

Department of Computer Science and Technology

Applicant Experience - Asteroids in Processing Sheet 4: Triangles

Currently, we have a circle underneath our cursor. This is fine, but a little bit bland. Let's mix things up and try making our own shape. As our end-goal here is Asteroids, let's make some form of triangle. (Yes, technically Asteroids wasn't exactly a triangle, more of a Star Trek-esque badge, but work with me here).

Let's start with the basics. What are Triangles made up of? Three Points. We call each point here a 'vertex', and when we're dealing with multiple of them together we say 'vertices'. As we eventually want our asteroids spaceship to be this triangle, we want the 'ship' centered on our position (for now that's our cursor). So we're going to need to do a bit of mathematics to figure out where to place our vertices.

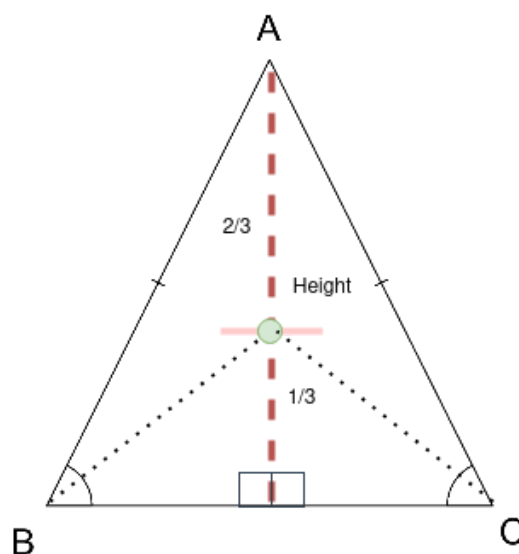


UNIVERSITY OF HULL

We can use an isosceles triangle. The bottom two angles are the same, and the upper two sides are the same length as each other. We want our position to be the center of this triangle. For simplicity, we can assume that if we draw a line straight down from A to the floor (bisecting the triangle), we would find the center of our triangle $\frac{2}{3}$ of the way down to the floor.

To make things easier for ourselves, we're going to cheat a little bit. Let us assume that the 'center' of this triangle is the center of our world - that is to say (0,0). That makes Point A easy to calculate, it's some distance above us (in the negative y direction).

Let's say we want point A to be 50 pixels above us. This means it would have coordinates (0, -50). Now we know that 50 represents $\frac{2}{3}$ of the total height of our triangle. If $50 = \frac{2}{3} H$ then $H = 50 * \frac{3}{2}$. As points B, and C are $\frac{1}{3} H$ below us (and a bit out to the side), we know their y-coordinates; 25 pixels in the positive y-direction, (?,25) for both. Depending on how pointy we want our triangle we can push out to either side by an x-amount. Let's just say 25 pixels to either side. Making our B & C points, (-25, 25), and (25, 25). Now we have our three points.



We can define these new points using something called a PVector. This is just a single point in some coordinate space. Perfect for what we want to use. It gives us the ability to store x and y in a single variable. Much more convenient.

```
PVector pA = new PVector( 0, -50 );  
PVector pB = new PVector( -25, 25 );  
PVector pC = new PVector( 25, 25 );
```

But how do I get the x-coordinate, and y-coordinate back out? We can use the variable name itself and something called dot notation. This lets us access components of the PVector. Not surprisingly these include x and y coordinates.

Task 8

Try the following:

1. Remove all print, or println statements from your sketch (they will get in the way).
2. Add the above 3 PVector lines for pA, pB, and pC.
3. After those are defined, add `println(pA.x);`
4. Do this for the .y component too.
5. Try this for points pB, and pC. Do these line up with what we worked out?

Sub-Task - Can you do some whizzy string combinations like what we did with frame rate? E.g So my console will say "pA is 0, -50" followed by "pB is -25, 25".

Finally, let's put this to good use. Similar to the `circle()` function, we have a `triangle()` function. This accepts 6 coordinates - 2 for each vertex.

Try the following:

```
triangle(pA.x, pA.y, pB.x, pB.y, pC.x, pC.y);
```

If we run the sketch now we should get something drawn to the screen every frame.



Task 9

Question: Did the previous task do what you expected it to do?

We now have what looks like it might be a triangle, however it's stuck in the top left corner. This isn't too surprising, as the origin of our screen (0,0) is the very top left. Some of our coordinates are OUTSIDE of our drawing area...

