

Processing Refresher

Introduction

The purpose of this document is to help you get started with processing on your projects. Many of you will already be familiar with Processing from previous courses. For the uninitiated, Processing is just a framework that allows users to easily add graphical elements to their projects.

This refresher begins by going through the basic concepts in Processing's Java mode (the default mode). It then goes into some useful tips for tracking your Processing project in git. The last section will show you how to download and set up Python mode for processing.

Download and Install

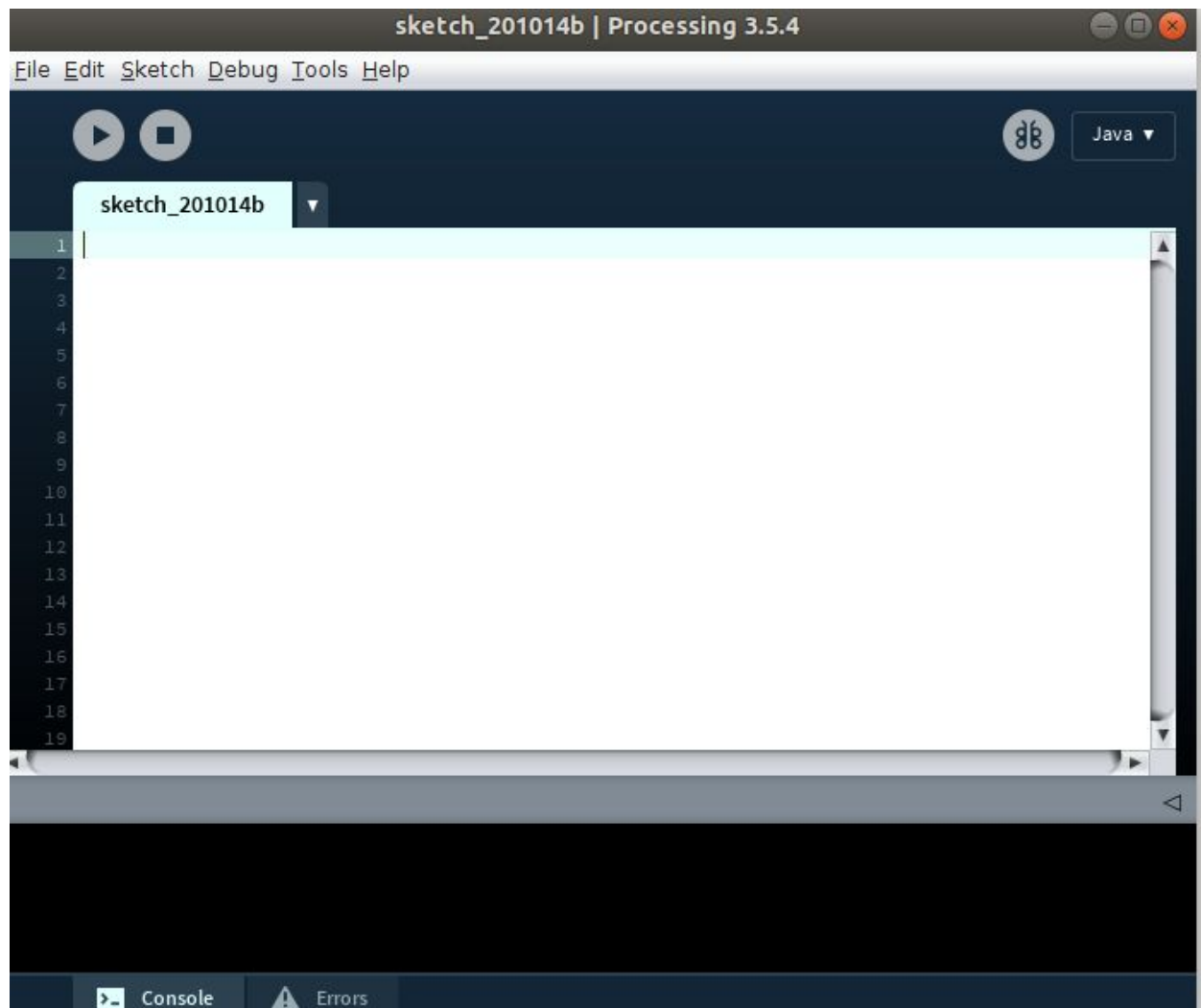
Download the latest version of processing from <https://processing.org/download/>. Most users will be using the 64-bit version.

