COSC 123

Computer Creativity

Introduction to Processing

Okanagan

Slides courtesy of Dr. Abdallah Mohamed.

Announcements

- Ed Discussion
  - EVERYBODY should now be on Ed Discussion! If you aren't, send me an email firas.moosvi@ubc.ca with a non-UBC email so I can invite you
- PrairieLearn (Test 0)
  - Test 0 was not for marks, but it was supposed to get you familiar with the testing system.
  - 80 (!!!!) of you have yet to do this!
- Test 1 Window will be open Thursday at 6 PM!
  - You will have until Saturday at 6 PM to complete the 60 min Test!
- Thursday's class is CANCELLED due to (my) unforeseen circumstances. We will finish content for the week today…

Processing Examples

Slides courtesy of Dr. Abdallah Mohamed.

Course Objectives

1) To be creative with programming and write fun, interesting computer programs.
2) To master fundamental programming skills of data variables, decisions, iteration, methods, and the basics of object-oriented programming, and how to create larger programs
3) To design and develop strategies for solving basic programing problems.
4) To algorithmically create 2D graphics, animations, and simple games using Processing language.
5) To design interactive graphical user interfaces.
6) To learn to solve problems cooperatively as a team (of two).
7) To learn how to switch from Processing to Java.

The Essence of the Course

- If you walk out of this course with nothing else you should:

Become a creative programmer with the ability to problem solve, perform critical thinking, and communicate precisely.

- This course is not only about learning a particular language (or even programming itself), it is about being a creative problem solver and critical thinker!

Programming using Processing

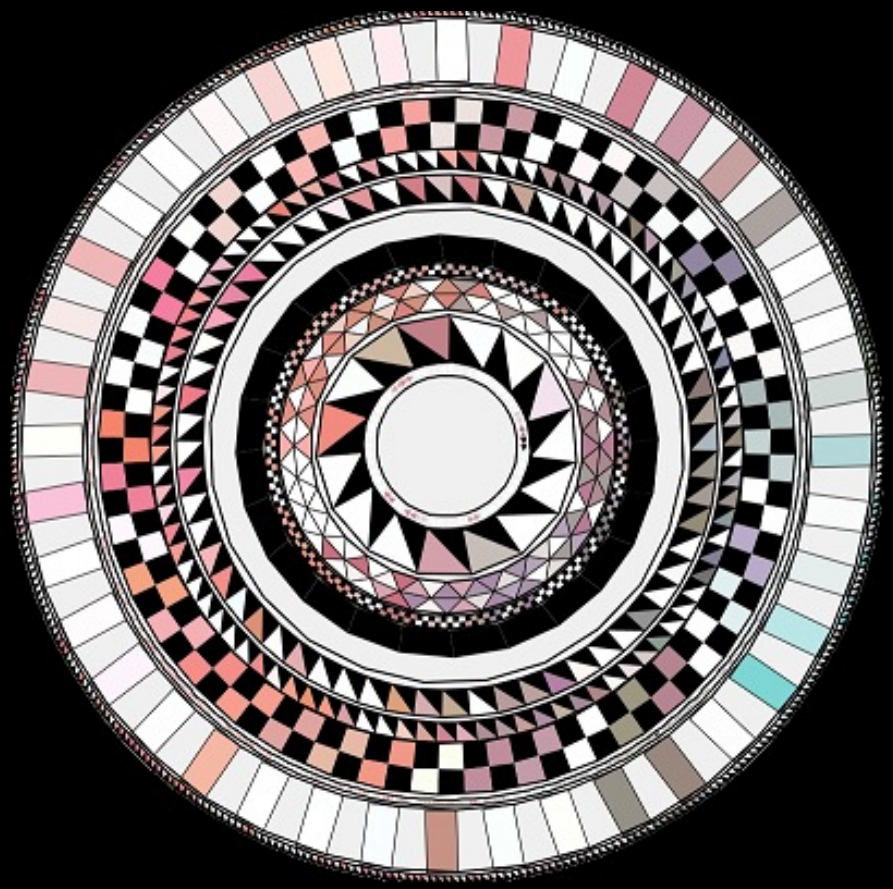
- You already learned algorithmic thinking using basic programming techniques in COSC111 and COSC122.

- In addition to being able to solving algorithmic problems (similar to what you did in COSC 111 or COSC 122), we will try to re-learn programming using graphical functions, especially to create user interfaces, animations, and simple 2D games.

- We will use basic programming techniques (e.g. conditionals, loops, arrays, objects, etc.) on *Sketches* to draw and interact with shapes and images.

Processing Examples!

- Algorithmic Drawing
  - Example: Artistic Designs

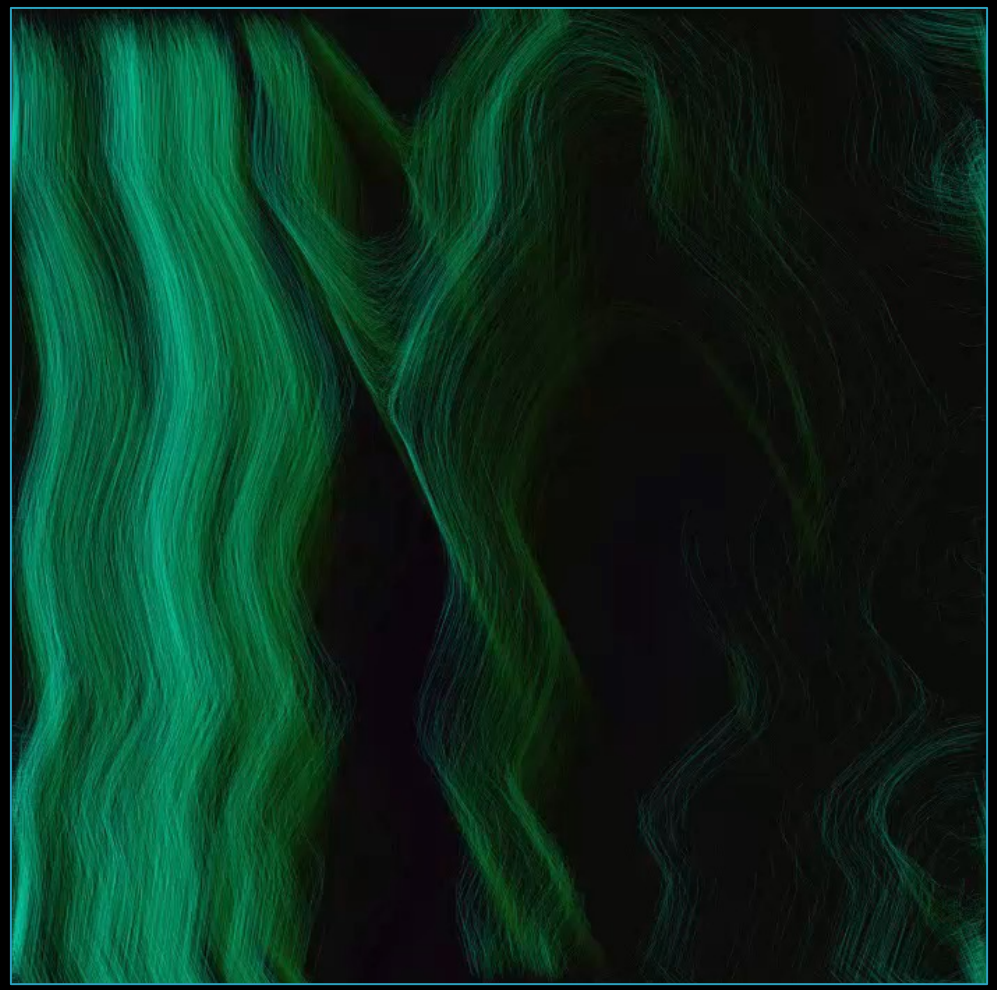


Processing Examples!

- Artistic Animations
  - Example:

Processing Examples!

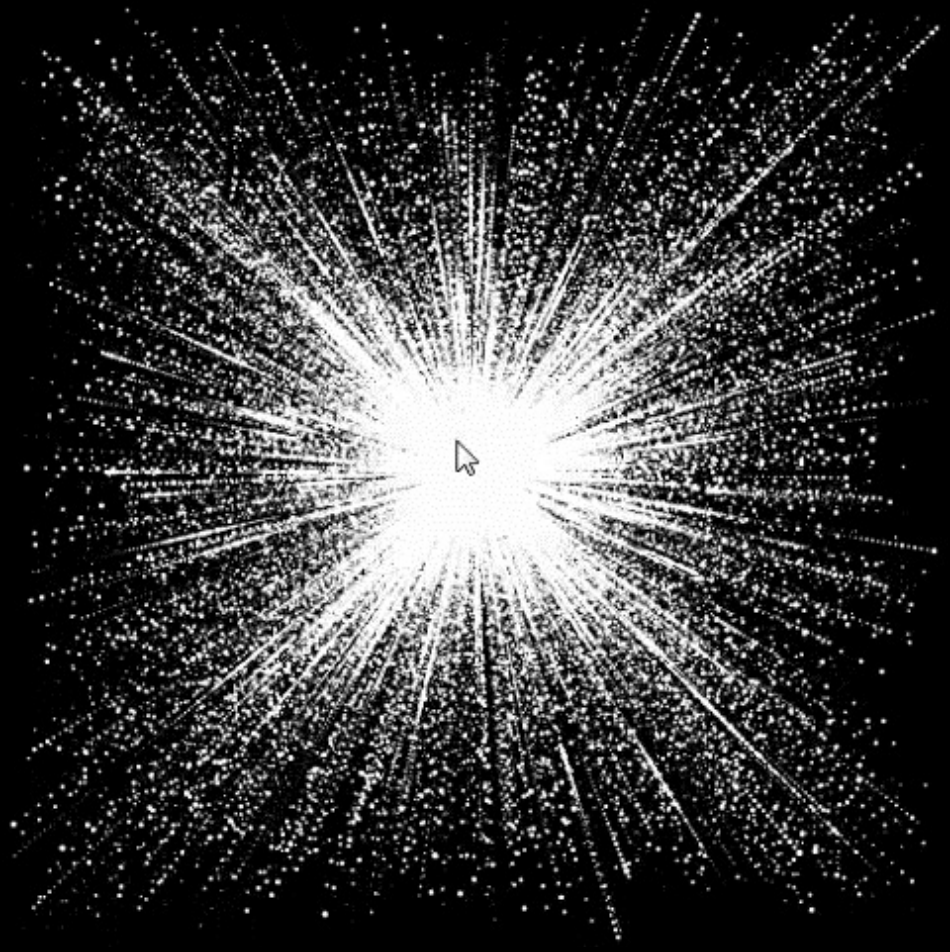
- Artistic Animations
  - Example: particle systems