If we want to center this on our cursor we're going to need to translate it around the screen. Fancy words for moving it left, right, up, or down. Thankfully, this is simple. We can add our cursor x position, and y position to our triangle coordinates. This will slide the triangle so that it sits directly on our cursor (and this is exactly why we defined the center of our triangle as (0,0) when writing the vertices).

I want point A of my triangle to not be 50 units above my canvas origin (0,0), I want it 50 units above my cursor instead. PVectors have some additional functionality built into them that let us do some manipulation easily.

Change your triangle code to the following:

```
PVector pA = new PVector( 0, -50 );
PVector pB = new PVector( -25, 25 );
PVector pC = new PVector( 25, 25 );

// Make it a variable, as we're going to reuse it!
// Three times. No need to recalculate.
PVector cursorPos = new PVector( mouseX, mouseY );

// We need to make a copy of our triangle here.
PVector moved_pA = pA.copy();
moved_pA.add( cursorPos );

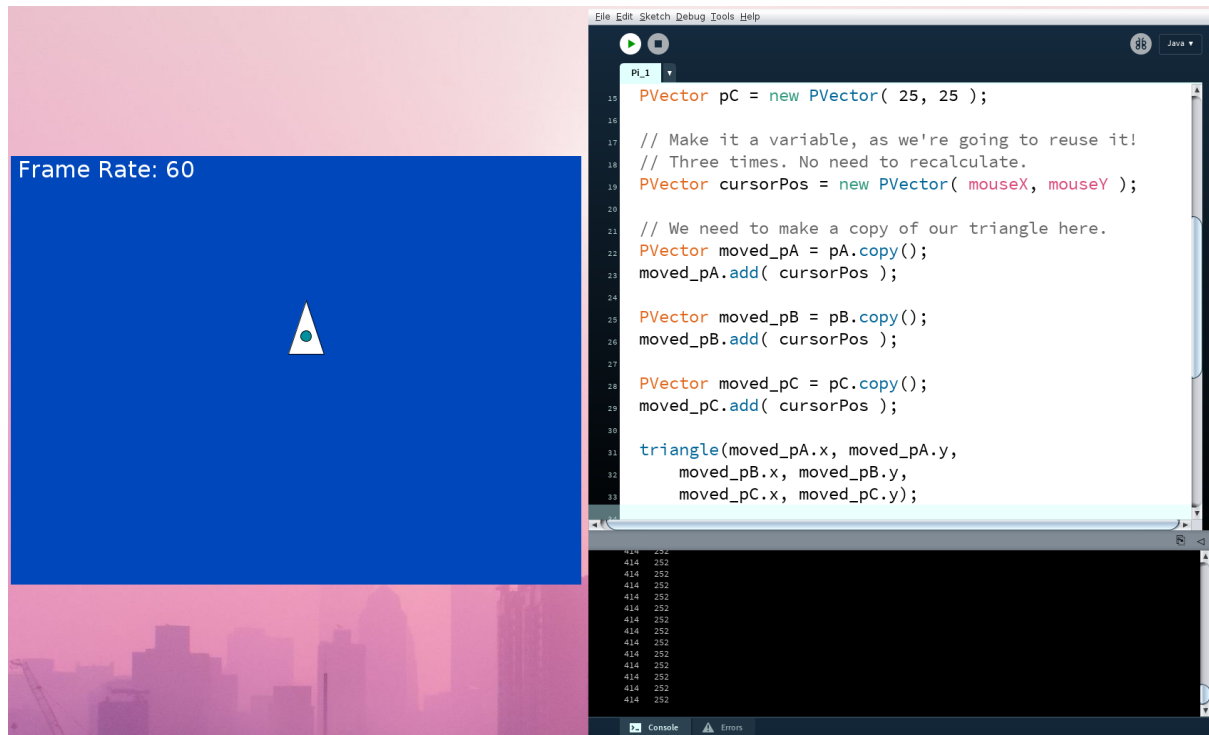
PVector moved_pB = pB.copy();
moved_pB.add( cursorPos );

PVector moved_pC = pC.copy();
moved_pC.add( cursorPos );

triangle(moved_pA.x, moved_pA.y,
        moved_pB.x, moved_pB.y,
        moved_pC.x, moved_pC.y);
```

We can create a vector to store our mouseX and mouseY in, then we don't have to write it out all the time (efficient). For each vertex we make a copy (as we're going to change it), then we simply add our cursor position vector to this. We then changed our triangle call to use the new moved points.

