*Corso di Laurea Magistrale in Design, Comunicazione Visiva e Multimediale - Sapienza Università di Roma*

# *Interaction Design*
## *A.A. 2017/2018*

10 – Advanced Concepts in Processing

Francesco Leotta, Andrea Marrella

Last update : 03/05/2018

# Libraries

- When we call a Processing function, such as `line(…)`, `background(…)` etc., we are calling a function that is available in the **core Processing library**.
- A **library** might consist of functions, variables, and objects.
  - Processing makes the assumption that the core Processing Library is **already imported** and there is no need to write an `import` statement for it.
- However, while the core library covers all the basics, for advanced functionality we have to **import specific libraries** that are not assumed at the top of your code.

`import` is a keyword to indicate that we are going to make use of a library named "`processing.core`". The wildcard ".*" indicates we want to access to everything in the library.

```
import processing.core.*;
```

## Built-in Libraries

- The full list of Processing built-in libraries is available at the following url:

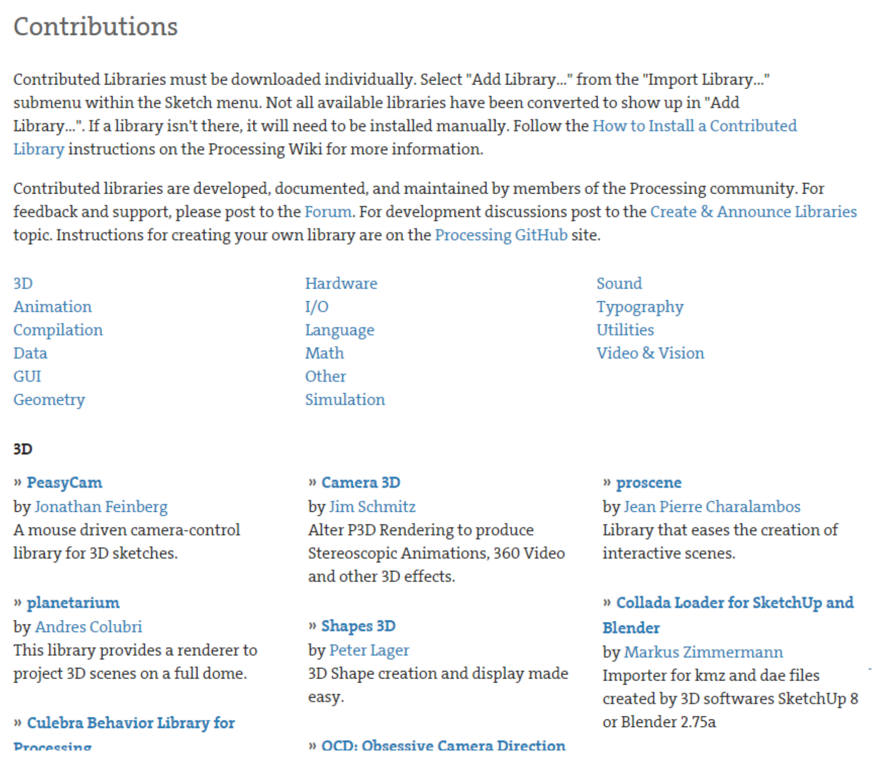
  https://www.processing.org/reference/libraries/

- For example, the following built-in libraries are covered by Processing:
  - **Video**: for capturing images from a camera and playing movie files.
    ```
    import processing.video.*;
    ```
  - **Sound**: for sound analysis, synthesis and playback.
    ```
    import processing.sound.*;
    ```
  - **Serial**: for sending data between Processing and an external device via serial communication.
    ```
    import processing.serial.*;
    ```
  - **Network**: For creating client and server sketches that can communicate across the internet.
    ```
    import processing.net.*;
    ```

# Contributed Libraries

- The world of third party (also known as "**contributed**") libraries for Processing provide capabilities ranging from packet sniffing, to physics simulations, to GUI controls.
- The full list of contributed libraries *compatible with Processing* is available at the following url:

[https://www.processing.org/reference/libraries/](https://www.processing.org/reference/libraries/)



## Contributions

Contributed Libraries must be downloaded individually. Select "Add Library..." from the "Import Library..." submenu within the Sketch menu. Not all available libraries have been converted to show up in "Add Library...". If a library isn't there, it will need to be installed manually. Follow the How to Install a Contributed Library instructions on the Processing Wiki for more information.

Contributed libraries are developed, documented, and maintained by members of the Processing community. For feedback and support, please post to the Forum. For development discussions post to the Create & Announce Libraries topic. Instructions for creating your own library are on the Processing GitHub site.

| | | |
|---|---|---|
| 3D | Hardware | Sound |
| Animation | I/O | Typography |
| Compilation | Language | Utilities |
| Data | Math | Video & Vision |
| GUI | Other | |
| Geometry | Simulation | |

**3D**

» **PeasyCam**
by Jonathan Feinberg
A mouse driven camera-control library for 3D sketches.

