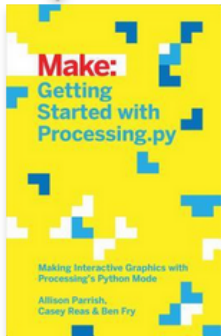


- ▶ I'll put take-home practice exam (due 12/6 or 12/9) online this evening. I put 4 years' old exams online at <http://positron.hep.upenn.edu/p8/files/oldexams/>
- ▶ Today: a tutorial of the “Processing.py” computer programming language — whose purpose is to learn how to code within the context of the visual arts. It makes coding fun and visual. Processing.py is a Python-based version of the (Java-based) Processing programming environment.
- ▶ **Extra-credit options (if you're interested):**
 - ▶ Learn to use Mathematica (ask me how), which is a system for doing mathematics by computer. (It is the brains behind Wolfram Alpha.) Penn's site license makes Mathematica free-of-charge for SAS and Wharton students.
 - ▶ Use “Processing.py” (or ordinary “Processing”) to write a program to draw or animate something that interests you. (Not necessarily physics-related.)
 - ▶ Knowing “how to code” is empowering & enlightening. So I offer you an excuse to give it a try, for extra credit, if you wish.
- ▶ Today's examples online at



Want to Read


Rate this book



Getting Started with Processing.Py: Making Interactive Graphics with Processing's Python Mode

by Allison Parrish, Ben Fry, Casey Reas



4.88 ·  Rating details · 8 Ratings · 3 Reviews

Processing opened up the world of programming to artists, designers, educators, and beginners. The Processing.py Python implementation of Processing reinterprets it for today's web. This short book gently introduces the core concepts of computer programming and working with Processing. Written by the co-founders of the Processing project, Reas and Fry, along with co-author Allison Parrish, *Getting Started with Processing.py* is your fast track to using Python's Processing mode. ([less](#))

The software is free & open-source. Runs on Mac, Windows, Linux. The “getting started” book will set you back about \$15.

or start with the in-browser video tutorial (no download needed): <http://hello.processing.org> (Processing, **not** Processing.py)

Processing



Cover

[Download](#)
[Donate](#)

[Exhibition](#)

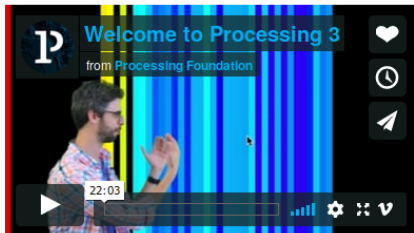
[Reference](#)
[Libraries](#)
[Tools](#)
[Environment](#)

[Tutorials](#)
[Examples](#)
[Books](#)
[Handbook](#)

[Overview](#)
[People](#)

[Shop](#)

» [Forum](#)
» [GitHub](#)
» [Issues](#)
» [Wiki](#)



Welcome to Processing 3! Dan explains the new features and changes; the links Dan mentions are on the [Vimeo page](#).

- » [Download Processing](#)
- » [Browse Tutorials](#)
- » [Visit the Reference](#)

Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. There are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning and prototyping.

» [Donate](#)

Please join us as a member of the Processing Foundation. [We need your help!](#)

» [Exhibition](#)



[komorebi](#)
by Leslie Nooteboom



[Particle Flow](#)
by NEOANALOG

[Cover](#)[Reference](#)[Tutorials](#)[Examples](#)[Bugs](#)

Python Mode for Processing

You write Processing code. In Python.

Processing is a programming language, development environment, and online community. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

Processing was initially released with a Java-based syntax, and with a lexicon of graphical primitives that took inspiration from OpenGL, Postscript, Design by Numbers, and other sources. With the gradual addition of alternative programming interfaces — including [JavaScript](#), [Python](#), and [Ruby](#) — it has become increasingly clear that Processing is not a single language, but rather, an arts-oriented approach to learning, teaching, and making things with code.



Cover

Download

Donate

Exhibition

Reference

Libraries

Tools

Environment

Tutorials

Examples

Books

Handbook

Overview

People

Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below.