By: Timo Lachmeijer

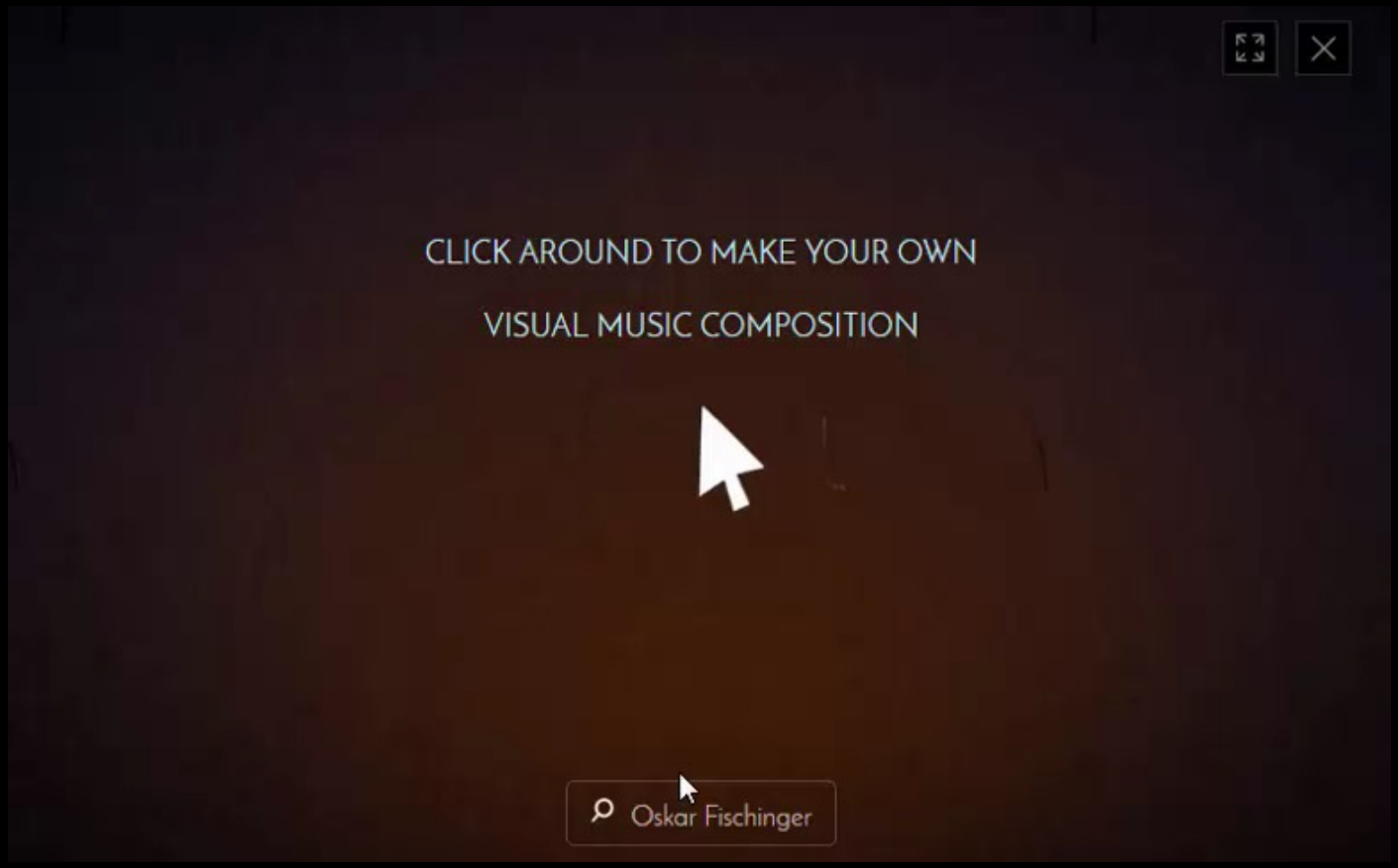
Processing Examples!

- Interactive Animations
  - Example: controlled particle system



Processing Examples!

- Interactive Animations
  - Example: Google Doodle (June 2017)



https://www.google.com/doodles/oskar-fischingers-117th-birthday

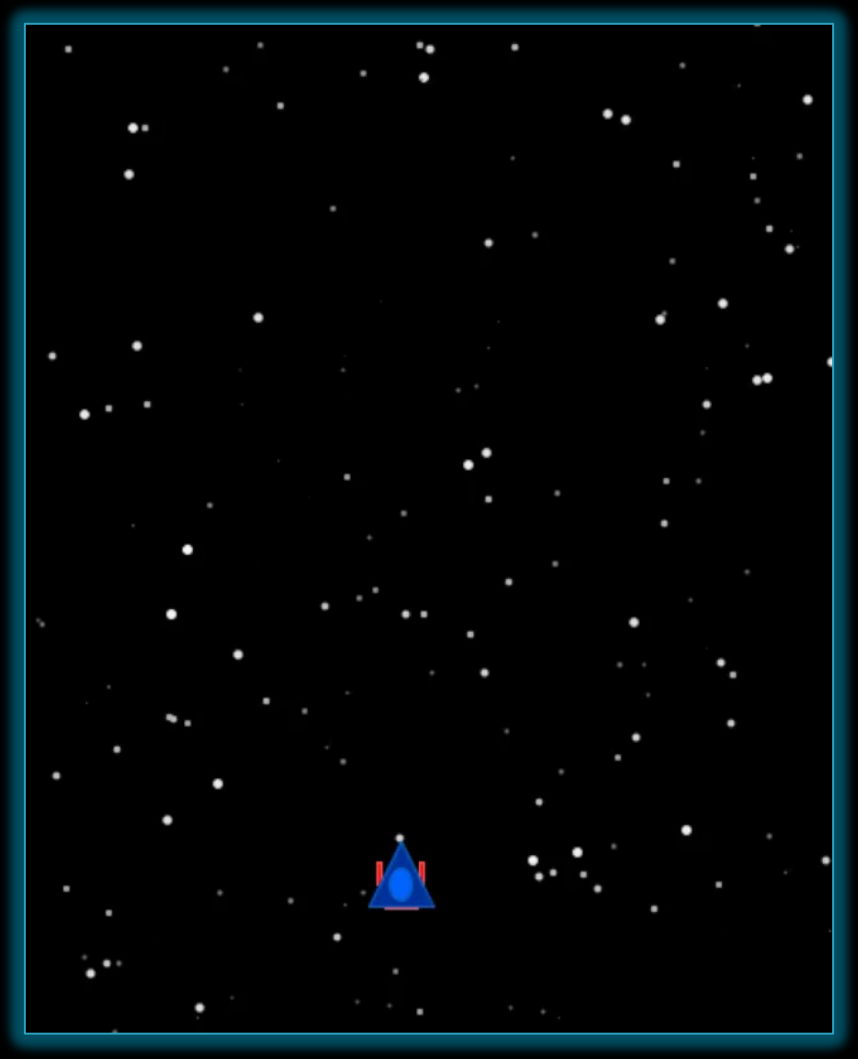
Processing Examples!

- Interactive Animations
  - Example: 2D Games

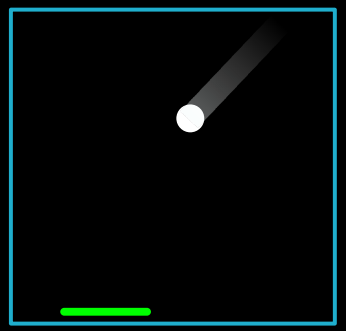
Game name: Toon Shooters 2

Example of things you will do!

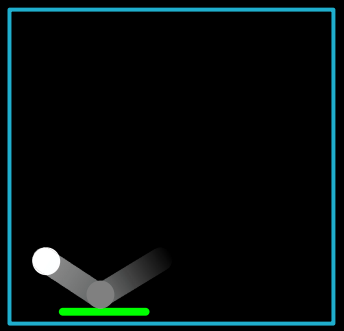
Example of things you will do!



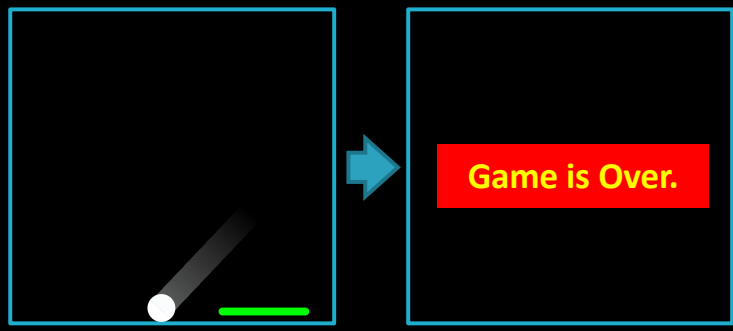
Example of things you will do!



Player needs to move the paddle to hit the ball up



When the ball is hit, score is incremented and ball speed increases



If ball touches bottom edge, game is over, and animation stops.

Why this Course is Important

- This course will make programming fun and relevant.
  - Our economy, health, and entertainment is dependent on software written by programmers.
  - We will learn to be creative programmers, so that we may create great software to be used by others.

- Important results:
  - ***Storyboarding*** – We will sketch our stories before programing them.
  - ***Algorithmic Thinking*** – We will learn how to solve problems by specifying precise sequences of actions.
  - ***Collaboration*** – We will program in teams of two to build interpersonal skills and increase our knowledge.
  - ***Processing and Java Languages*** – We will use Processing which is based on Java programming language – Java can be used in many areas including future computer science courses.

COSC 123

Computer Creativity

Getting Started with Processing

Okanagan

Slides courtesy of Dr. Abdallah Mohamed.

