Jocelyn Documentation

Release 0.1.0

Danny O'Connor

November 20, 2012

CONTENTS

Jocelyn is shim that makes it easier to use the Processing core libraries from Jython. For how Jocelyn differs from other Python Processing implementations see Rationale.

The library ships with a version of Processing that is automatically added to the classpath (if not already there) when the module is imported. The current version of Processing bundled is 1.5.0.

CHAPTER

ONE

INSTALLATION

The recommended way to install Jocelyn is to create a Jython virtual-env and install into that like so:

```
$ pip install jocelyn
```

It's also possible to install the package from the source code on github.

CHAPTER

TWO

TUTORIAL

2.1 Introduction

As you'll see Jocelyn uses some reflection trickery to make it pretty easy to translate code almost directly from traditional Processing. Therefore, it's pretty useful to have the Processing Language Reference available when playing with sketches.

2.2 Getting Started

To get started, create a file called (say) circles.py, then import the library and create a sublclass of `Sketch` like so:

```
from jocelyn import *

class CircleSketch(Sketch):

    def setup(self):
        pass

    def draw(self):
        pass

if __name__ == '__main__':
    CircleSketch().run_sketch()
```

Open a terminal and activate the virtualenv in which you have jocelyn installed, for example:

```
$ source ~/.virtualenvs/jocelyn/bin/activate
```

Run the sketch like so:

```
$ jython circle.py
```

At this point, if everything is installed correctly, you should see a blank box with window controls that looks something like

2.3 Drawing Simple 2D Shapes

Jocelyn works by assigning the current sketch to a thread local variable and creating delegates for the declared methods on PApplet using Java reflection.

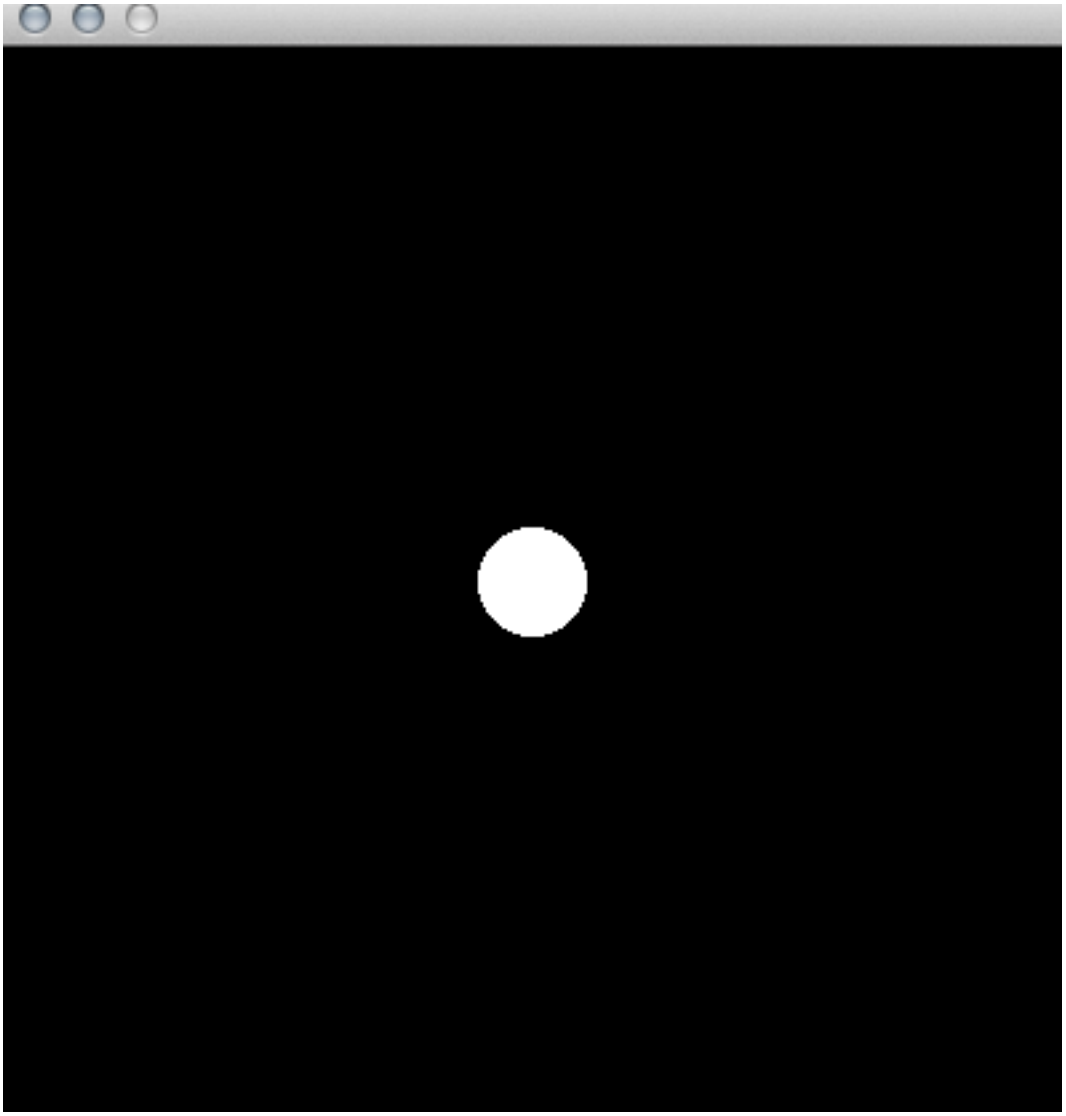
In essence what this means is that you can translate Processing code fairly easily into jocelyn code. All of the methods in the Processing language (like `line`, `ellipse` etc.) can be imported from jocelyn and invoked anywhere in your module.