3.4 (26 July 2018)

Windows 64-bit
Windows 32-bit

Linux 64-bit
Linux 32-bit
Linux ARM
(running on Pi?)

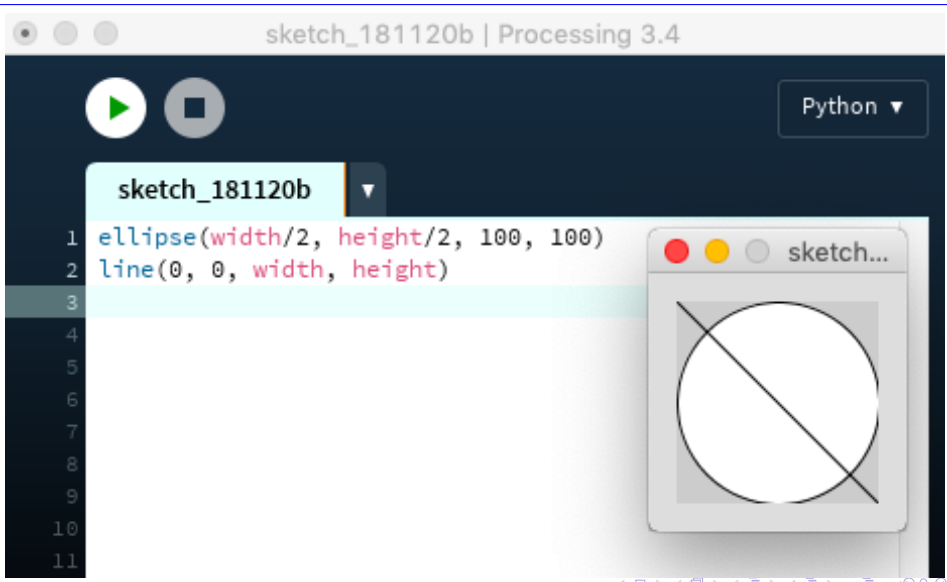
Mac OS X

- » Github
» Report Bugs
» Wiki
» Supported Platforms

Read about the changes in 3.0. The list of revisions covers the differences between releases in detail.

“hello world” program

Let's draw a circle and a line.



More commonly, a Processing program has a function called `setup()` that runs once when the program starts, and another function called `draw()` that runs once per frame.

```
def setup():  
    # this function runs once when the program starts up  
    size(900, 450) # sets width & height of window (in pixels)  
  
def draw():  
    # this function runs once per frame of the animation  
    line(0, frameCount, width, height-frameCount)
```

Let's make it do something repetitive

```
def setup():
    # this function runs once when the program starts up
    size(900, 450) # sets width & height of window (in pixels)

def draw():
    # this function runs once per frame of the animation
    dy = 0.5*height + 0.5*height*sin(0.01*frameCount)
    line(0, dy, width, height-dy)
```


How about repeating something more exciting?

```
def setup():
    # this function runs once when the program starts up
    size(900, 450) # sets width & height of window (in pixels)

def draw():
    # this function runs once per frame of the animation
    dy = 0.5*height + 0.5*height*sin(0.01*frameCount)
    line(0, dy, width, height-dy)
    t = 0.02*frameCount
    x = 0.5*width + 200*cos(t)
    y = 0.5*height + 200*sin(t)
    ellipse(x, y, 20, 20)
```

Did you ever have a Spirograph toy when you were a kid?

```
def setup():  
    size(900, 450)  
  
def draw():  
    t = 0.02*frameCount  
    x = 0.5*width + 200*cos(t) + 30*cos(11*t)  
    y = 0.5*height + 200*sin(t) - 30*sin(11*t)  
    ellipse(x, y, 5, 5)
```



Spirograph

How about something that starts to resemble physics? A really, really low-tech animation of an planet orbiting a star.

```
def setup():  
    size(900, 450)  
  
def draw():  
    t = 0.01*frameCount  
    xsun = 0.5*width  
    ysun = 0.5*height  
    ellipse(xsun, ysun, 20, 20)  
    rplanet = 200  
    xplanet = xsun + rplanet*cos(t)  
    yplanet = ysun + rplanet*sin(t)  
    ellipse(xplanet, yplanet, 10, 10)
```