Key Points

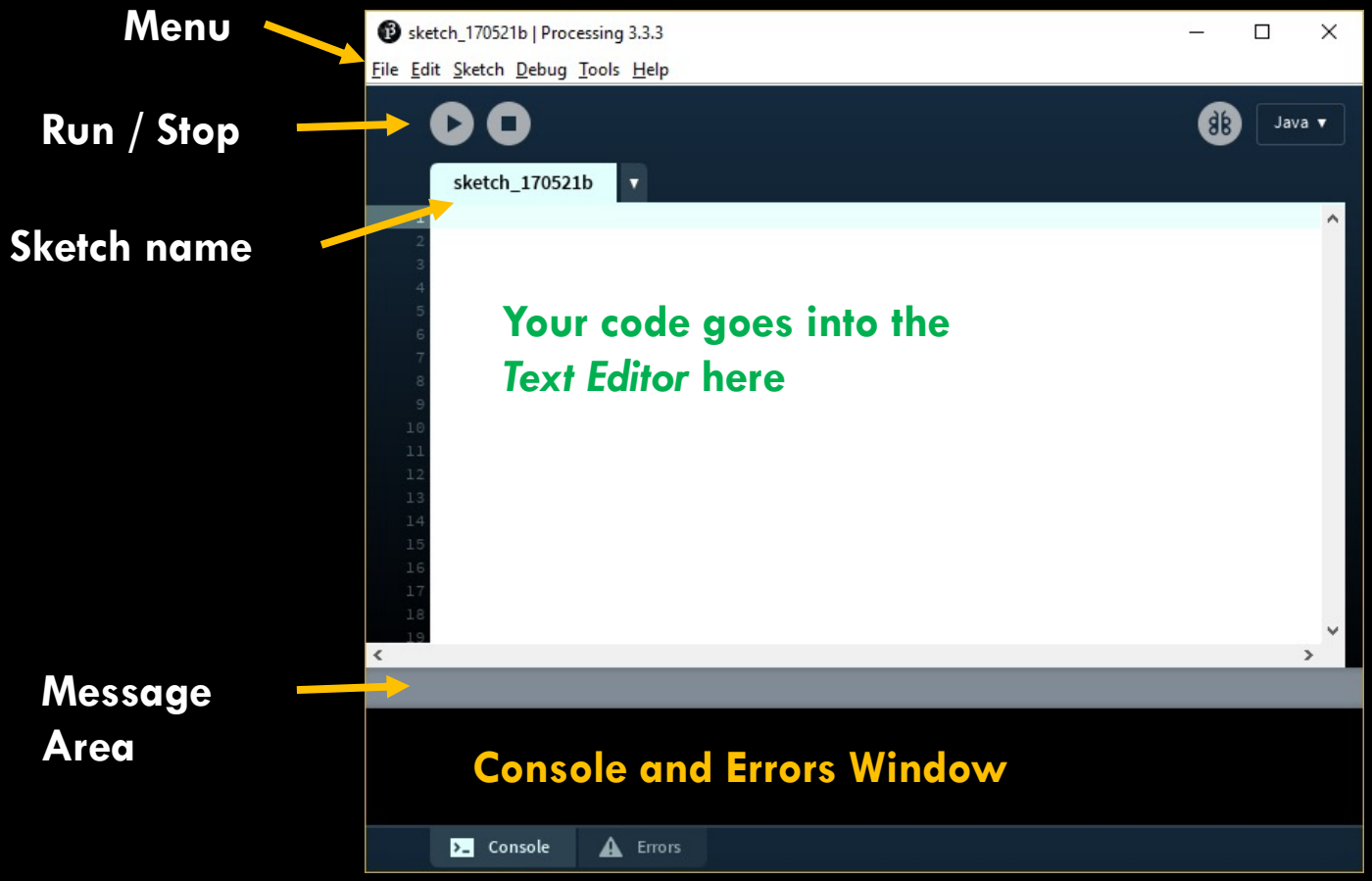
1) What is Processing
2) Experiment with the Processing Development Environment.
3) Printing on the console

The Processing Language

- Processing is a programming environment that aims to help create **visually oriented applications**, such as sketches, animations, and games.

- Processing consists of:
  - The Processing Development Environment (PDE).
    - The software we will use to write and run our code in this course.
    - Has a minimalist set of features suitable for developing small programs
  - The Processing core **API** and other libraries
    - A collection of functions (aka commands or methods) for performing the different actions in a program.
  - A language syntax identical to Java.
    - ***Processing is Java***, but with simpler syntax.
    - Processing was ported to other languages later (e.g. JS, Python).

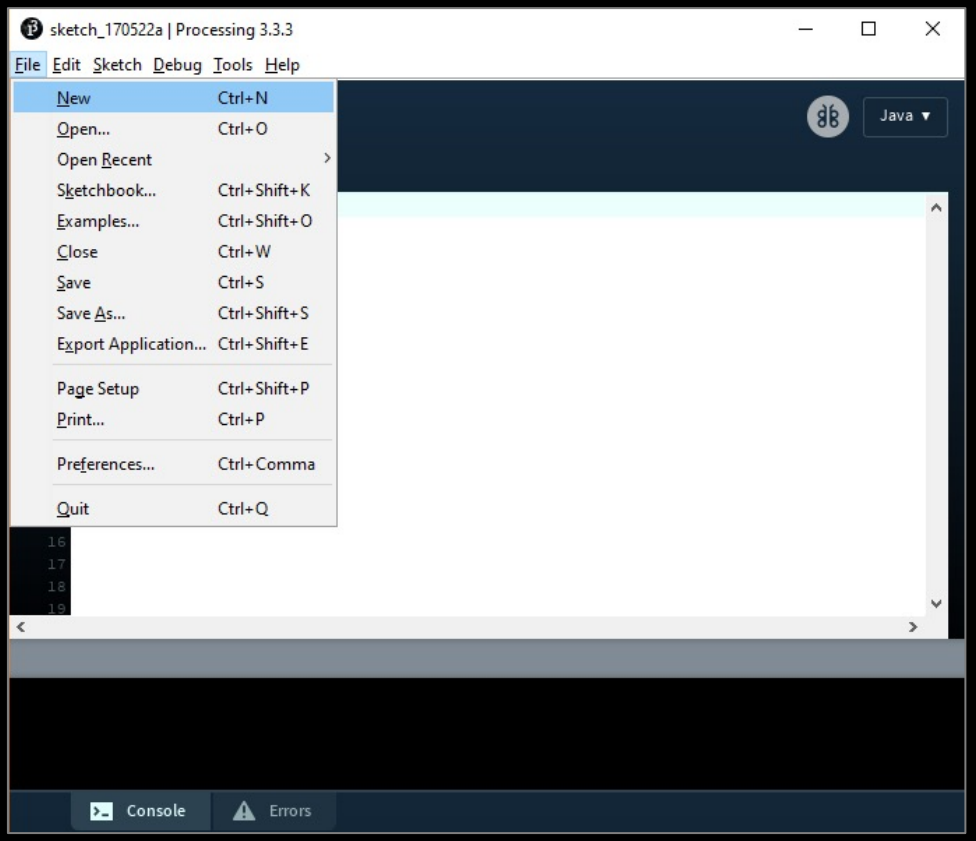
Processing Development Environment (PDE)



The code represents a sketch. Each sketch is actually a subclass of the `PApplet` Java class

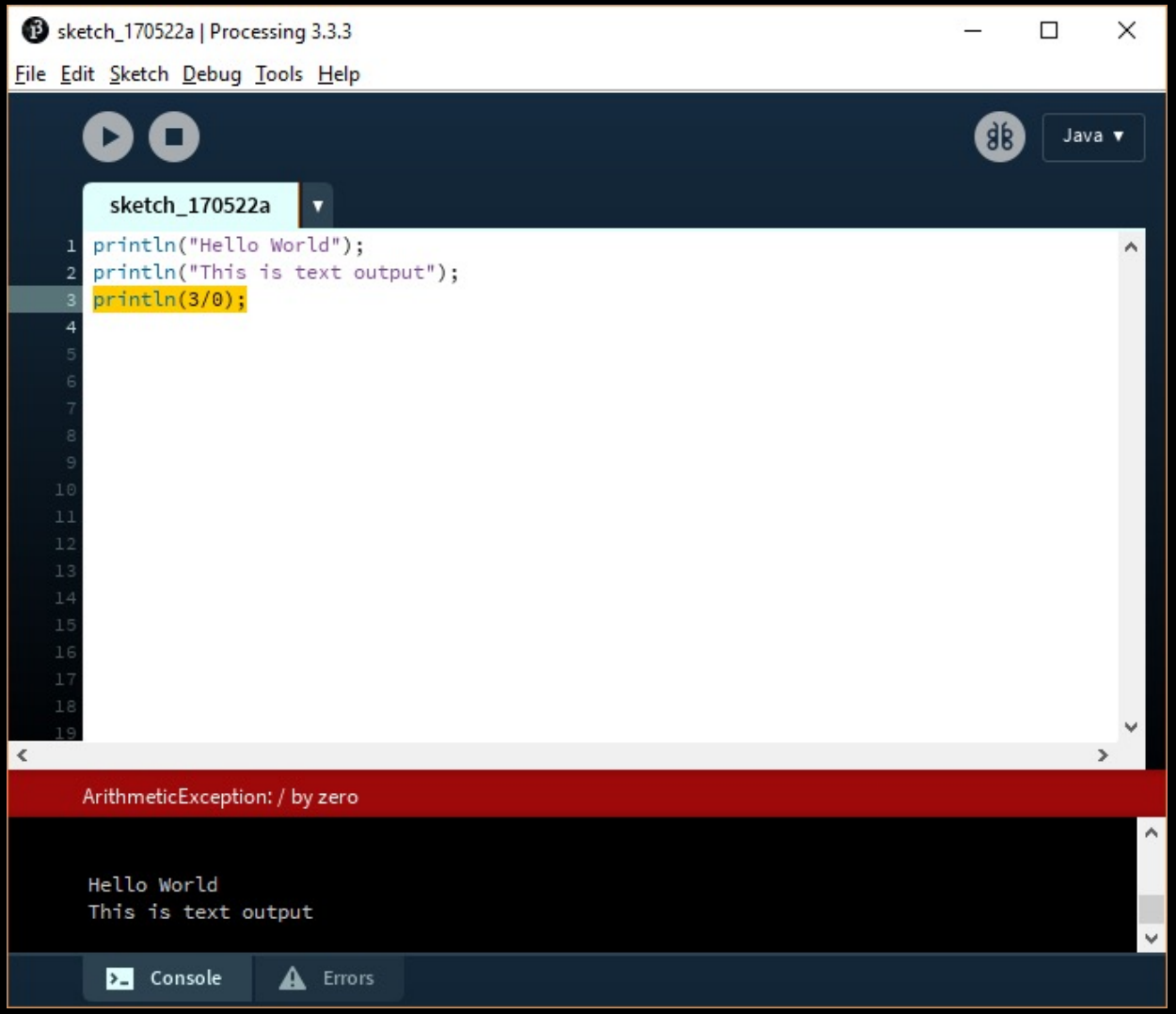
PDE: Creating and Running a Sketch

- 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.



