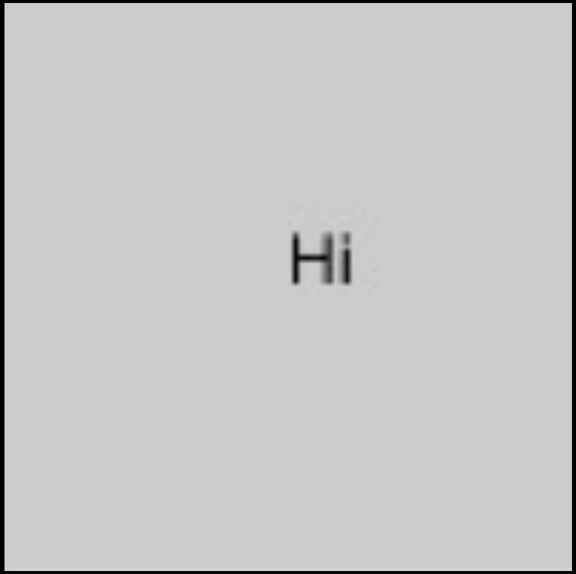
A. `write("Hi",50,50);`

B. `text("Hi",50,50);`

C. `text(50,50,"Hi");`

D. `writeText("Hi",50,50);`

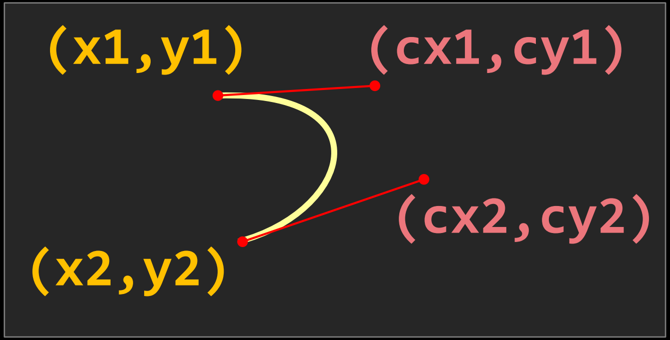
E. `drawText("Hi",50,50);`



Question

Bezier shapes

Have you understood how **`bezier`** function work?



```
bezier(x1,y1,cx1,cy1,cx2,cy2,x2,y2)
```

A. Yes

B. No