» **planetarium**
by Andres Colubri
This library provides a renderer to project 3D scenes on a full dome.

» **Culebra Behavior Library for Processing**

» **Camera 3D**
by Jim Schmitz
Alter P3D Rendering to produce Stereoscopic Animations, 360 Video and other 3D effects.

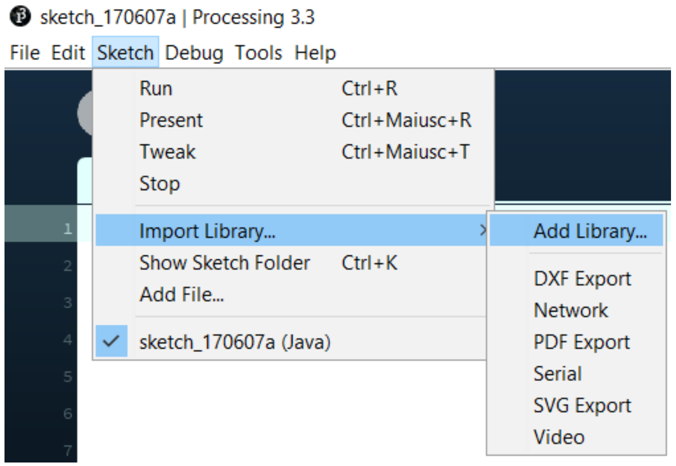
» **Shapes 3D**
by Peter Lager
3D Shape creation and display made easy.

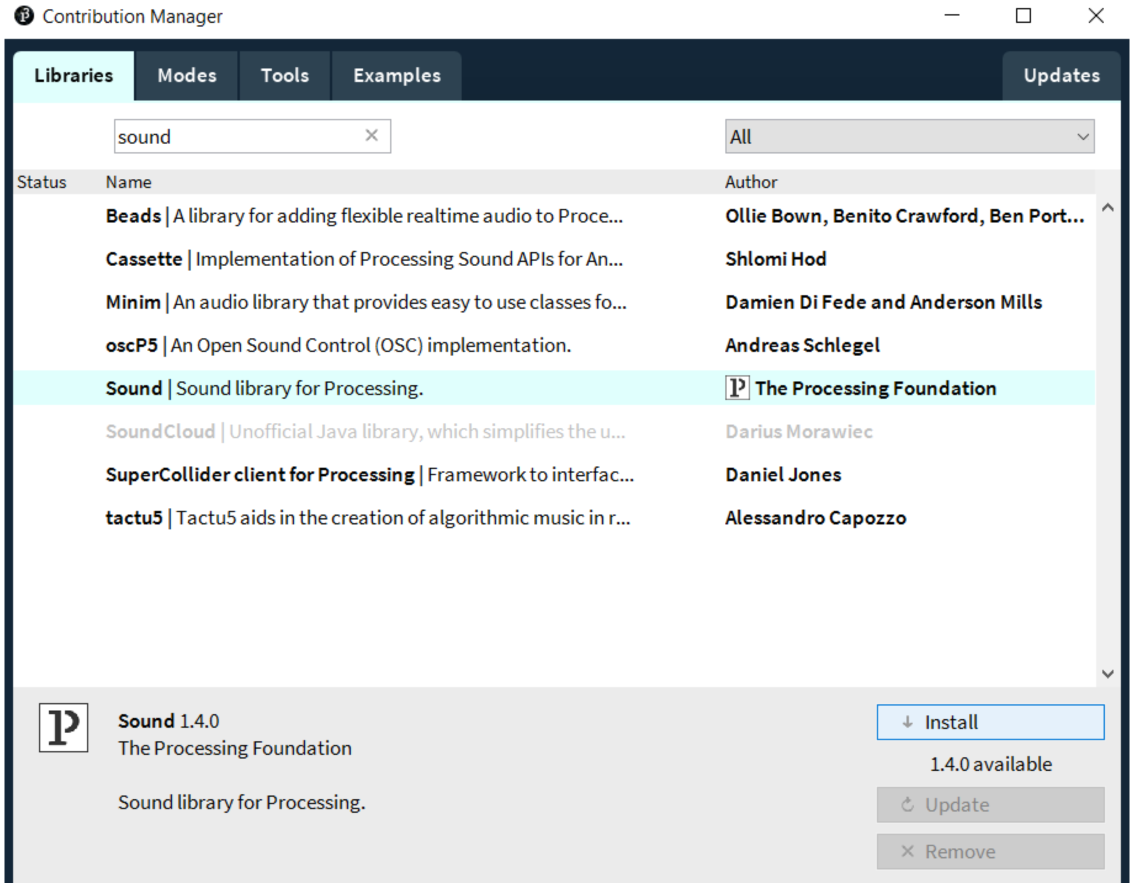
» **OCD: Obsessive Camera Direction**

» **proscene**
by Jean Pierre Charalambos
Library that eases the creation of interactive scenes.