PDE: The Console Window

- The console window displays
  1) Text output, e.g. when printing text using `print()` and `println()` functions.
  2) Error messages



Functions

- A ***function*** is a sequence of statements that performs a specific action.
  - Creating a function avoids repeating statements and allows for better code organization.
- A function must have a name. Whenever we want to perform the function's action, we need to call (invoke) the function by its name.
  - For example, to print something on the console, we write

```
println("Hello World");
```

- *Processing* comes with a library of **predefined functions** that may be used to perform different actions such as drawing shapes. To use these functions, you need to call their names with the appropriate parameters.
  - In Java, a function is also called a "method".

Exercise

Output Text to the Console

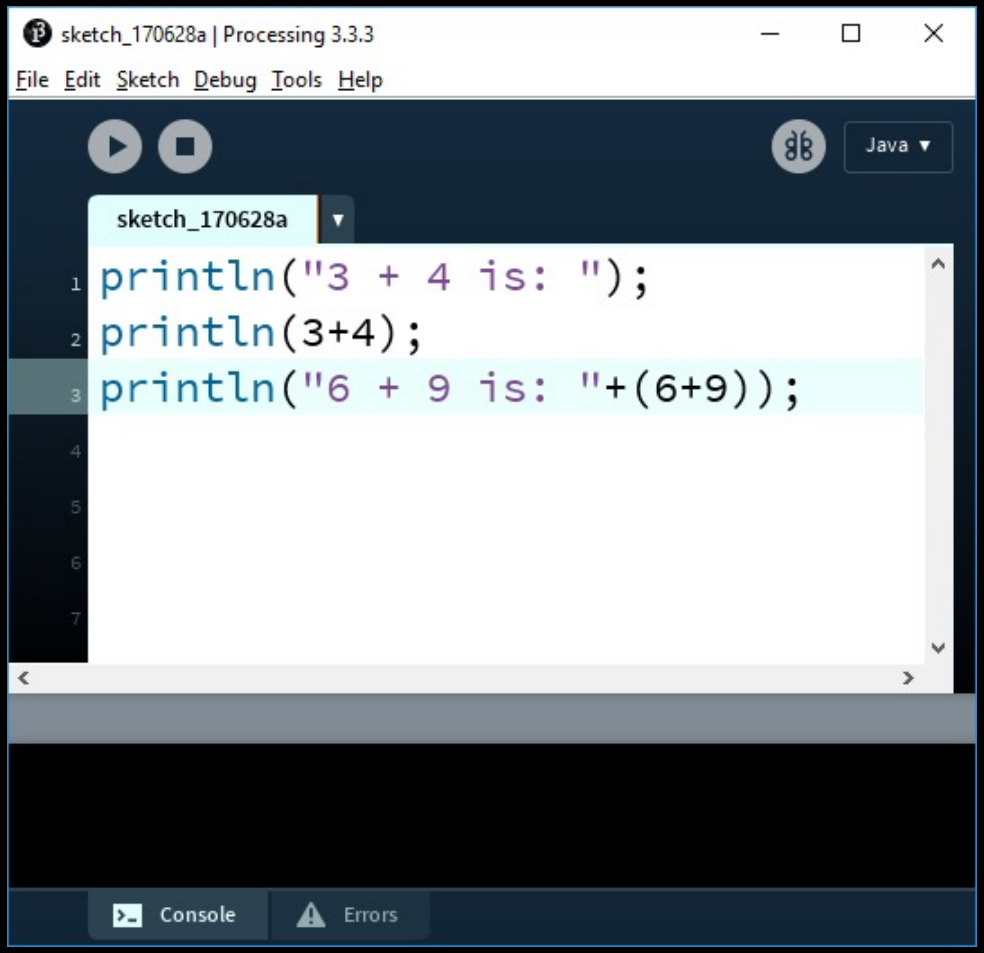
- Use `print()` and `println()` to display the following text on the console. Note that the number 6 on the second line is computed as 3*2.

```
This is a mathematical expression
3 x 2 = 6
Processing is mainly used for graphical applications, not console-based applications.
```

Exercise

Output Text to the Console

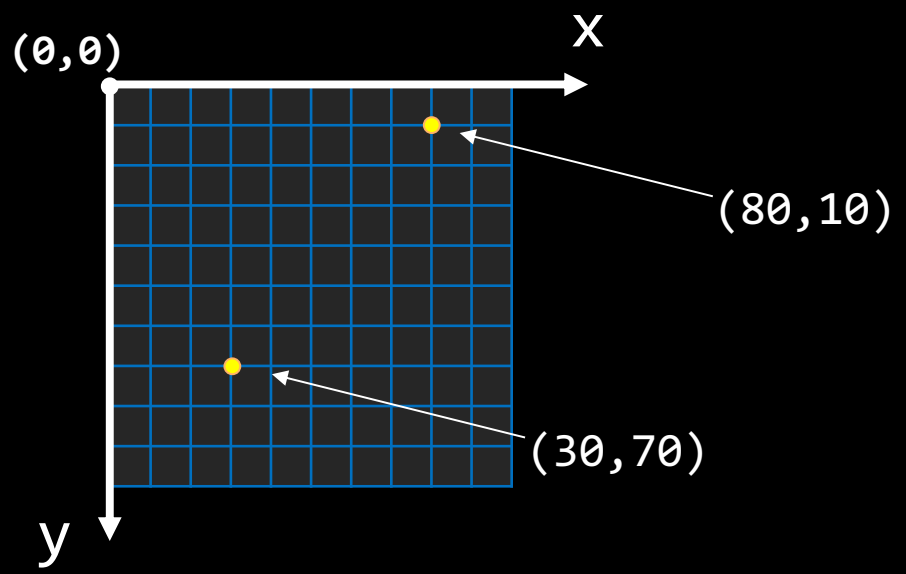
- What is the output of this program? Explain.



2D Coordinate System

The Coordinate System

- Drawing on the screen is done by specifying coordinates which refer to a location on the screen.
- By default
  - ***origin*** is the upper-left hand corner of the screen.
  - x coordinate is horizontal, getting bigger as we move right.
  - y coordinate is vertical, getting bigger as we move down.



Question

Coordinate system

Assume we have the 100x100 sketch shown below. Each small square is 10x10 pixels. What is the (x , y) location of the point?

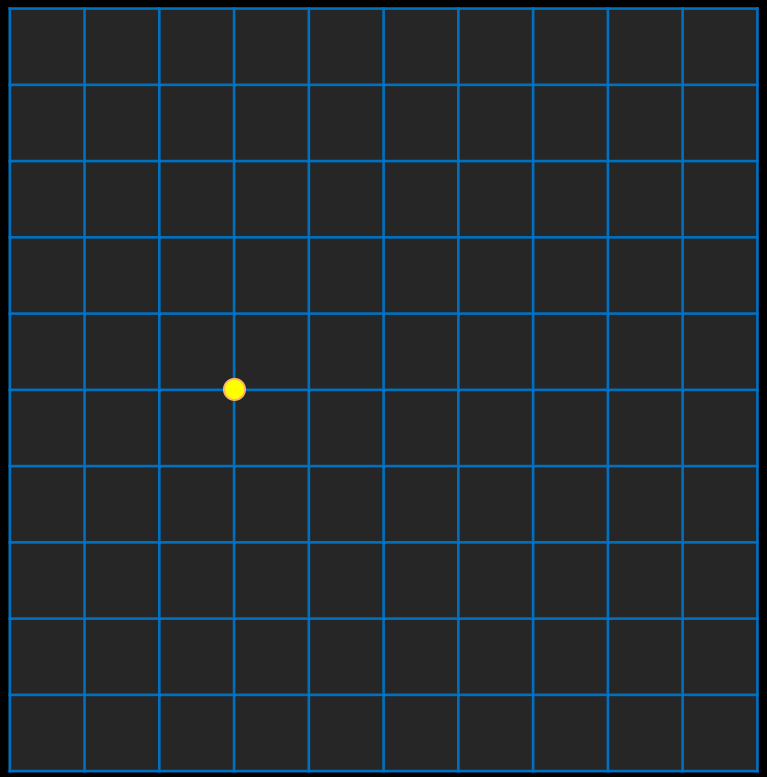
A. (30, 50)