Processing in Java

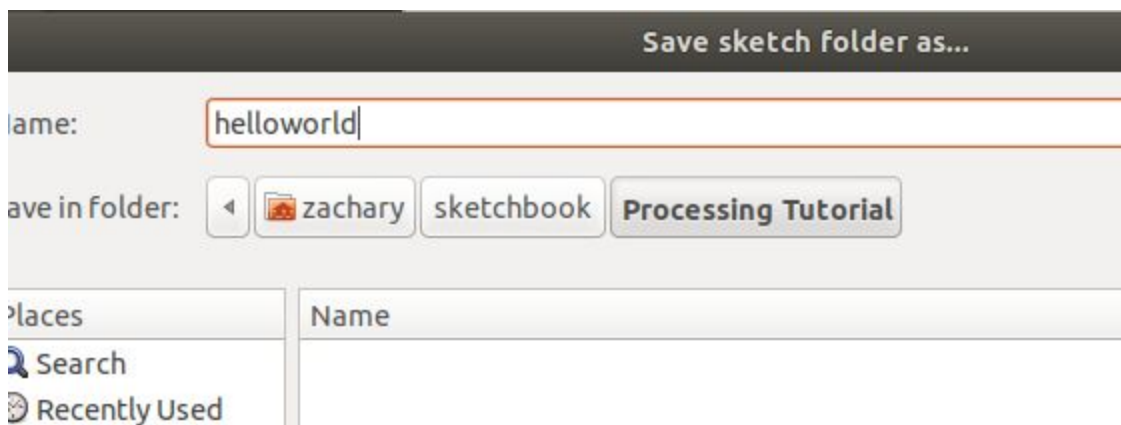
Processing provides two modes for users to use: Python and Java. I'll start with explaining how to use it in Java since this is the default, and most of the concepts will transfer over to the Python version.

Creating a sketch

A "sketch" is basically just the processing way of saying a program. When we start Processing it will usually start with a blank sketch such as the one below.



The first thing you'll want to do is save that under a more meaningful name. I recommend creating a sub-folder just for your project so that it becomes easier to integrate with git and github. Here I've saved my project as "helloworld" in a "Processing Tutorial" folder that I made.



Setup and draw

Setup

Now let's actually do something with our project! First of all, there are two very important functions to define in Processing. These are our setup and our draw functions. Setup will be run once when the program is first run, and draw will continually run as long as the program is still running. Below I've provide stubs for these functions.

```
helloworld ▼  
1 void setup(){  
2 }  
3  
4 void draw(){  
5 }  
6
```

Now we can actually run this program by hitting the triangle in the top left. It will produce a nice little square like this:



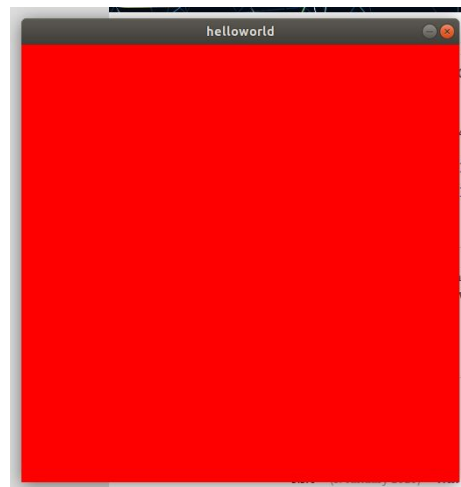
But that's kind of boring. First of all, you might notice that it's kind of small. There is a special processing function called "size" that allows us to change the size of our canvas. For example, below I have set the size to 500 by 500 pixels.

```
void setup(){  
  size(500,500);  
}
```

What about changing the colour? That can be done with the "background" function as shown below.

```
void setup(){  
  size(500,500);  
  background(255,0,0);  
}
```

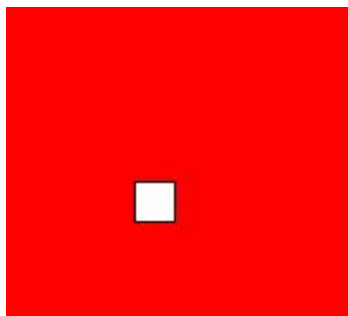
This just takes in three integers, the “R”, “G”, and “B” values of the desired background color. The code I just wrote makes a red background as shown below.



What about drawing some cars? Well maybe we'll take a shortcut and draw rectangles instead. Fortunately there is a processing function for that too! It just needs the x and y coordinates and width and height of our rectangle. Note that the x and y are measured from the top left corner.

```
void setup(){  
  size(500,500);  
  background(255,0,0);  
  rect(50,50,20,20);  
}
```

And we can see this produces a nice little white square.