If all went well, we should now have the triangle following us around like the circle.



Task 10

Now we have something pointy, we can tell which orientation / heading we're facing. Very important for a spaceship with guns to know which way it is pointing.

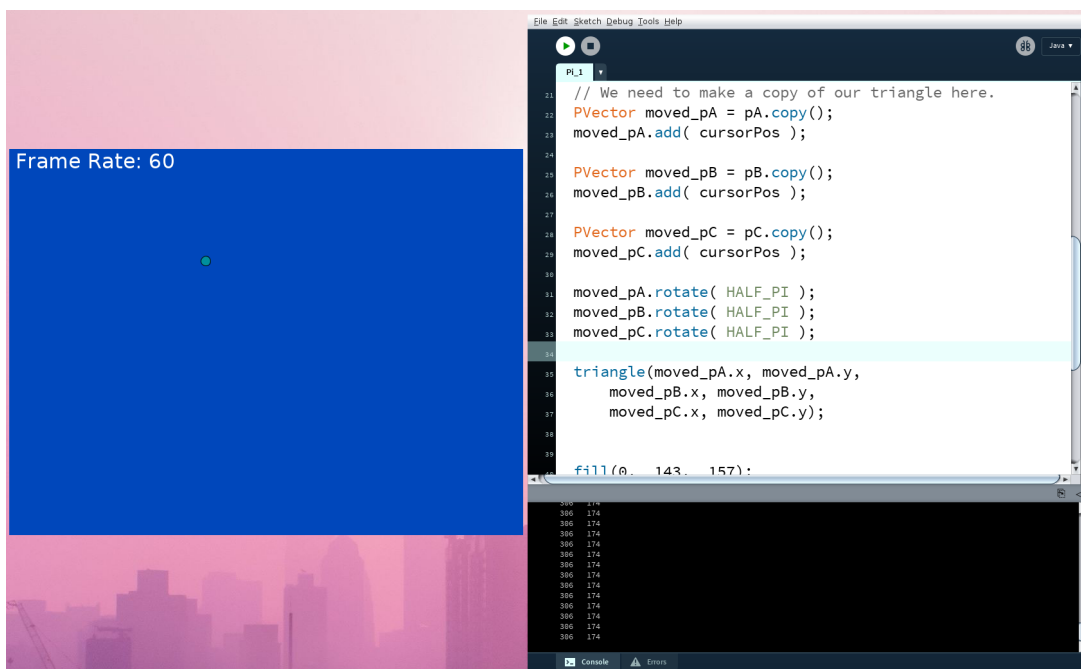
At some point we're going to need to stop facing straight up, and face incoming threats. Let's introduce rotations. At first, wild rotations like we're spinning out of control!

Remember how I said PVectors have some nice things attached to them, kind of like a kitchen sink full of possible utilities? Well, rotation is one of those. We don't need to do any fancy mathematics to rotate our points every time, the mathematics was complex enough as-is, we'll let Processing handle that. I just want to give it an angle to move by, and have it figure the rest out!

We can take any PVector (E.g pA) and use the `.rotate()` defined on it. This will move the points by the angle provided. However, this is in Radians. But we know that 360 Degrees are in a circle, equivalent to spinning all the way back around, and 2π Radians = 360 Degrees. This would make a half-turn be π Radians worth of turning. Processing has some predefined variables for π , so those of you who memorised 100 digits of π can rest your fingers. (See <https://processing.org/reference/PI.html> for more information. They even support Tau, my favourite).

Add the following snippet between your translation, and your drawing lines.

```
moved_pA.rotate( HALF_PI );
moved_pB.rotate( HALF_PI );
moved_pC.rotate( HALF_PI );
```



Where did it go?

When dealing with vectors, all of our rotations are by some angle around the origin (0,0). In our case, this is the top-left corner.