» **Collada Loader for SketchUp and Blender**
by Markus Zimmermann
Importer for kmz and dae files created by 3D softwares SketchUp 8 or Blender 2.75a

# Install a Contributed Libraries





sound

| Status | Name | Author |
|---|---|---|
| | **Beads** \| A library for adding flexible realtime audio to Proce... | **Ollie Bown, Benito Crawford, Ben Port...** |
| | **Cassette** \| Implementation of Processing Sound APIs for An... | **Shlomi Hod** |
| | **Minim** \| An audio library that provides easy to use classes fo... | **Damien Di Fede and Anderson Mills** |
| | **oscP5** \| An Open Sound Control (OSC) implementation. | **Andreas Schlegel** |
| | **Sound** \| Sound library for Processing. | **The Processing Foundation** |
| | SoundCloud \| Unofficial Java library, which simplifies the u... | Darius Morawiec |
| | **SuperCollider client for Processing** \| Framework to interfac... | **Daniel Jones** |
| | **tactu5** \| Tactu5 aids in the creation of algorithmic music in r... | **Alessandro Capozzo** |

**Sound** 1.4.0
The Processing Foundation

Sound library for Processing.

1.4.0 available

## Strings

- The **String** class is built into the Processing environment for storing and manipulating text.

- We have already dealt with Strings before whenever we have printed some text to the message window or loaded an image from a file.

```
println( "printing some text to the message window!");

PImage img = loadImage("filename.jpg");
```

- In order to learn the details about all the built-in variables and functions available for Strings, Processing includes documentation in its reference:

  http://www.processing.org/reference/String.html

## What is a String?

- A String is nothing more than a **list of characters in between quotes**.

```
String sometext = "Type characters between quotes!";
```

- Nevertheless, this is only the data of a String. We must remember that a String is an **object** with **functions**. Let's see some of them!

- A first useful method is `length()`.
  - This is easy to confuse with the `length` property of an array.
  - However, when we ask for the length of a String object, **we must use the parentheses** since we are calling a function called `length()` rather than accessing a property called length.

```
String message = "This String is 34 characters long." ;
println(message.length());
```

It will print on the console: “34”

# UpperCase, LowerCase and Equality

- We can also change a String to <u>all uppercase</u> (or <u>lowercase</u>) using the `toUpperCase()` or `toLowerCase()` functions.

```
String message = "CiaO";
String uppercase = message.toUpperCase();
String lowercase = message.toLowerCase();
println(uppercase); //It will print to the console: CIAO
println(lowercase); //It will print to the console: ciao
```

- If we want to compare Strings (for example, in a conditional *if*), instead of using == we must use the function `equals(…),` which returns a boolean value that is TRUE if two strings contain the same list of characters.

```
if(uppercase.equals(lowercase)) {…}
```

- If we want to perform a comparison that is not case sensitive, we can use the function `equalsIgnoreCase(…)`

# Concatenation

- Another feature of String objects is **concatenation** for joining two Strings together.
- Strings are joined with the plus (+) operator.
  - The plus operator (+), of course, usually means add in the case of numbers. When used with Strings, it means **join**.

```
String helloworld = " Hello " + " World ";
```

- Variables can also be brought into a String using concatenation.

```
int x = 10;

String message = "The value of x is: " + x;
```

It will print on the console: “The value of x is 10”

# Splitting Strings

- The `split()` function separates a longer String into an **array of Strings**, based on a ***split character*** known as the ***delimiter***.
  - It takes two arguments, the String to be split and the delimiter.
  - The delimiter can be a single character or a String.

```
// Splitting a String based on spaces
 String word = "Ciao mamma, come stai?" ;
 String[] list = split(word, " " );
 printArray(list);


// Splitting a String based on spaces
 String word = "Ciao mamma, come stai?" ;
 String[] list = split(word, ',' );
 printArray(list);
```

The delimiters are " " and ','

The function `printArray(…)` prints the content of the array.

In the first case:
    Ciao
    Mamma,
    Come
    Stai?

In the second case:
    Ciao Mamma
    come stai?