B. (50, 30)

C. (3, 5)

D. (5, 3)

E. None of the above

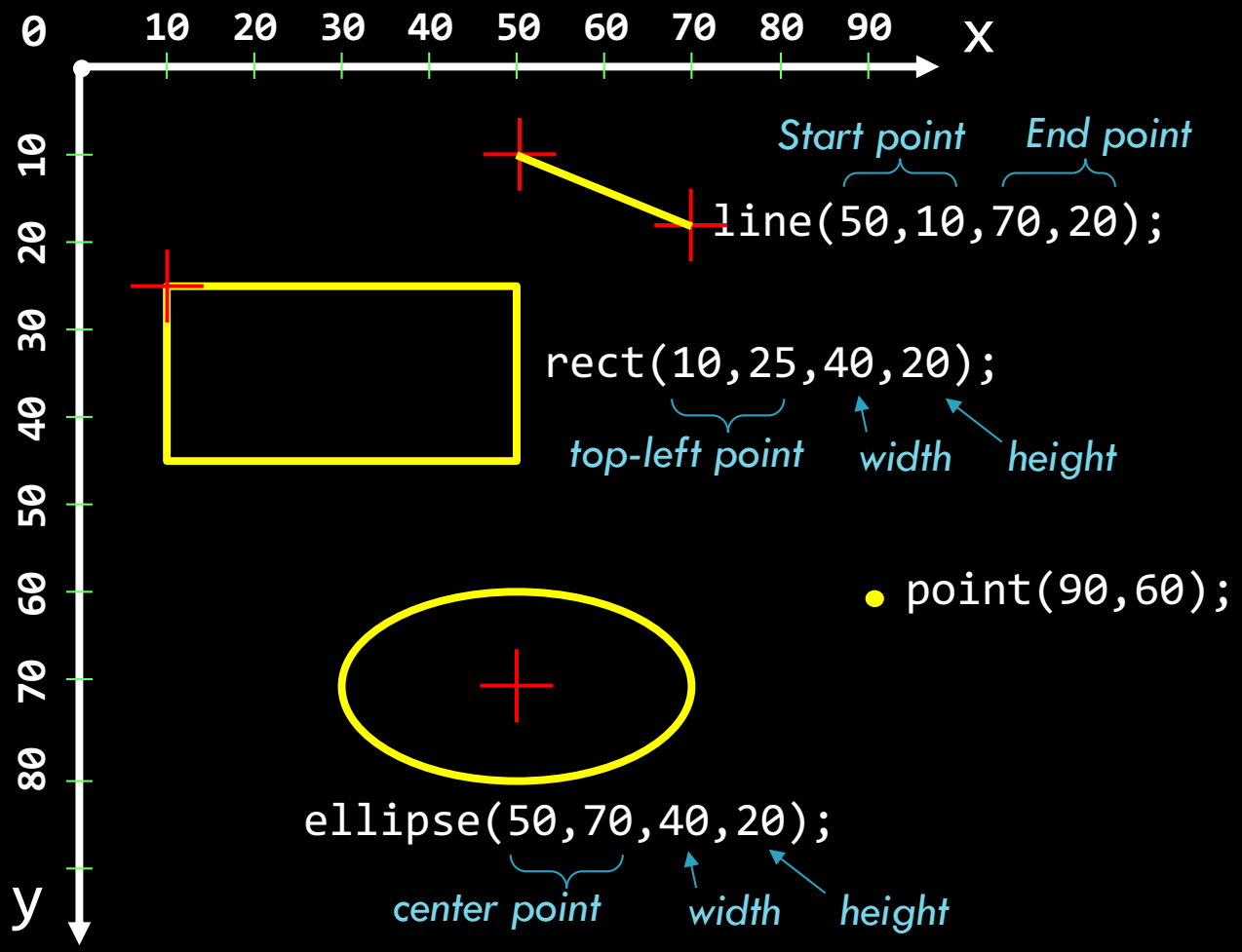


Drawing Primitive Shapes

- To draw shapes on the screen, we call the function that represent each shape with arguments representing the shape dimensions.

- Example of primitive shapes
  - Point: `point(90,60);`
  - Line: `line(50,10,70,20);`
  - Rectangle: `rect(10,25,40,20);`
  - Ellipse: `ellipse(50,70,40,20);`

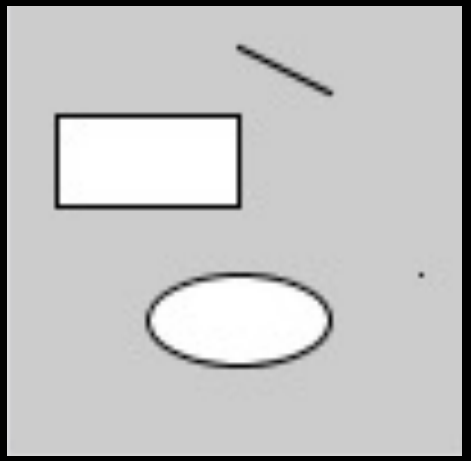
Drawing Primitive Shapes, cont'd



Drawing Primitive Shapes, cont'd

- Here is the Processing code and output

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



Computer Creativity

Notes

Processing Basics

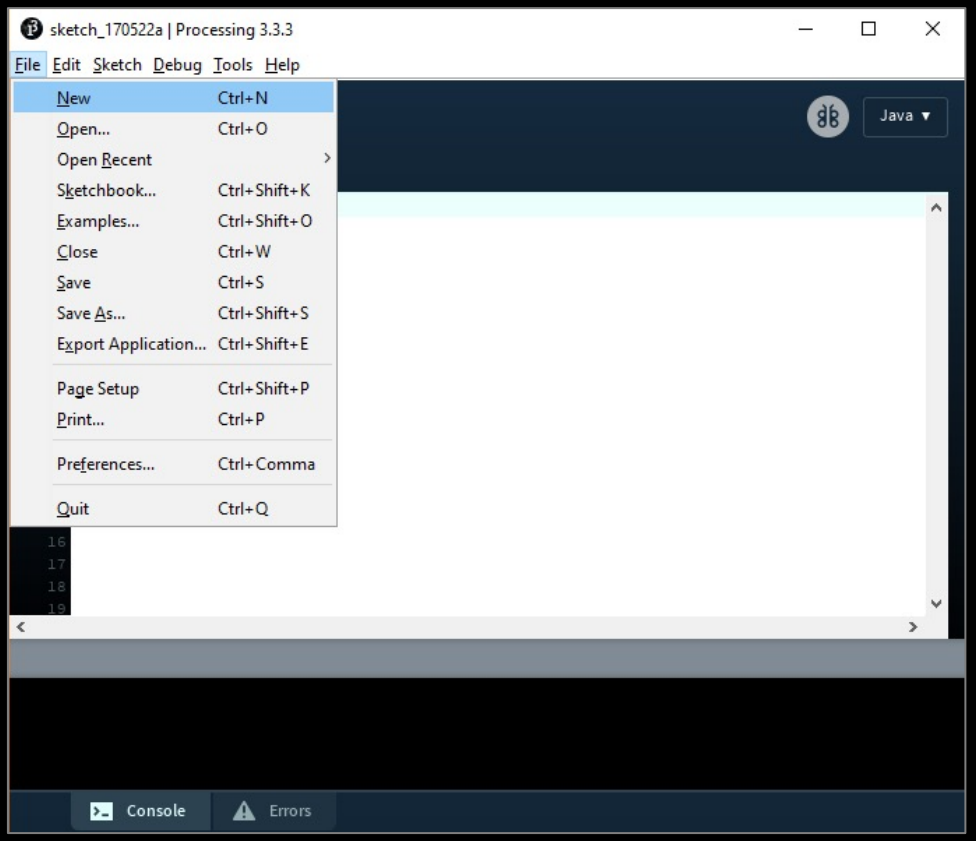
Slides courtesy of Dr. Abdallah Mohamed.

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.

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.

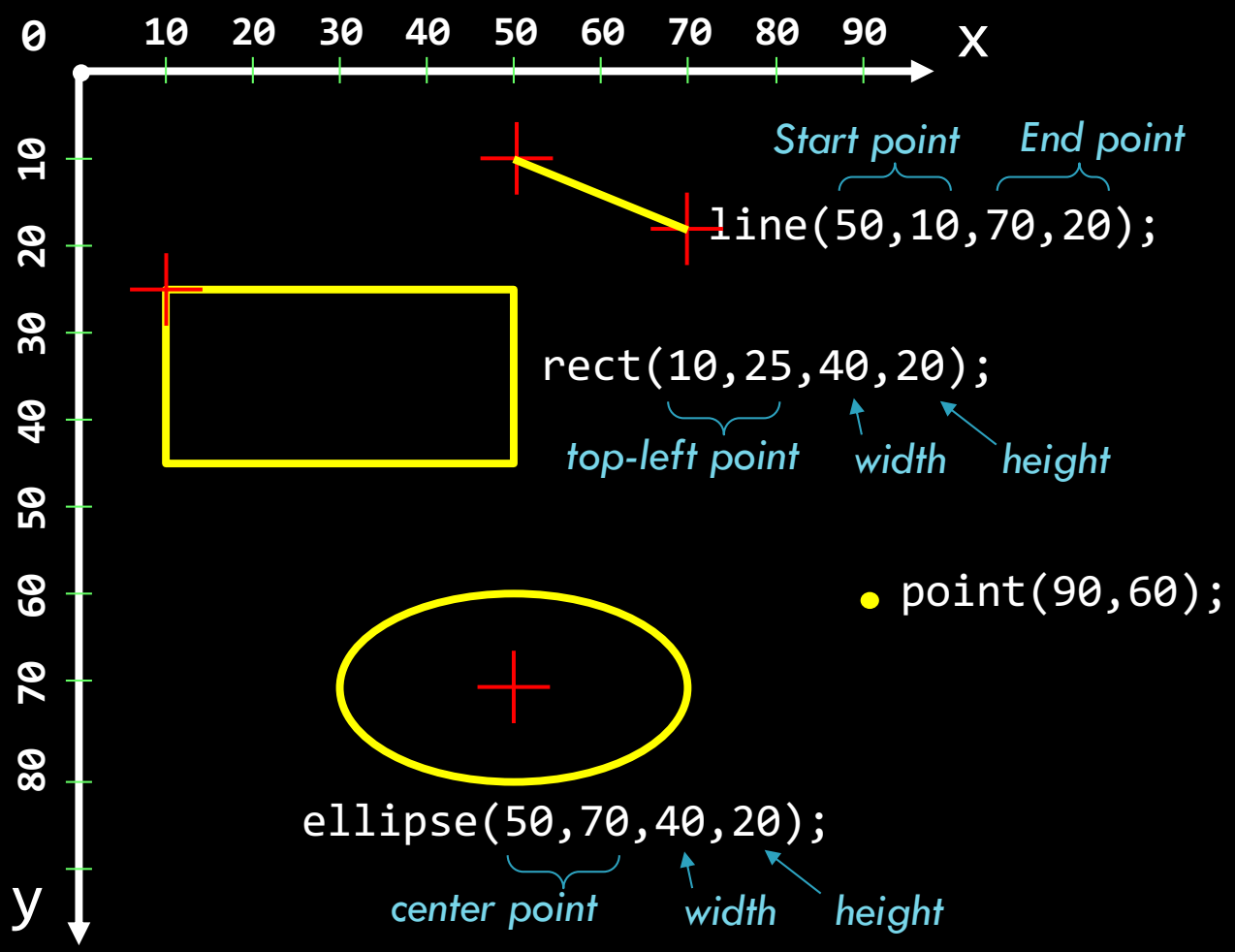


Primitive Shapes

- Example of primitive shapes
  - Point: `point(90,60);`
  - Line: `line(50,10,70,20);`
  - Rectangle: `rect(10,25,40,20);`
  - Ellipse: `ellipse(50,70,40,20);`