To access variables of the sketch (like `mouseX`, `width` and `height`) jocelyn provides the `Q` method.

Let's update the Sketch to create something simple like a white circle on a black background:

```
from jocelyn import *

# These are obtainable from the Sketch using the 'Q' method but in
# general it's better to define them as module variables ( which also
# makes it slightly easier to translate Processing code.

width = 400
height = 400

class CircleSketch(Sketch):

    def setup(self):
        size(width, height)
        background(0)

    def draw(self):
        stroke(255)
        ellipse(width/2,height/2,40,40)

if __name__ == '__main__':
    CircleSketch().run_sketch()
```

You should something that looks like:



2.4 Interactivity

The Sketch class is a subclass of `PApplet` so to implement Processing methods like `mousePressed` we add them as methods on the Sketch class.

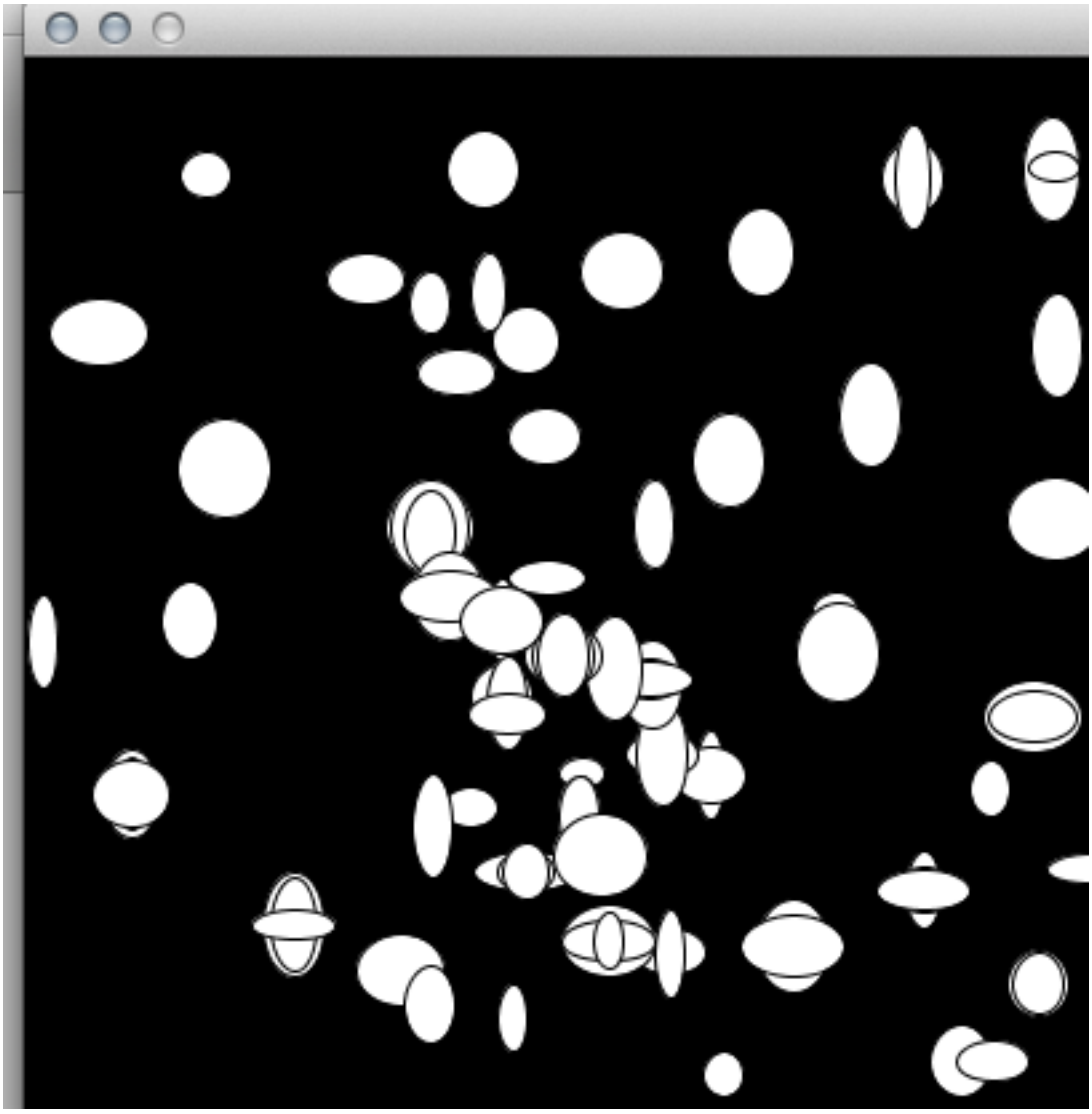
So, make our sketch slightly more interesting, lets make it so we draw a random circle on each mouse-press:

```
from jocelyn import *

width = 400
height = 400

class Circles(Sketch):

    def setup(self):
        size(width, height)
        background(0)
        smooth()
```

```
    def mousePressed(self,e):
        circle_height = random(10,40)
        circle_width = random(10,40)
        ellipse(Q('mouseX'),Q('mouseY'),circle_height, circle_width)

    def draw(self):
        pass

if __name__ == '__main__':
    Circles().run_sketch()
```

Notice above that we're using the `Q` method to access the `mouseX` and `mouseY` properties of the sketch when the `mousePressed` method is invoked.

Run the sketch as before and click around to make some art!



2.5 Creating Static Images

To make it marginally simpler to create static images, jocelyn includes a specialization of the `Sketch` class, `SketchedImage`. Rather than creating a frame with window borders (like the standard `Sketch`) `SketchedImage` writes directly to a file specified in it's constructor.