# Displaying Text

- The easiest way to display a String is to print it in the message window with the function `println(…)`
- While this is valuable for debugging, it is not going to help our goal of displaying text for a user.
  - To place text on screen, we have to follow a series of simple steps.
- **STEP 1**: **Declare an object of type PFont.**

```
PFont f;
```

- **STEP 2**: **Specify the font by referencing the name and the size of the font.**

```
f = createFont("Georgia",16);
```

  - This should be done **only once** in `setup()`
  - For a list of available fonts, you can use `printArray(PFont.list());`

# Displaying Text

- **STEP 3: Specify the font using `textFont(…)`**
  - It takes one or two arguments, the font variable and the font size, which is optional. If you do not include the font size, the font will be displayed at the size originally loaded.

```
textFont(f,16);
```

- **STEP 4: Specify a font color using `fill(…)`**

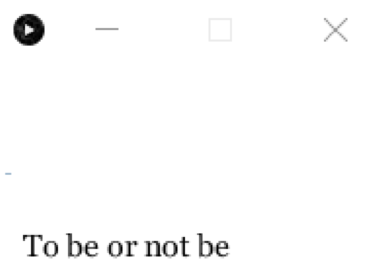
```
fill(0);
```

- **STEP 5: Call the `text(…)` function to display text.**
  - This function takes three arguments: the text to be displayed, and the x and y coordinate to display that text.
  - By default, text is LEFT-ALIGNED

```
text( "Mmmmm... Strings... " ,10,100);
```

# The Complete Example

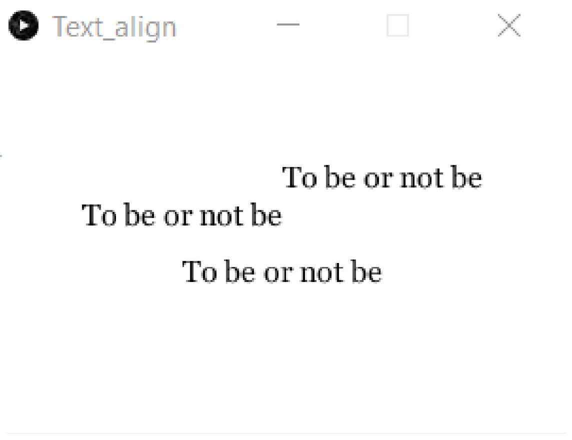
```
PFont f; // STEP 1: Declare PFont variable
void setup() {
 size(200,200);
 f = createFont("Georgia",16); // STEP 2: Load Font
}
void draw() {
 background(255);
 textFont(f,16); // STEP 3: Specify font to be used
 fill(0); // STEP 4: Specify font color
 text("To be or not be",10,100); // STEP 5: Display Text
}
```



```
To be or not be
```

# Aligning Text

```
PFont f;
void setup() {
 size(300,200);
 f = createFont("Georgia",16);
}
void draw() {
 background(255);
 textFont(f,16);
 fill(0);
 textAlign(LEFT);
 text("To be or not be",width/2,70);
 textAlign(RIGHT);
 text("To be or not be",width/2,90);
 textAlign(CENTER);
 text("To be or not be",width/2,120);
}
```

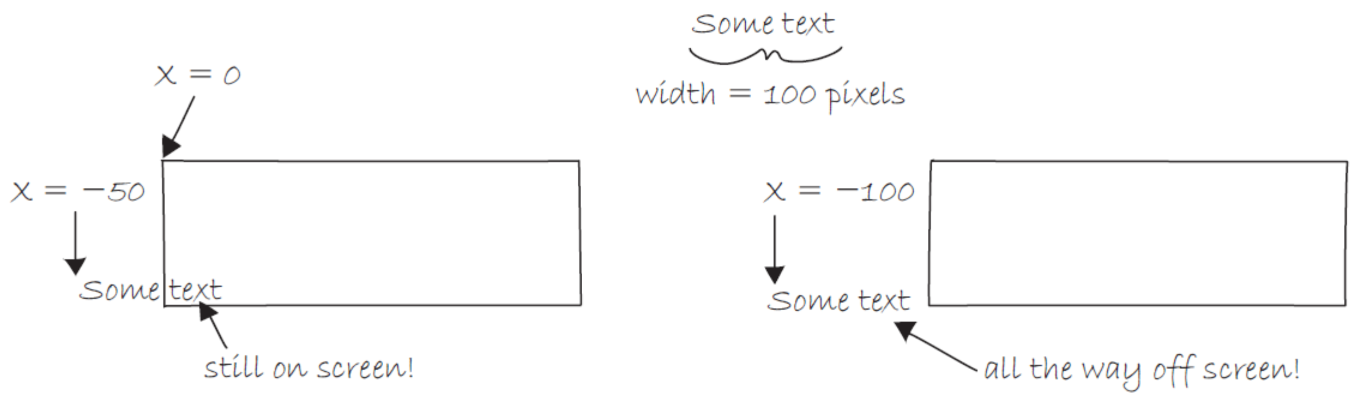


To be or not be
To be or not be
To be or not be

The function `textAlign(…)` specifies RIGHT, LEFT, or CENTER alignment for text .