You can see exactly how this function works by going to the processing reference at <https://processing.org/reference/>. IMPORTANT NOTE: Learning how to read a reference page

like this is a very important skill as a software developer so it's good to develop the skills now. For now, see if you can figure out how to make a triangle appear on the screen.

That's all good, but what about making things move? Well that's where the "draw" function comes in.

Draw

To make our rectangle move around, we will just redraw it every frame. This means moving it to the draw function like below.

```
void draw(){  
  rect(50,50,20,20);  
}
```

This alone won't be enough however, since we're just drawing it in the same spot. Let's introduce a variable x that we increment each time. This will just substitute for the rectangles x-coordinate as shown. Incrementing each time the loop is run means it will move across the screen.

```
helloworld  
1 int x;  
2  
3 void setup(){  
4   size(500,500);  
5   background(255,0,0);  
6   x = 0;  
7 }  
8  
9 void draw(){  
10  rect(x,50,20,20);  
11  x++;  
12 }  
13
```

Unfortunately this is not exactly what we want.



The reason is that we forgot to draw the background again so we just have a whole bunch of squares! Running “background” in draw will fix this.

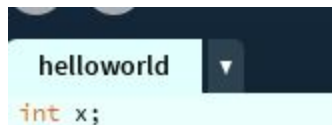
```
int x;

void setup(){
  size(500,500);
  x = 0;
}

void draw(){
  background(255,0,0);
  rect(x,50,20,20);
  x++;
}
```

Using separate tabs

A little aside, you might wonder how you can keep your code organized in processing. Tabs are really your friend here. To create a new tab you just need to hit the little triangle next to your current tab (shown below), and then “new tab”. Any classes or functions defined in our new tab will be accessible throughout our program.



Events

The next thing you might worry about is how to interact with your program. Fortunately processing provides this too! If we go on over to the processing reference we will see this section:

Input

Mouse

mouseButton
mouseClicked()
mouseDragged()
mouseMoved()
mousePressed()
mousePressed
mouseReleased()
mouseWheel()
mouseX
mouseY
pmouseX
pmouseY

Keyboard

key
keyCode
keyPressed()
keyPressed
keyReleased()
keyTyped()

which shows us all the ways we can pass input to our program. There are two important ways to consider: variables and callbacks.

First of all, variables. These just hold a value that we can access directly in our program. For example, if I want to stop the white box from moving whenever a key is pressed, I can do the following.

```
void draw(){  
  background(255,0,0);  
  rect(x,50,20,20);  
  if(!keyPressed)  
    x += 1;  
}
```

The other way to interact with the user is through “callback” functions. These will be called whenever a specified event occurs. For example, the mouseClicked() function will be called whenever the mouse is clicked. If I want to send my square back to the start whenever I click the mouse I could add the following code.

```
15 void mousePressed(){  
16     x = 0; |  
17 }
```

To see all the ways you can interact with your program, see the reference!

Processing with git

Finding your source files

Next I'd like to touch on how we can use source control with our processing programs. First of all, how do I access the files? The best thing to do is just remember where you saved the files. If you don't remember you can open up the "Save As" menu and then note where the default location is. Then hit "cancel".

Create a git repository

Once you know where your files have been saved, open up a git bash/terminal and change to that directory. If you don't know how to do this, you might want to check the git tutorial. For example I did:

```
zachary@zach-lenovo-linux:~$ cd ~/sketchbook/Processing\ Tutorial/  
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$ ls  
helloworld  
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$
```

We can create a git repository by typing git init.

```
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$ git init  
Initialized empty Git repository in /home/zachary/sketchbook/Processing Tutorial/.git/  
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$
```

Now you might want to add a .gitignore file. This will prevent git from tracking files that are not source code such as java .class files or the .tmp files that processing uses to save the intermediate state of a program (i.e. it "ignores" them). You can make this in any text editor, and it will look something like this:


```
*.class
*.tmp
~
~
~
~
~
```

Each line simply contains a pattern of files to ignore. The “*” will match anything, so the first line says “ignore any file ending in .class”.

I recommend you add the .gitignore to your project, since it will be useful for other people who download your repo. Then you can go ahead and make your initial commit!

```
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$ git add .gitignore
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$ git add helloworld/
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$ git commit -m "Initial commit"
[master (root-commit) 5e8226f] Initial commit
2 files changed, 19 insertions(+)
create mode 100644 .gitignore
create mode 100644 helloworld/helloworld.pde
zachary@zach-lenovo-linux:~/sketchbook/Processing Tutorial$
```

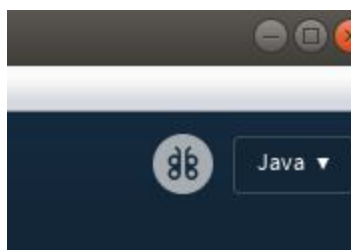
You should also add it to github as shown in the git tutorials so that your teammates can begin collaborating with you.