Let's add a moon in orbit around the planet.

```
def draw():
    t = 0.01*frameCount
    xsun = 0.5*width
    ysun = 0.5*height
    # clear screen before each new frame
    background(128)
    # draw sun
    ellipse(xsun, ysun, 20, 20)
    rplanet = 200
    xplanet = xsun + rplanet*cos(t)
    yplanet = ysun + rplanet*sin(t)
    # draw planet
    ellipse(xplanet, yplanet, 10, 10)
    rmoon = 30
    xmoon = xplanet + rmoon*cos(t*365/27.3)
    ymoon = yplanet + rmoon*sin(t*365/27.3)
    # draw moon
    ellipse(xmoon, ymoon, 5, 5)
```

How about adding an inner planet?

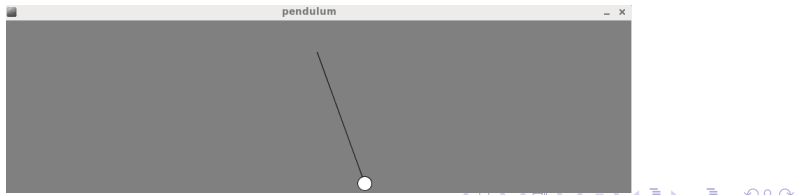
```
def draw():
    ... other stuff suppressed ...
    # draw moon
    ellipse(xmoon, ymoon, 5, 5)
    # add second planet
    year_mercury_days = 115.88 # from Wikipedia
    T_ratio = year_mercury_days/365.25
    R_ratio = T_ratio**(2.0/3)
    xplanet = xsun + R_ratio*rplanet*cos(t/T_ratio)
    yplanet = ysun + R_ratio*rplanet*sin(t/T_ratio)
    ellipse(xplanet, yplanet, 7, 7)
```

Animate a pendulum (skip?)

```
def setup():
    size(900, 450)

def draw():
    t = 0.01*frameCount
    g = 9.8
    L = 2.0
    degree = PI/180
    amplitude = 20*degree
    omega = sqrt(g/L)
    theta = amplitude * sin(omega*t)
    xbob = L * sin(theta)
    ybob = L * cos(theta)
    # convert coordinates into pixel coordinates
    ... continued on next slide ...
```

```
def draw():
    ... continued from previous slide ...
    # convert coordinates into pixel coordinates
    xpixel_pivot = 0.5*width
    ypixel_pivot = 0.1*height
    scale = 100.0 # pixels per meter
    xpixel_bob = xpixel_pivot + scale*xbob
    ypixel_bob = ypixel_pivot + scale*ybob
    # clear the screen for each new frame of animation
    background(128)
    # draw the string
    line(xpixel_pivot, ypixel_pivot, xpixel_bob, ypixel_bob)
    # draw the bob
    ellipse(xpixel_bob, ypixel_bob, 20, 20)
```

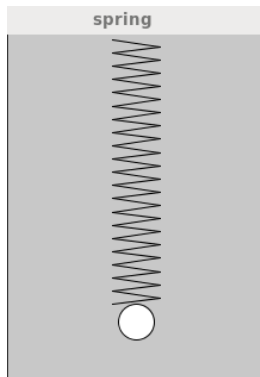


Animate a mass bobbing on a spring

```
def draw():
    t = 0.01*frameCount
    omega = 1.0
    amplitude = 0.5
    Lequilibrium = 2.0
    xbob = 0
    ybob = Lequilibrium + amplitude * cos(omega*t)
    xpixel_anchor = 0.5*width
    ypixel_anchor = 0.01*height
    scale = 100.0
    xpixel_bob = xpixel_anchor + scale*xbob
    ypixel_bob = ypixel_anchor + scale*ybob
    // draw the bob
    rbob = 15
    ellipse(xpixel_bob, ypixel_bob, 2*rbob, 2*rbob)
```