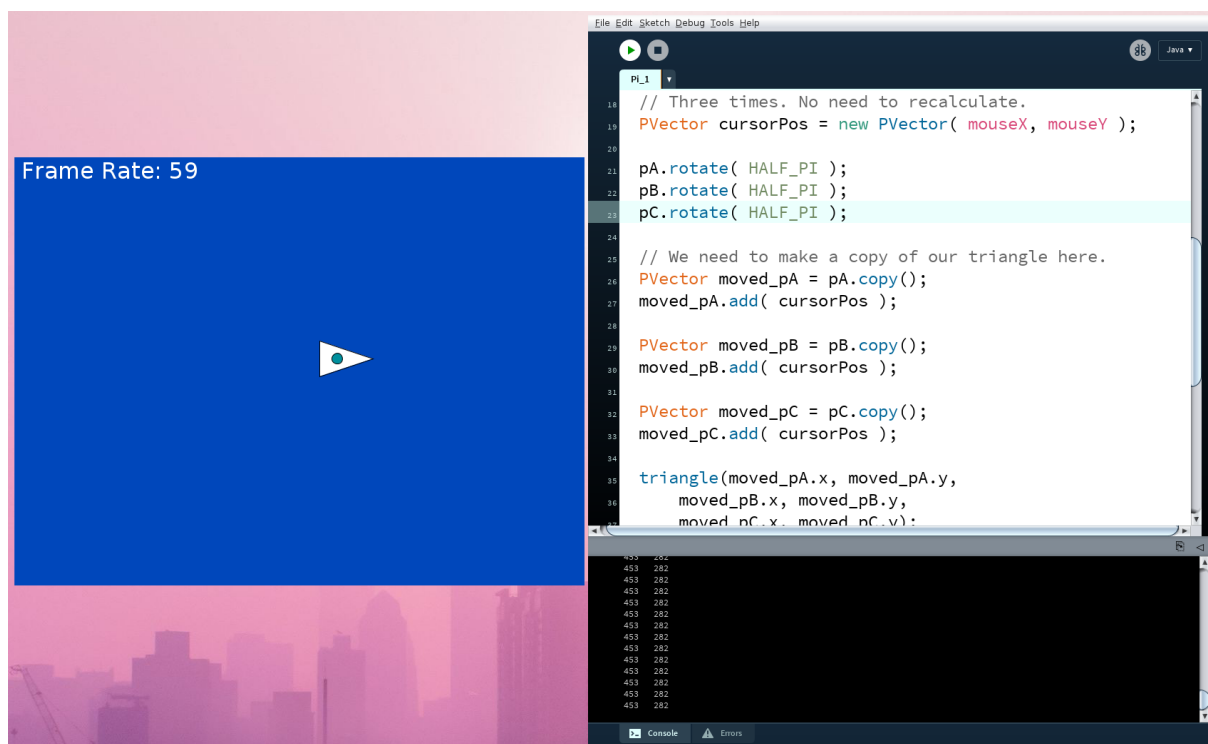
This isn't what we intended when we wanted to spin our spaceship. We need to spin the vectors whilst they are still in the top-left corner - that is to say BEFORE we move them over our cursor.

In general when we manipulate shapes like this (squares, etc), we move them back to the origin, rotate, then move them back to where they should be. Tedious... isn't it.

However, we have our original local pA, pB, and pC where we made the canvas and our triangle's origin align. If we rotate those, it would be the same as rotating around the center of our triangle! Let's try it.

```
pA.rotate( HALF_PI );
pB.rotate( HALF_PI );
pC.rotate( HALF_PI );
```

It doesn't look much different, but we have to make sure that these are all rotated BEFORE we copy them and use them everywhere. Otherwise we'll be rotating vectors and not using them. Change your .rotate lines, and move them up before we started doing .copy() and after the definition of pA, pB, pC.