Processing in Python

We’ve been using Java for most of this tutorial, but before you go, I want to touch on how to use Python in Processing.

Setup

If you haven’t already, the first thing you need to do is download Processing. You should then be able to go to the top right corner where it says “Java” as shown below. Click that, and then hit “Add mode” followed by “Python”.



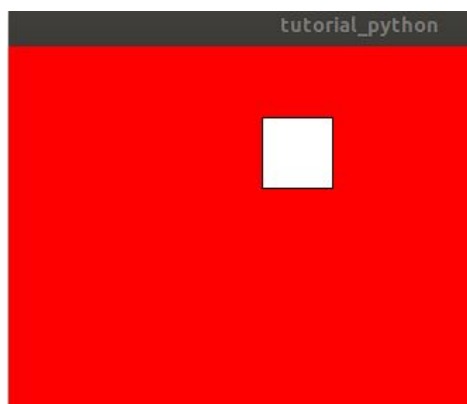
Once you have added the Python mode you will need to switch to it. Click the top right corner again and then change your mode to Python. If you have any open sketches, you may need to save the sketch, close it, and open a new blank sketch. Don't save the new blank sketch before changing mode otherwise it will write it as a Java file!

Coding in Python

Using Python in Processing is almost exactly the same as in Java. All the functions have the same names, just with Python syntax. Note that if you modify any global variables within a function, you will be forced to declare them as global. Below you can see how I have written the same program we wrote in the first part of the refresher, but in Python.

```
tutorial_python
1 x = 0
2
3 def setup():
4     size(500,500)
5
6
7 def draw():
8     global x
9     background(255,0,0)
10    rect(x,50,50,50)
11    if not keyPressed:
12        x += 1
13
14 def mouseClicked():
15     global x
16     x = 0
```

If you did everything right, you should get a little something like this

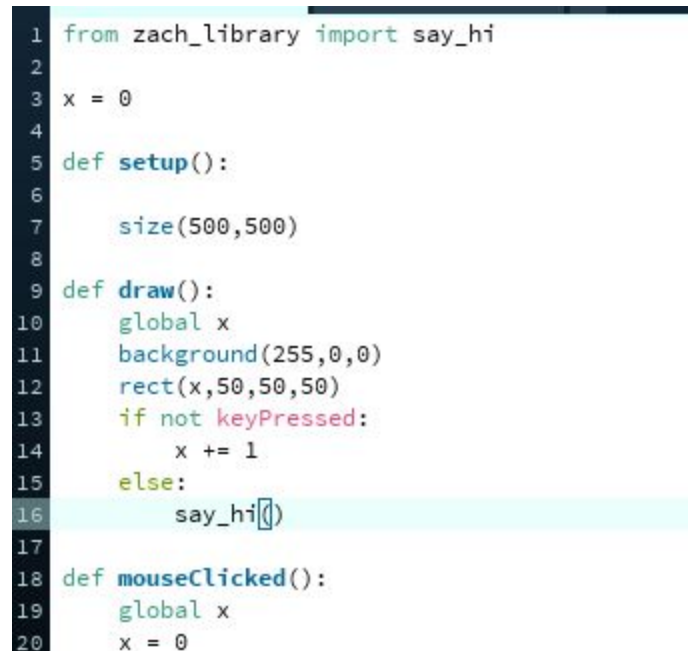


The last thing to note is that if you make any new tabs, you will need to “import” them. For example, if I create a new function as shown below:



```
tutorial_python zach_library.py ▼  
1 def say_hi():  
2     text("Hi", width/2, height/2)
```

I can import and use it like so:



```
1 from zach_library import say_hi  
2  
3 x = 0  
4  
5 def setup():  
6     size(500,500)  
7  
8  
9 def draw():  
10     global x  
11     background(255,0,0)  
12     rect(x,50,50,50)  
13     if not keyPressed:  
14         x += 1  
15     else:  
16         say_hi()  
17  
18 def mouseClicked():  
19     global x  
20     x = 0
```

And this will just print “hi” in the middle of the screen whenever I press a key.

Summary

In this tutorial we’ve seen how to use Processing both Python and Java. Recall that the most important functions are the `setup()` and `draw()` functions, and that the language reference at <https://processing.org/reference/> is your best friend. And if you’re using git, don’t forget about the `.gitignore` file! This will make your life a lot easier.

Good luck!