Clear screen between frames; draw the spring

```
def draw():
    ... other stuff suppressed ...
    # clear the screen for each new frame
    background(200)
    # draw the bob
    rbob = 15
    ellipse(xpixel_bob, ypixel_bob, 2*rbob, 2*rbob)
    # draw the spring as a series of zig-zag lines
    nzigzag = 20
    for i in range(nzigzag):
        spring_top = ypixel_anchor
        spring_bottom = ypixel_bob - rbob
        dy = (spring_bottom - spring_top) / nzigzag
        xzig = xpixel_anchor - 20
        yzig = ypixel_anchor + i * dy
        xzag = xpixel_anchor + 20
        ymid = yzig + 0.5 * dy
        yzag = yzig + dy
        line(xzig, yzig, xzag, ymid)
        line(xzag, ymid, xzig, yzag)
```

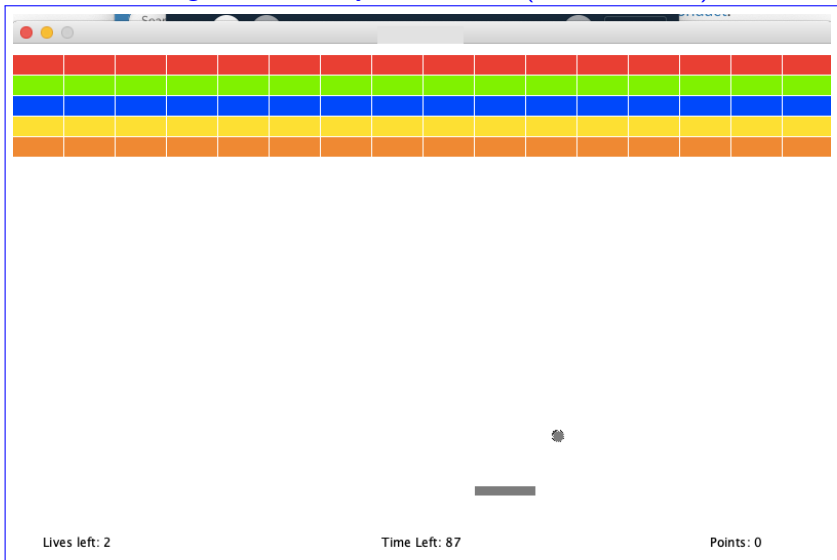


Let's add some "physics" to the spring.

```
# we will update position & velocity frame-by-frame,  
# so we store them in these "global" variables  
y = 1.49 # need to change this to make anything happen!  
vy = 0.0  
  
def draw():  
    dt = 0.01  
    k = 20.0  
    m = 1.0  
    g = 9.8  
    Lrelaxed = 1.0  
    y = y + vy*dt  
    Fy = m*g - k*(y-Lrelaxed)  
    vy = vy + (Fy/m)*dt  
    xbob = 0  
    ybob = Lrelaxed + y  
    ... the rest is unchanged ...
```

https://en.wikipedia.org/wiki/Leapfrog_integration

A “breakout” game coded by a Fall 2017 (and Fall 2018) student.