# An example of text animation

- Let's say we want to create a ***news ticker***, where text scrolls across the bottom of the screen from left to right.
  - When a news headline leaves the window, a further news reappears on the right-hand side and scrolls again.
  - If we know the x location of the beginning of the text and we know the width of that text, we can determine when it is no longer in view.



- We can use the function `textWidth()`, which calculates and returns the width of any character or text string.

# An example of text animation

```
String[] headlines = {"To be or not to be:",
                      "that is the question",
                      "W.Shakespeare"};
PFont f;
float x;
int index = 0;

void setup() {
 size(200,200);
 f = createFont("Georgia",16);
 x = width;
}
```

To start, we declare an array called `headlines` that contains the news to visualize. Then, we declare font and x location variables, initializing them in `setup()`.

Initializing headline off-screen to the right.

## An example of text animation

```
void draw() {
 background(255);
 fill(0);
 textFont(f,16);
 textAlign(LEFT);
 text(headlines[index],x,40);
 x = x-3;
 float w = textWidth(headlines[index]);

if(x<-w) {
   x = width;
   index = index + 1;
   if(index == headlines.length) {
    index=0;
   }
 }
 }
```

The i-th news in the array is shown.

Decrement x to simulate the movement from right to left.

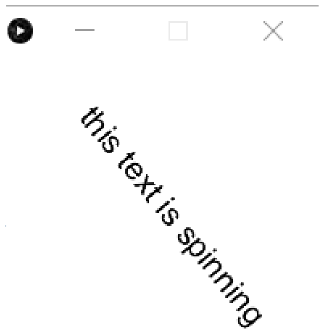
Calculate the width of the String representing the i-th news visualized.

To check if the text reached the left side of the screen, **we can not simply ask**: **is x less than 0?**

Since the text is left-aligned, when x equals zero, it is still viewable on screen. Instead, the text will be invisible when x is less than 0 minus the width of the text.

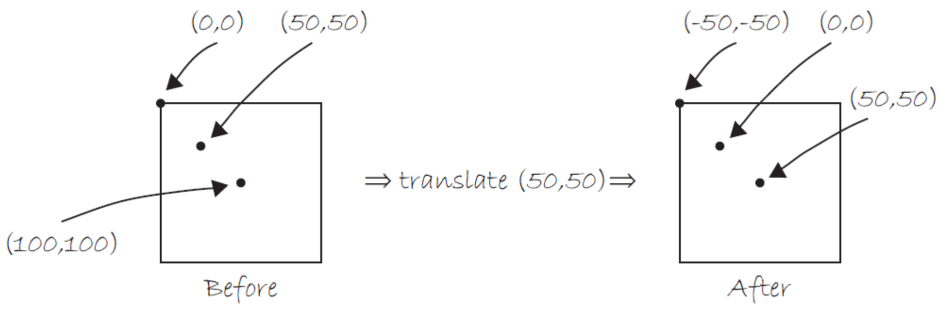
When that is the case, we reset x back to the right-hand side of the window, that is, `width`. If we have visualized all the news, we reset the index of array to show the first news.

# Translating and Rotating Text



```
PFont f;
String message = " this text is spinning";
float angle;
void setup() {
  size(200,200);
  f = createFont("Arial",20);
}
void draw() {
 background(255);
 fill(0);
 textFont(f);
 // Translate to the center
 translate(width/2,height/2);
 rotate(angle); // Rotate by angle
 textAlign(CENTER) ;
 text(message,0,0);
 angle += 0.05; // Increase rotation
}
```

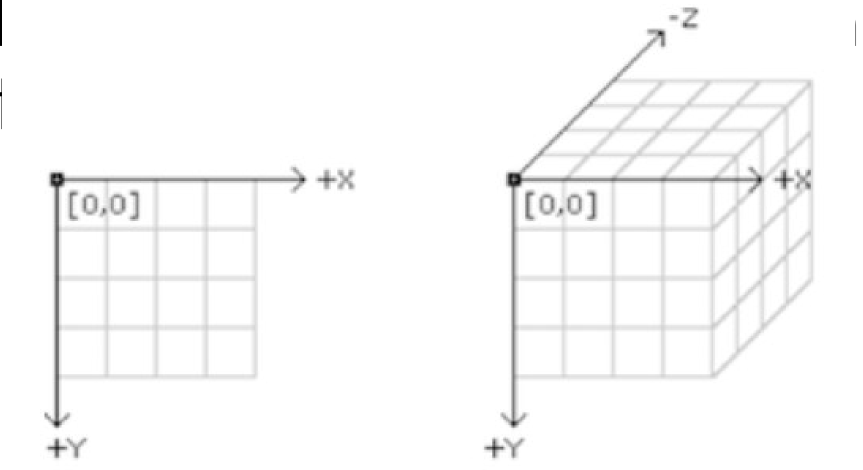
The function `translate(...)` moves the origin point—(0,0)— of a specific amount within the display window. The x parameter specifies left/right translation, the y parameter specifies up/down translation.