Function name *Parameters*

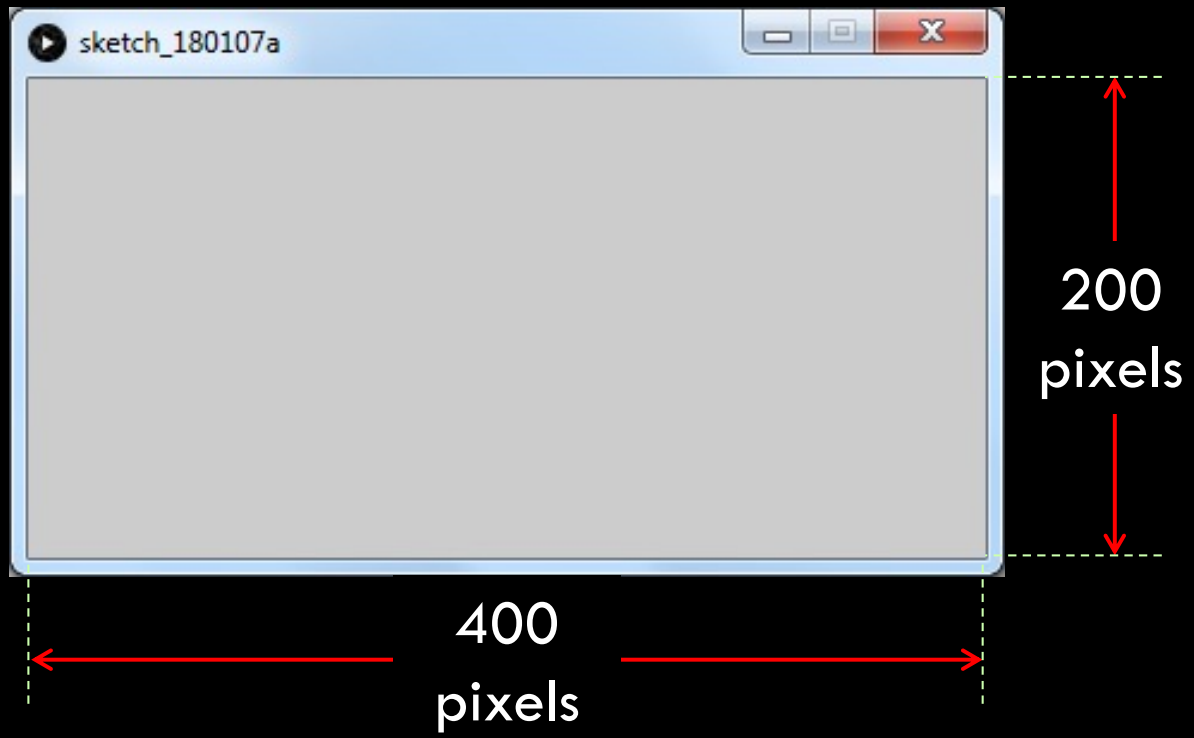
Drawing Primitive Shapes



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.

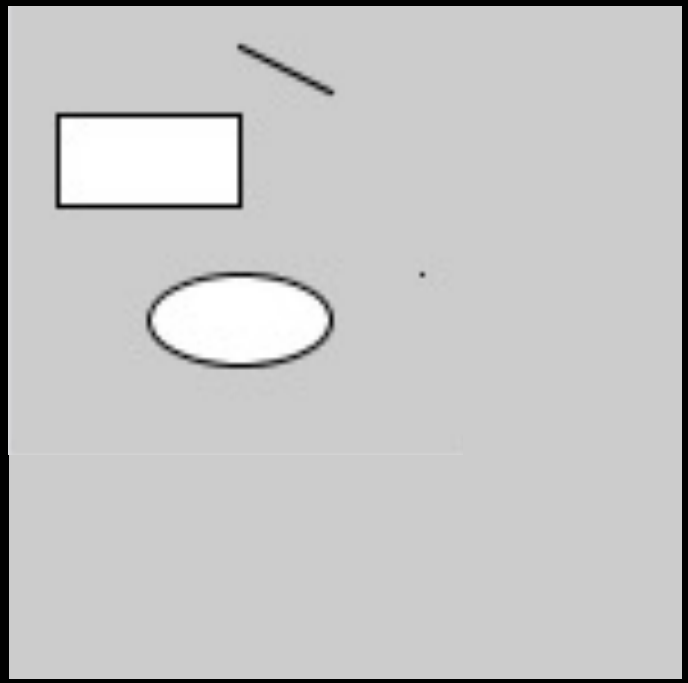
```
size(400,200);
```



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

Slides courtesy of Dr. Abdallah Mohamed.

Objectives

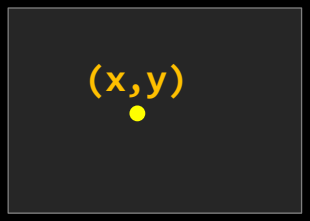
- These are some notes for you to work on outside of class. After finishing these notes, you should be able to:
  - Recognize and use primitive shape functions
    - point(), line(), rect(), ellipse(), quad(), triangle(), bezier()
  - Understand and specify shape ***coordinates***
    - i.e. specify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

Drawing Primitive Shapes

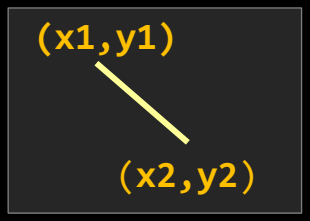
- You learned before how to draw some of the primitive shapes, namely: point, line, ellipse, and rectangle.

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

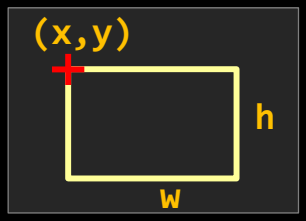
Primitive Shapes



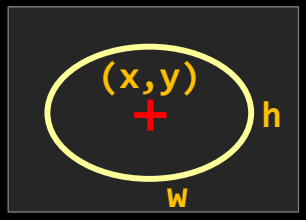
point(x,y)



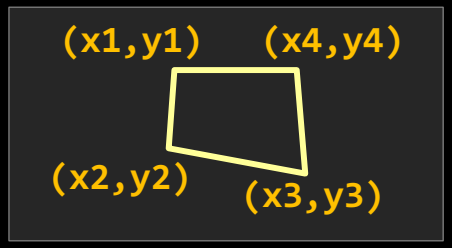
line(x1,y1,x2,y2)



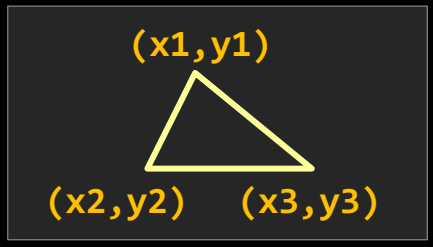
rect(x,y,w,h)



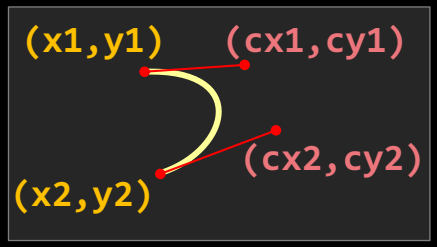
ellipse(x,y,w,h)



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)

Example

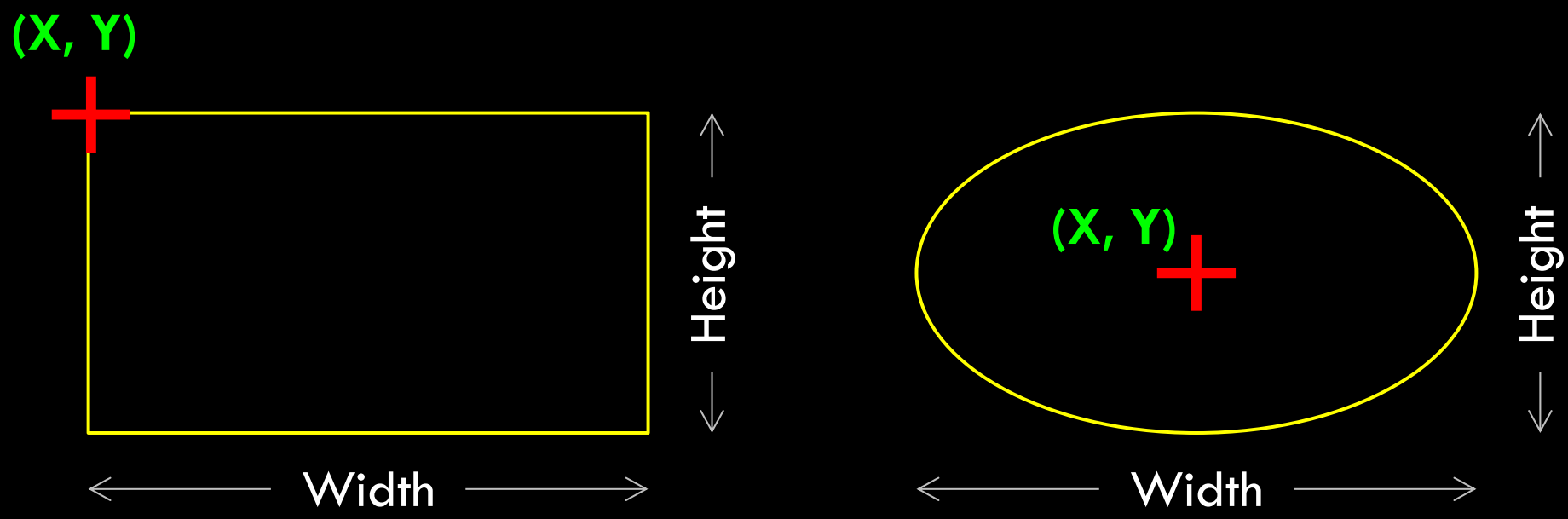
Primitive Shapes

```
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

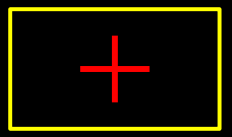
- The *default* coordinates for **`rect`** and **`ellipse`** are:

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

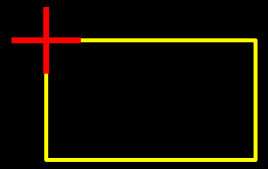


Specifying Shape Coordinates

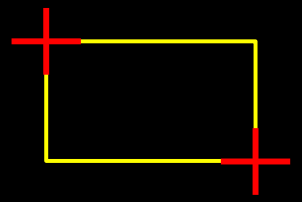
- Default coordinates can be explicitly set to one of three modes:
  - CENTER
    `(Center_X, Center_Y, Width, Height)`



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



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

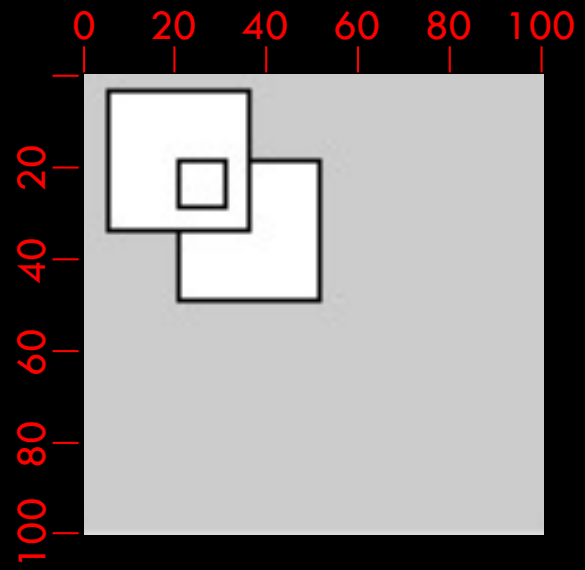


- The above applies to **`rect`** and **`ellipse`** but not necessarily to all shapes

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);
```