This was done in Java Processing. Let's try to imitate it in Python!

```
def setup():
    size(900, 450)
    global b
    # Make rectangle location be its center position
    rectMode(CENTER)
    # Instantiate the state of the game board
    b = Breakout()

def draw():
    global b
    b.update()
    b.draw()

class Breakout:

    # "Constructor" for new Breakout object
    def __init__(self):
        ... continued on next slide ...
```

```
class Breakout:

    def __init__(self):
        # various screen boundaries
        self.ytop = 0.0
        self.ybot = height
        self.xleft = 0.0
        self.xright = width
        # ball's size, position, velocity
        self.rball = 7.0
        self.xball = 0.5*width
        self.yball = 0.5*height
        self.speed = 3.0
        self.vxball = self.speed/sqrt(2)
        self.vyball = self.speed/sqrt(2)

    def update(self):
        ... see next slide ...

    def draw(self):
        ... see next slide ...
```

```
class Breakout:

    def __init__(self):
        ... see previous slide ...

    def update(self):
        dt = 1.0
        # use ball velocity to update ball position
        self.xball += self.vxball*dt
        self.yball += self.vyball*dt
        # update ball velocity if it hits the game boundary
        if ((self.xball >= self.xright) or
            (self.xball <= self.xleft)):
            self.vxball *= -1.0
        if ((self.yball >= self.ybot) or
            (self.yball <= self.ytop)):
            self.vyball *= -1.0

    def draw(self):
        ... see next slide ...
```

```
class Breakout:

    def __init__(self):
        ... see earlier slide ...

    def update(self):
        ... see previous slide ...

    def draw(self):
        # clear the screen
        background(200)
        # draw the ball (black)
        fill(color(0, 0, 0))
        ellipse(self.xball, self.yball,
                2*self.rball, 2*self.rball)
```



```
... insert this into Breakout :: __init__
# paddle's location and x,y thickness
self.xpaddle = 0.5*width
self.ypaddle = 0.95*height
self.dxpaddle = 0.1*width
self.dypaddle = 0.02*height

... insert this into Breakout :: update
# make the paddle follow the horizontal mouse position
self.xpaddle = mouseX
# check for ball bouncing off of the paddle
if (abs(self.yball - self.ypaddle) < self.dypaddle/2 and
    abs(self.xball - self.xpaddle) < self.dxpaddle/2 and
    self.vyball > 0):
    self.vyball *= -1.0

... insert this into Breakout :: draw
# draw the paddle (white)
fill(color(255, 255, 255))
rect(self.xpaddle, self.ypaddle,
     self.dxpaddle, self.dypaddle)
```

```
class Brick:
```

```
    def __init__(self, x, y, dx, dy):  
        self.x = x  
        self.y = y  
        self.dx = dx  
        self.dy = dy  
        self.rcolor = random(0, 255)  
        self.gcolor = random(0, 255)  
        self.bcolor = random(0, 255)
```

```
    def checkCollision(self, x, y):  
        if abs(x-self.x) > 0.5*self.dx:  
            return False  
        if abs(y-self.y) > 0.5*self.dy:  
            return False  
        return True
```

```
    def draw(self):  
        fill(color(self.rcolor, self.gcolor, self.bcolor))  
        rect(self.x, self.y, self.dx, self.dy)
```

```
... insert into Breakout :: __init__
# make list of bricks
self.bricks = []
ncol = 10
for irow in range(5):
    for jcol in range(ncol):
        dxbrick = 1.0*width/ncol
        dybrick = 0.05*height
        xbrick = (jcol+0.5)*dxbrick
        if (irow % 2) != 0:
            xbrick += 0.5*dxbrick
        ybrick = 0.1*height + (irow+0.5)*dybrick
        self.bricks.append(Brick(x=xbrick, y=ybrick,
                                dx=dxbrick, dy=dybrick))

... insert into Breakout :: draw
# draw the bricks
for b in self.bricks:
    b.draw()
```

```
... insert into Breakout :: update
# check for collisions with bricks
for i in range(len(self.bricks)):
    b = self.bricks[i]
    if b.checkCollision(self.xball, self.yball):
        # collision! reverse the ball's velocity
        self.vxball *= -1.0
        self.vyball *= -1.0
        # delete the struck brick from the list!
        self.bricks.pop(i)
        # don't check any more bricks this frame,
        # as we modified the list of bricks
        break
```

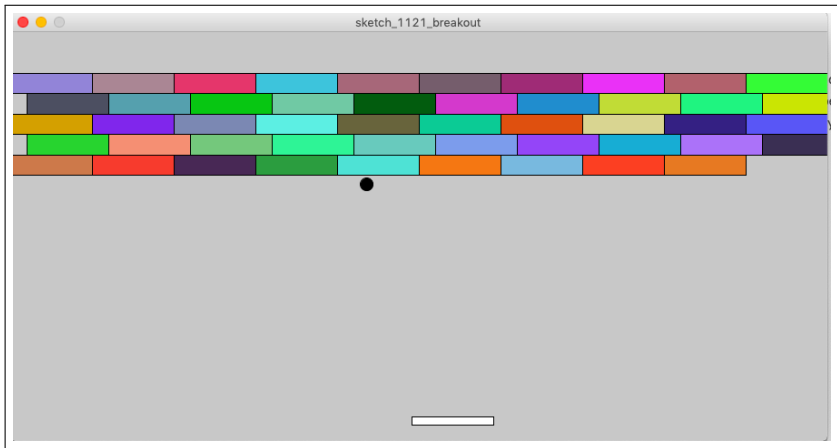
```
... in Breakout :: __init__  
self.previousMouseX = mouseX
```