Rotate the center of an object of a specific angle.

## Processing and 3D

- Processing offers an easy support to 3D interfaces
- It can be accessed using P3D as last argument of size() function, e.g., size(400, 400, P3D)
- Drawing in 3I t of view on coordinat



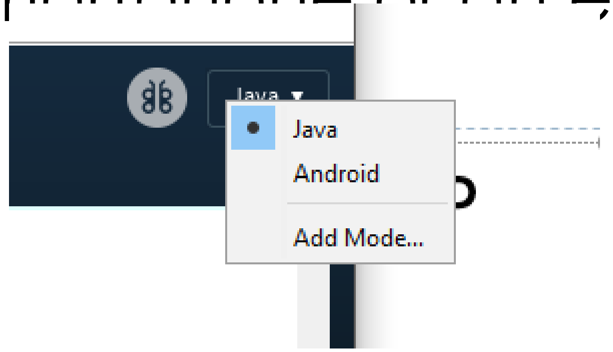
- https://processing.org/tutorials/p3d/

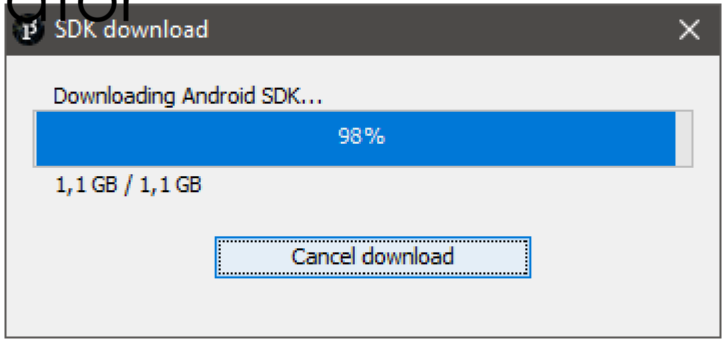
## Processing and 3D

- If we want to draw a box instead of a rect, our first instinct would be to add an additional parameter to the rect function
- This simply does not work as we do not know the orientation of the rect in the space
- Solution: translate(), rotate() and scale()
- This operations are combined into mathematical structures called matrices (plural of matrix)
- Matrix must be created and enabled after every transformation
- At the beginning the program uses the identity matrix for drawing

## Processing and Android

- Processing offers a native support to Android APP development
- http://android.processing.org/tutorials/index.html
- To enable Android mode click on Android into the node selection combo and wait
- Applications can be tested using your own smartphone or an emulator




SDK download

Downloading Android SDK...

98%

1,1 GB / 1,1 GB

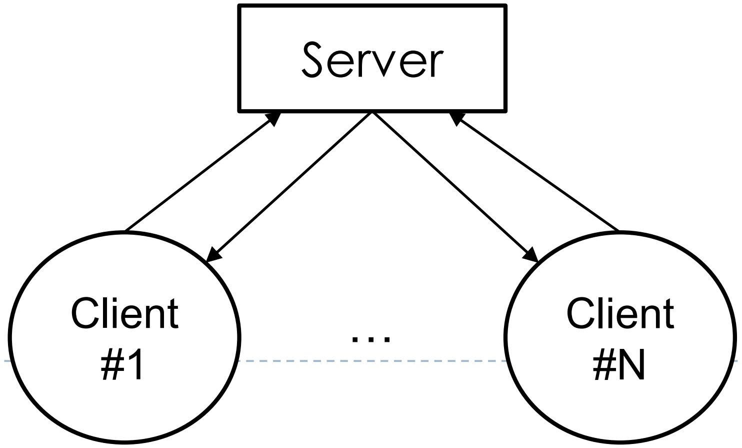
Cancel download

## Processing Android and VR

- Among the possibilities offered by processing and Android we have a simple support for virtual reality
- We can have VR using expensive visors (e.g., Oculus Rift) or using Cardboards
- http://android.processing.org/tutorials/vr_intro/index.html

# Data Stream and Networking

- We will see now how to use the Processing network library to create sketches that talk to each other in real-time.
  - Example of this are chat, games, instant messenger, etc.
- The communication happens in “client–server mode”, where tasks are partitioned between the providers of a resource or service, called **Server**, and service requesters, called **Client**.
  - When you make a request on Google, you act as a **client**, while Google is the ***server*** that provides a response.
- For multiuser applications in Processing where we need near **real-time communication**, we need a synchronous connection with a `Socket`.