For example, the 'Single Circle' sketch above could also be written as a `SketchedImage`:

```
from jocelyn import *

width = 400
height = 400

class CircleSketch(SketchedImage):
    """
    Sample of SketchedImage, the 'setup' and 'draw' methods are
    replaced by one 'draw_image' method.

    Also the explicit call to 'size' is no longer required.

    """
    def draw_image(self):
        background(0)
        stroke(255)
        ellipse(width/2,height/2,40,40)

if __name__ == '__main__':
    CircleSketch(width, height, "single_circle.png").run_sketch()
```

CHAPTER

THREE

USING PROCESSING LIBRARIES

Processing ships with lots of useful libraries. To make it easier to use these from Jocelyn, any jar files in the `libraries` directory or any of its sub-directories (relative to the sketch module) will be added to the classpath along with the Processing libraries.

See the libarary example in the examples directory of the source for details.

CHAPTER

FOUR

RATIONALE

This library is designed for people with a Python background who want to play with Processing and who don't mind running Jython.

Soomebody more familiar with Processing than Python would probably be happier with processing.py which is more faithful reconstruction of the Processing language.

This library differs from processing.py in that it's invoked using a standard Jython interpreter, it comes as a setup-tools enabled package and it doesn't modify built-ins.

If you would prefer to run CPython (and don't mind not running the Processing libraries themselves) there is also a package called pyprocessing that implements the Processing language using OpenGL and Piglet.

CHAPTER

FIVE

API REFERENCE

The important modules of the jocelyn module are:

CHAPTER

SIX

CHANGELOG

6.1 Version 0.1.0

First release.

PYTHON MODULE INDEX