...

```
... in Breakout :: update  
# estimate the horizontal velocity of the paddle  
vxpaddle = (mouseX - self.previousMouseX)/dt  
self.previousMouseX = mouseX
```

...

```
... upon detecting collision with paddle  
# allow paddle velocity to affect horizontal  
# ball velocity, since otherwise we can get  
# stuck with bricks that cannot be reached  
self.vxball += vxpaddle  
# don't let ball velocity become too horizontal  
minvh = 0.5*self.speed  
if abs(self.vyball) < minvh:  
    self.vyball = -minvh  
# but keep the overall ball speed constant  
temp_speed = sqrt(self.vxball**2 + self.vyball**2)  
self.vxball *= self.speed/temp_speed  
self.vyball *= self.speed/temp_speed
```

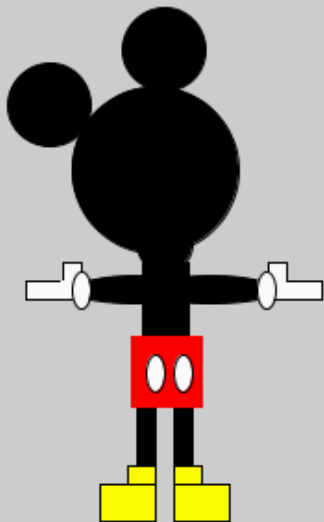


- ▶ The easiest way to get started with the original Java-based version of Processing is to start with this easy online video tutorial that will get you coding in Processing in about an hour! No download or software install is needed for this tutorial — you type your first programs directly into your web browser as you follow along with the video.

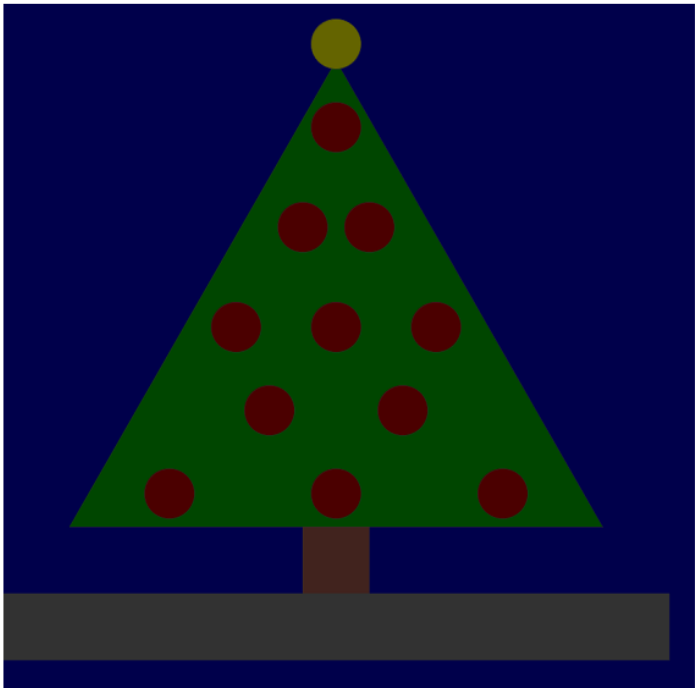
<http://hello.processing.org>

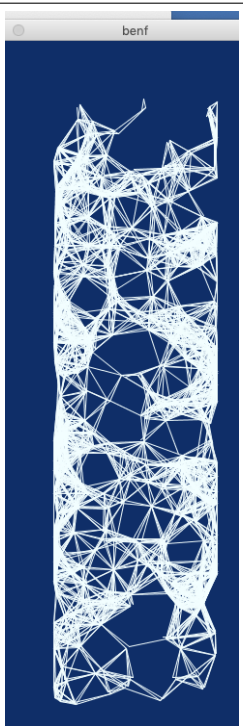
- ▶ For the Python version, work through the first few tutorials at <http://py.processing.org/tutorials>
- ▶ If you're in Addams Hall often, you might ask Orkan Telhan if he has ideas — I believe he still teaches Processing in FNAR 264 / VLST 264, “Art, Design, and Digital Culture.”
- ▶ There are also tons of examples at <http://processing.org> that you could use as starting points or for inspiration, though again these examples use the Java version of Processing.
- ▶ In Fall 2017, ten students sent me Processing sketches! I include a few screen captures on the next few slides.