## Creating a Server

- In order to start working with sockets, we need first to build a **server**.
- The server job will be to:
  - **Open connections** for clients;
  - **Respond to their requests**.
- To create a server, we must first import the library and create an instance of a Server object.

```
import processing.net.* ;
Server server;
```

- The server is initialized via the constructor, which takes two arguments: the keyword `this` (reference to this sketch; if the server is on a remote location, we must use is IP address) and an integer value for a port number.

```
server = new Server(this, 5204);
```

Port numbers range from 0 to 65,536. However, ports 0 through 1,024 are usually reserved for common services, so it is best to avoid those.

# Creating a Server

- The server starts and waits for connections as soon as it is created. It can be closed at any time by calling the `stop()` function.

```
server.stop();
```

- At this point, we can find out if a new client has connected to our server by implementing the built-in function `serverEvent()`, which requires two arguments:
  - a server object (the one generating the event)
  - a client object (that has connected). We might use this function, for example, to retrieve the IP address of the connected client.

```
// The serverEvent function is called if a new client connects
  void serverEvent(Server server, Client client) {
     println( "A new client has connected: " + client.ip());
  }
```

## Creating a Server

- When a client sends a message (after having connected), a `serverEvent()` **is not generated**.
- Instead, we must use the `available()` function to determine if there is a new message from any client available to be read.
- If there is, a reference to the client broadcasting the method is returned and we can read the content using the `readString()` function.
- If nothing is available, the function will return the value `null`, meaning no value (or no client object exists).

```
void draw() {
// If a client is available, we will find out.
// If there is no client, it will be "null"
Client someClient = server.available();
if (someClient! = null) {
 println( "Client says: " + SomeClient.readString());
}
}
```

## Creating a Server

- The function `readString()` is useful in applications where text information is sent across the network.
- If the data should be treated differently, for instance, as a number or an image, other `read()` methods can be called.

- A server can also send messages out to clients, and this is done with the `write()` function.

```
server.write("Great, thanks for the message!\n");
```

- It is often a good idea to send a *newline character* ‘\n’ at the end of your messages to recognize incoming/outgoing messages.

## Example of a simple server

```
import processing.net.*; // Import the net libraries

Server server; // Declare a server

float newMessageColor = 255; // Used to indicate a new message has arrived

String incomingMessage = "" ;

void setup() {
 size(400,200);
 server = new Server(this, 5204); // Create the Server on port 5204
}
void draw() {
 background(newMessageColor);
 newMessageColor += 0.3; //newMessageColor fades to white over time
 newMessageColor = constrain(newMessageColor,0,255);
 textAlign(CENTER);
 fill(255);
```

## Example of a simple server

```
text(incomingMessage,width/2,height/2);
Client client = server.available(); // If there is no client, it will be "null"
if (client != null) {// We should only proceed if the client is not null
   incomingMessage = client.readString(); // Receive the message
   incomingMessage = incomingMessage.trim();

  if(incomingMessage.equalsIgnoreCase("hello") ||
                                incomingMessage.equalsIgnoreCase("hi")) {
    server.write( "Hello my friend!\n" );
  }
  else if(incomingMessage.equalsIgnoreCase("how are you")) {
    server.write("Fine, thanks!\n");
  }
  else if(incomingMessage.equalsIgnoreCase("i am tired")) {
    server.write("Well, you can drink coffeee!\n");
    }
  else { server.write("I am sorry, but my vocabulary is very limited...\n"); }
```

## Example of a simple server

```
// Reset newMessageColor to black
  newMessageColor = 0;
}
}


// The serverEvent function is called whenever a new client connects.
void serverEvent(Server server, Client client) {
 incomingMessage = " A new client has connected: " + client.ip();
 println(incomingMessage);
 // Reset newMessageColor to black
 newMessageColor = 0;
}
```

## Creating a Client

- Once the server is running, we can create a client that connects to the server.
- We start off the same way we did with a server, importing the net library and declaring an instance of a Client object.

```
import processing.net.*;

Client client;
```

- The client constructor requires three arguments: "this", referring again to this sketch, the IP address we want to connect to (as a String), and the port number (as an integer).
- If the **server is running on a different computer** than the client, **you will need to know the IP address** of that server computer.
- In addition, if there is no server running at the specified IP and port, the Processing sketch will give the error message: " java.net.ConnectException: Connection refused " meaning either the server rejected the client or that there is no server.