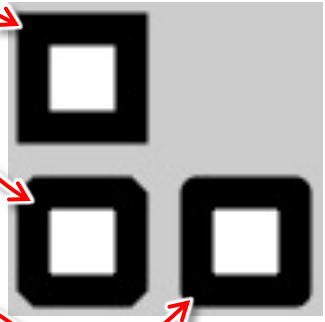
- *Question: Can you link each statement to the right shape?*

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);
```

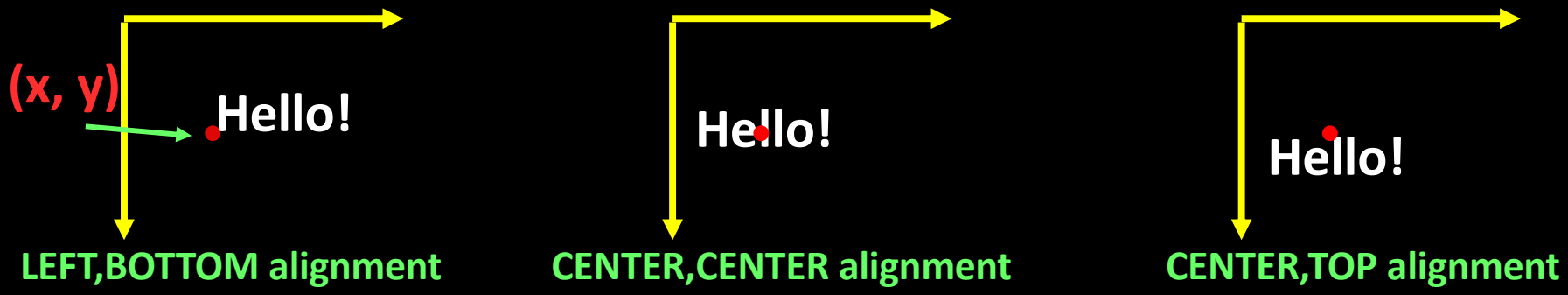
```
strokeWeight(10);
strokeJoin(MITER);
rect(10, 10, 30, 30);
strokeJoin(BEVEL);
rect(10, 60, 30, 30);
strokeJoin(ROUND);
rect(60, 60, 30, 30);
```



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);
```

Computer Creativity

Primitive Shapes, Text

Slides courtesy of Dr. Abdallah Mohamed.

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

Question

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

Question

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);

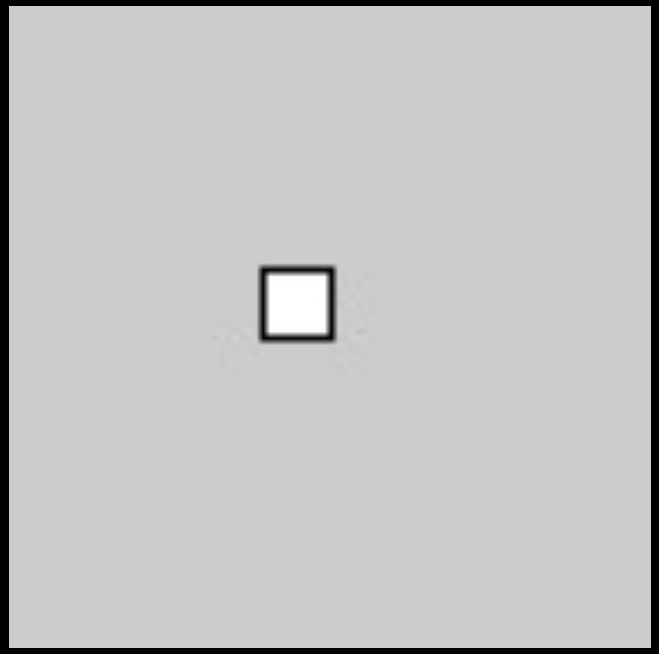
E. CENTER;

Question

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

Question

Stroke Attributes

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

A. width()

B. strokeWdith()

C. weight()

D. strokeWeight()

E. stroke()

Question

Text

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

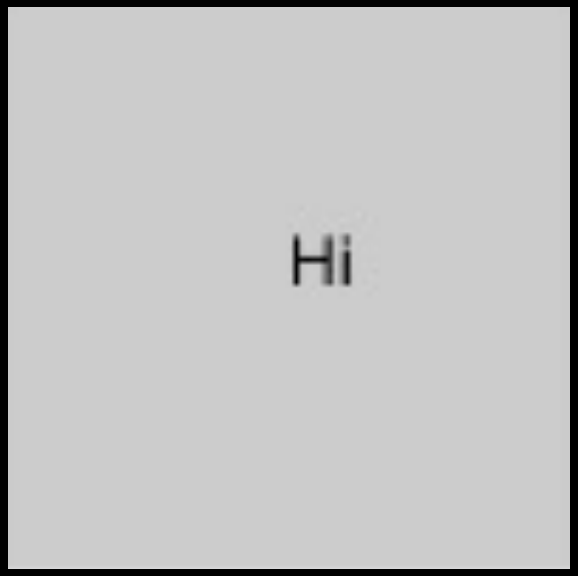
A. `write("Hi",50,50);`

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

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

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

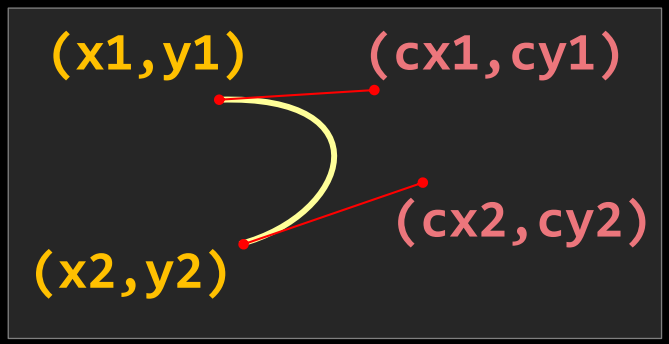
E. `drawText("Hi",50,50);`



Question

Bezier shapes

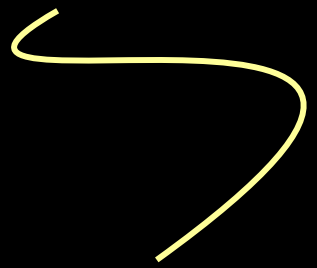
Have you understood how **bezier** function work?




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

A. Yes

B. No



Notes

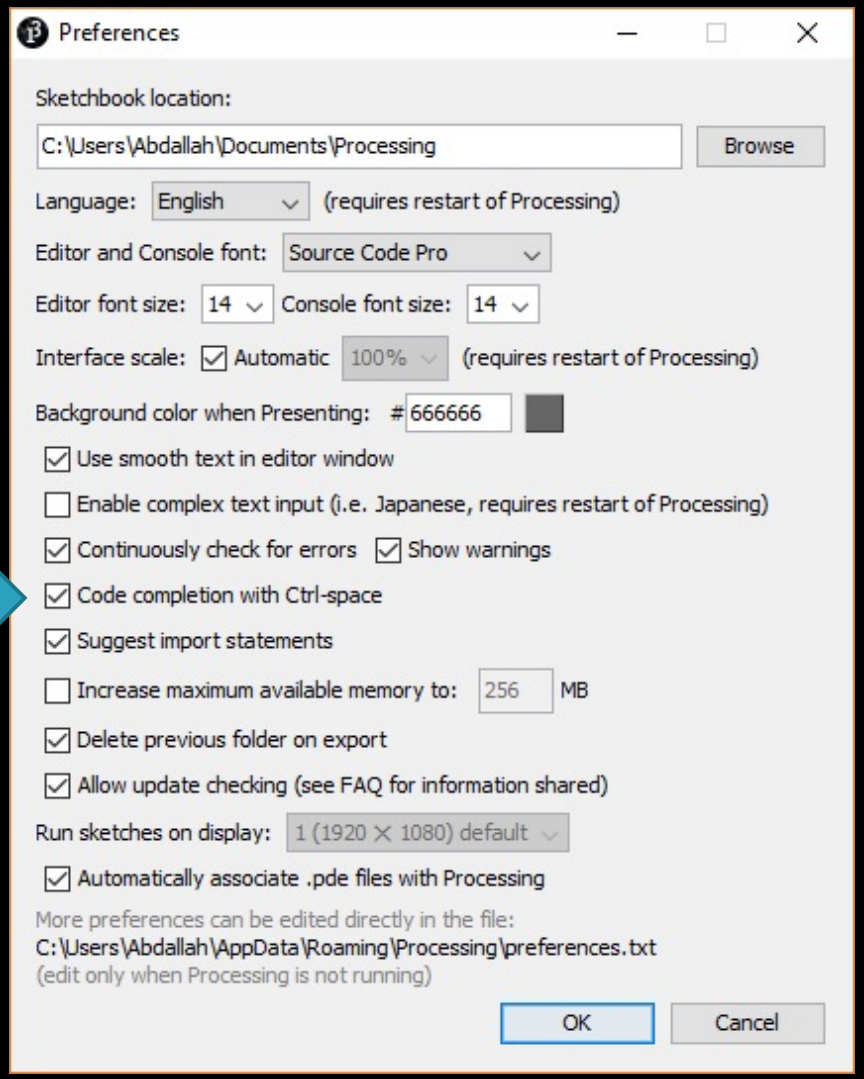
PDE Features

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.