felix

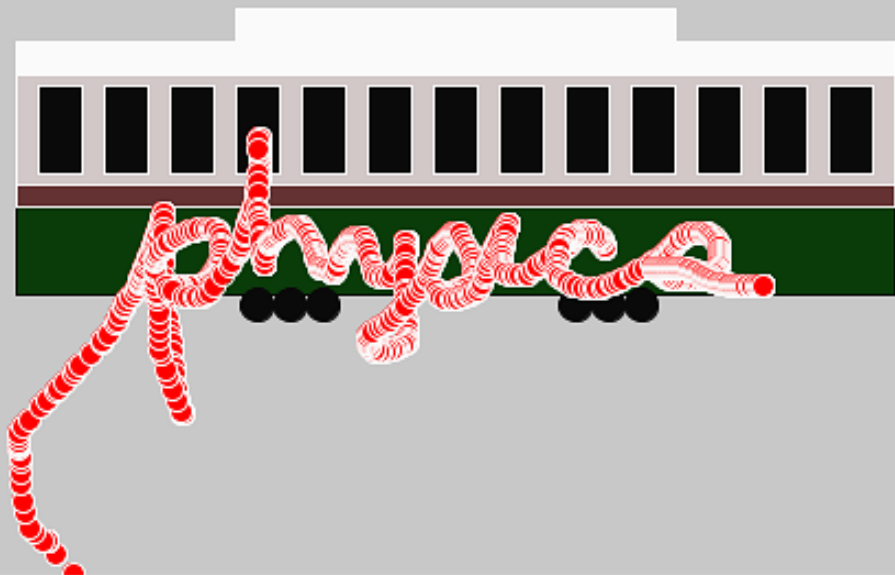




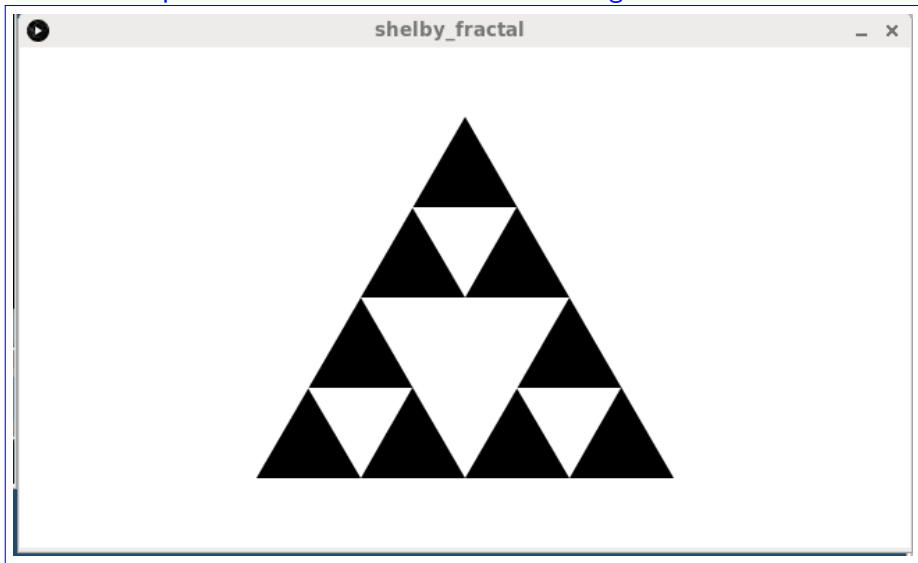








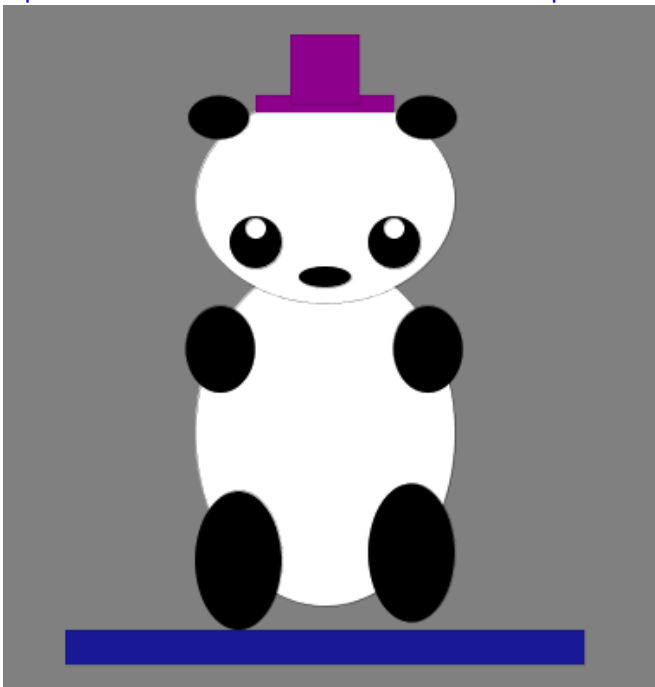
An example from a Fall 2013 student: drawing a fractal.



Another Fall 2013 student: ball bouncing between two springs



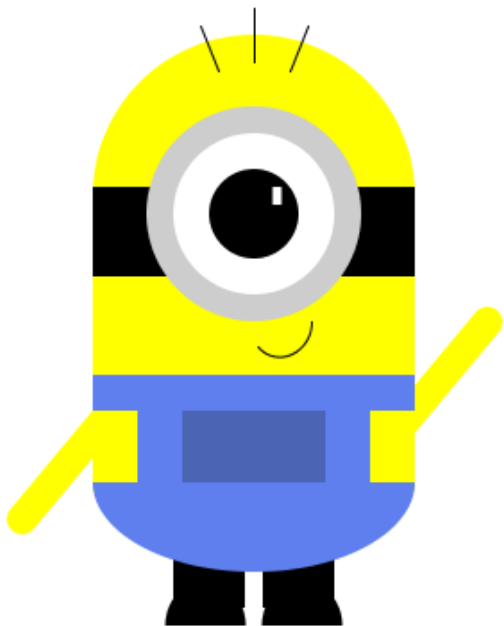
An example from a Fall 2015 student: an animated panda.



An example from a Fall 2015 student: a rotating fractal.



An example from a Fall 2015 student: a minion.



Fall 2015 student: bird moves where you move the mouse pointer.



Physics 8 — Wednesday, November 27, 2019

- ▶ I'll put take-home practice exam (due 12/6 or 12/9) online this evening. I put 4 years' old exams online at <http://positron.hep.upenn.edu/p8/files/oldexams/>
- ▶ **Extra-credit options (if you're interested):**
 - ▶ Learn to use Mathematica (ask me how), which is a system for doing mathematics by computer. (It is the brains behind Wolfram Alpha.) Penn's site license makes Mathematica free-of-charge for SAS and Wharton students.
 - ▶ Use "Processing.py" (or ordinary "Processing") to write a program to draw or animate whatever interests you. (Not necessarily physics-related.)
 - ▶ Read O/K ch9 on columns & summarize what you learned.
 - ▶ Read Mazur ch13 on gravity & summarize what you learned.
 - ▶ Go through Prof. Phil Nelson's book on using Python for data modeling. Several Huntsman students seem keen to do this.
 - ▶ Respond to podcast about near-fatal flaw in Citigroup Center, 601 Lexington Ave, NYC

http://positron.hep.upenn.edu/p8/files/citicorp_tower.mp3

- ▶ Pursue your own XC idea: [phys/math/coding/structures/etc](#)
- ▶ **Today's examples online at**

<http://positron.hep.upenn.edu/p8/2018/files/pyprocessing/>