## Creating a Client

- Sending to the server is easy using the `write()` function.

```
        client.write("Hello!");
```

- Reading messages from the server is done with the `read()` function. To read the entire message as a String, `readString()` is used.
- In this case, we use the function `readStringUntil()` to guarantee the reading of the entire string until the line breaks.
- Before we can even contemplate reading from the server, we must be sure there is something to read. The `clientEvent()` whenever there is data avalaible to read.

```
if (client.available() > 0) {
        String message = client.readString();
}
```

## Example of a simple client

```
import processing.net.*;
Client client;
float newMessageColor = 0; // Used to indicate a new message
String messageFromServer = " " ; // A String to hold whatever the server says
String typing = " " ; // A String to hold what the user types

void setup() {
 size(400,200);
 client = new Client(this, "127.0.0.1", 5204); // Create the Client

}
void draw() {
 background(255);
 fill(newMessageColor); // Display message from server
 textAlign(CENTER);
 text(messageFromServer,width/2,140);
```

## Example of a simple client

```
// Fade message from server to white
newMessageColor += 1;
newMessageColor = constrain(newMessageColor,0,255);

 // Display Instructions
 fill(0);
 text( "Type text and hit return to send to server. ",width/2,60);

 // Display text typed by user
 fill(0);
 text(typing,width/2,80);
}
```

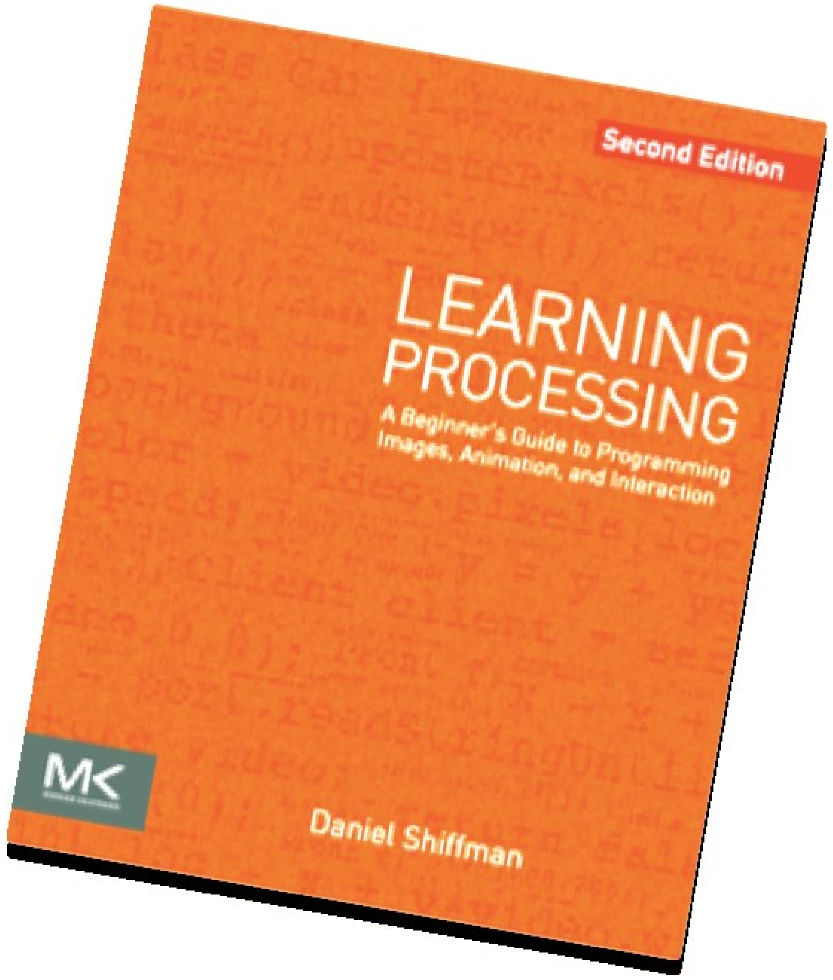
# Example of a simple client

```
 void clientEvent(Client client) {
  String msg = client.readStringUntil('\n');
  println(msg);
   if (msg != null) {
      messageFromServer = msg; // Read it as a String
      newMessageColor = 0; // Set brightness to 0
   }
}
void keyPressed() {
if (key == '\n') {
 // If the return key is pressed, save the String and clear it
 // When the user hits enter, write the sentence out to the Server
 client.write(typing);
 typing = " " ;
 } else {
 typing = typing + key; }
}
```

## Video, Sound and more on Images

- More on 2D images
  - https://processing.org/tutorials/pixels/
- 3D images
  - https://processing.org/tutorials/p3d/
- Video
  - https://processing.org/tutorials/video/
- Sound
  - https://processing.org/tutorials/sound/
- Interesting Examples:
  - https://processing.org/examples/

## Reference Book


Second Edition
LEARNING PROCESSING
A Beginner's Guide to Programming Images, Animation, and Interaction
MK
Daniel Shiffman