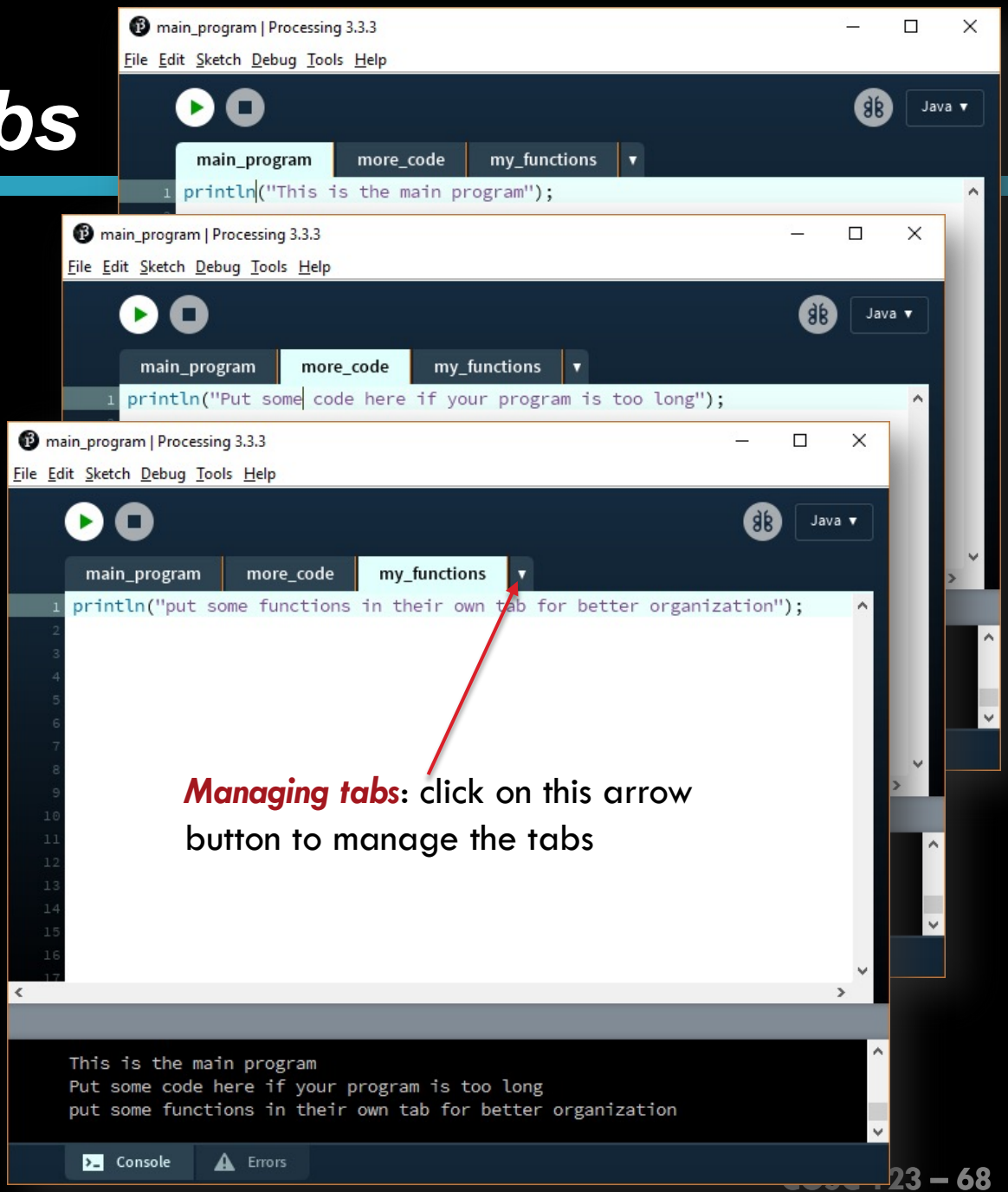
Code Completion

- 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.



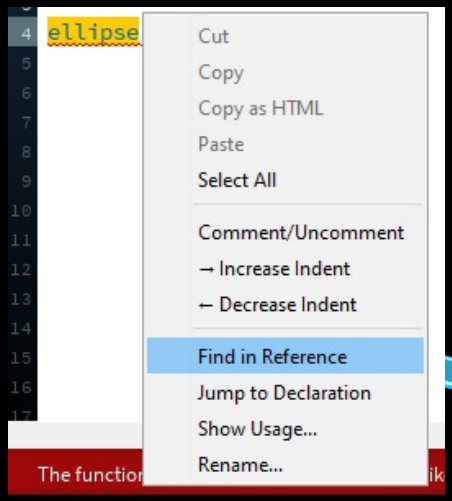
Sketchbook Tabs

- 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.



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.

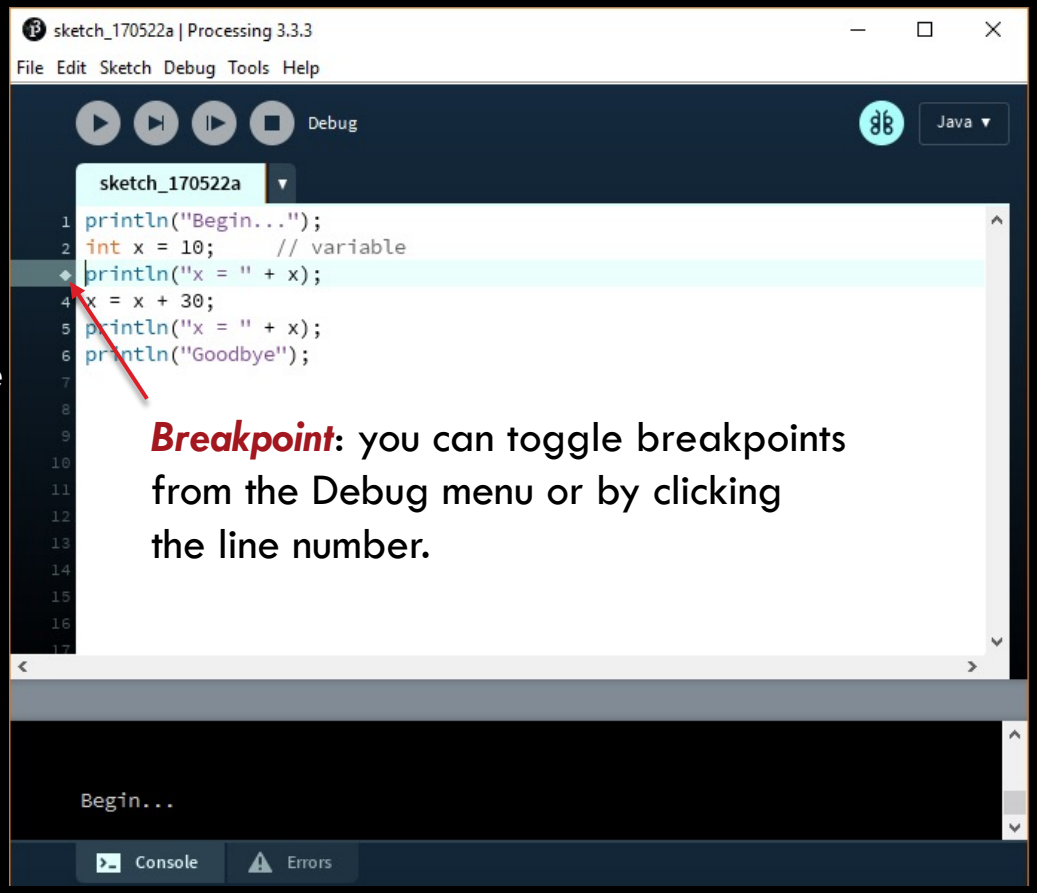


PDE Debugger

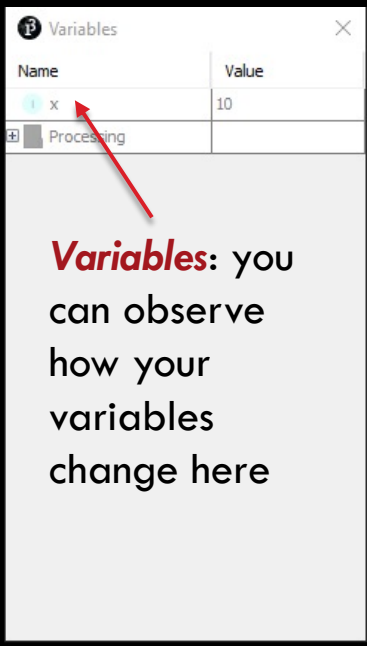
You can enable the debugging mode from the Debug menu or by clicking the Debugger icon.

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.**



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



***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.

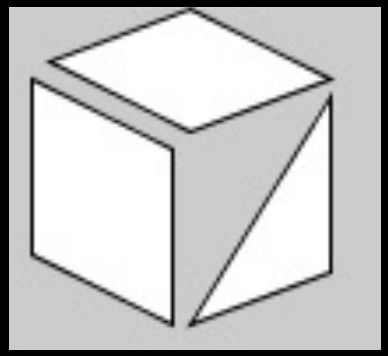
Lecture Activity 3

Accept the GH Classroom Link on the course website:

Canvas > Course Content > GitHub Classroom Links > Lecture Activities

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

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 **one character** of text.
- Hint: sketch your drawing on paper first, try to figure out the coordinates, then write code