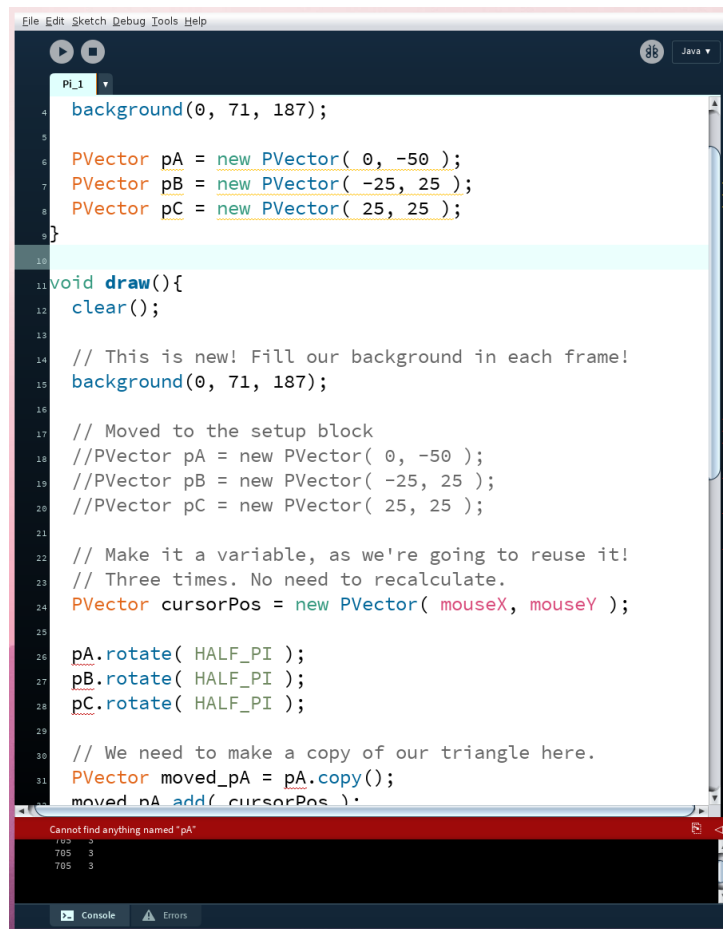


Fantastic, we've rotated it 90 degrees (Half PI) clockwise. We still have an issue, it's locked at that heading. It's always facing right now.

Task 11

The reason for this is that each frame drawn remakes the vectors pA, pB, pC (i.e. The triangle facing straight up). We then rotate each point 90 degrees clockwise, then draw it. If we want it to spin, we need to not constantly reset the pA, pB, pC vectors. We only want to make our triangle once, then simply rotate it by an amount each frame. This should turn it into the beyblade that we want.

Try moving the PVector pA, PVector pB, and PVector pC lines into the setup block. This seems like it might work, as that is only made once.



```

File Edit Sketch Debug Tools Help
Pi_1
4 background(0, 71, 187);
5
6 PVector pA = new PVector( 0, -50 );
7 PVector pB = new PVector( -25, 25 );
8 PVector pC = new PVector( 25, 25 );
9
10
11 void draw(){
12   clear();
13
14   // This is new! Fill our background in each frame!
15   background(0, 71, 187);
16
17   // Moved to the setup block
18   //PVector pA = new PVector( 0, -50 );
19   //PVector pB = new PVector( -25, 25 );
20   //PVector pC = new PVector( 25, 25 );
21
22   // Make it a variable, as we're going to reuse it!
23   // Three times. No need to recalculate.
24   PVector cursorPos = new PVector( mouseX, mouseY );
25
26   pA.rotate( HALF_PI );
27   pB.rotate( HALF_PI );
28   pC.rotate( HALF_PI );
29
30   // We need to make a copy of our triangle here.
31   PVector moved_pA = pA.copy();
32   moved_pA.add( cursorPos );
33 }

```

Cannot find anything named "pA"
 109 3
 705 3
 705 3

Uh-Oh. We have so many underlined things now. In our draw() block, it no longer recognises pA, pB, and pC as variables. Yet in our setup, it's then complaining that pA, pB, and pC are unused? What's going on?!?

This is because within those curly braces is like a new world. By defining the variables themselves (PVector pA) inside of setup { }, we've locked it in a box. It can't get out to be used anywhere else, only inside that block.

To fix this we can make it global, so everybody can see it! At the very top of the file, the very first few lines, let's put them there. Then the setup block and the draw block can see that "Yes, there is a variable called pA" and it shouldn't complain about it not existing.

Adapt your code so it looks like the following (for brevity, I've omitted some lines which we're not touching):

```
PVector pA;
PVector pB;
PVector pC;

void setup(){
  // Window + background code.

  // Initially set the value
  pA = new PVector( 0, -50 );
  pB = new PVector( -25, 25 );
  pC = new PVector( 25, 25 );
}

void draw(){
  // Clearing and background setting code, keep it.

  // Make it a variable, as we're going to reuse it!
  // Three times. No need to recalculate.
  PVector cursorPos = new PVector( mouseX, mouseY );

  // We still have access!
  pA.rotate( HALF_PI );
  pB.rotate( HALF_PI );
  pC.rotate( HALF_PI );

  // We need to make a copy of our triangle here.
  PVector moved_pA = pA.copy();
  moved_pA.add( cursorPos );

  PVector moved_pB = pB.copy();
  moved_pB.add( cursorPos );

  PVector moved_pC = pC.copy();
  moved_pC.add( cursorPos );

  triangle(moved_pA.x, moved_pA.y,
           moved_pB.x, moved_pB.y,
```

```
moved_pC.x, moved_pC.y);
```

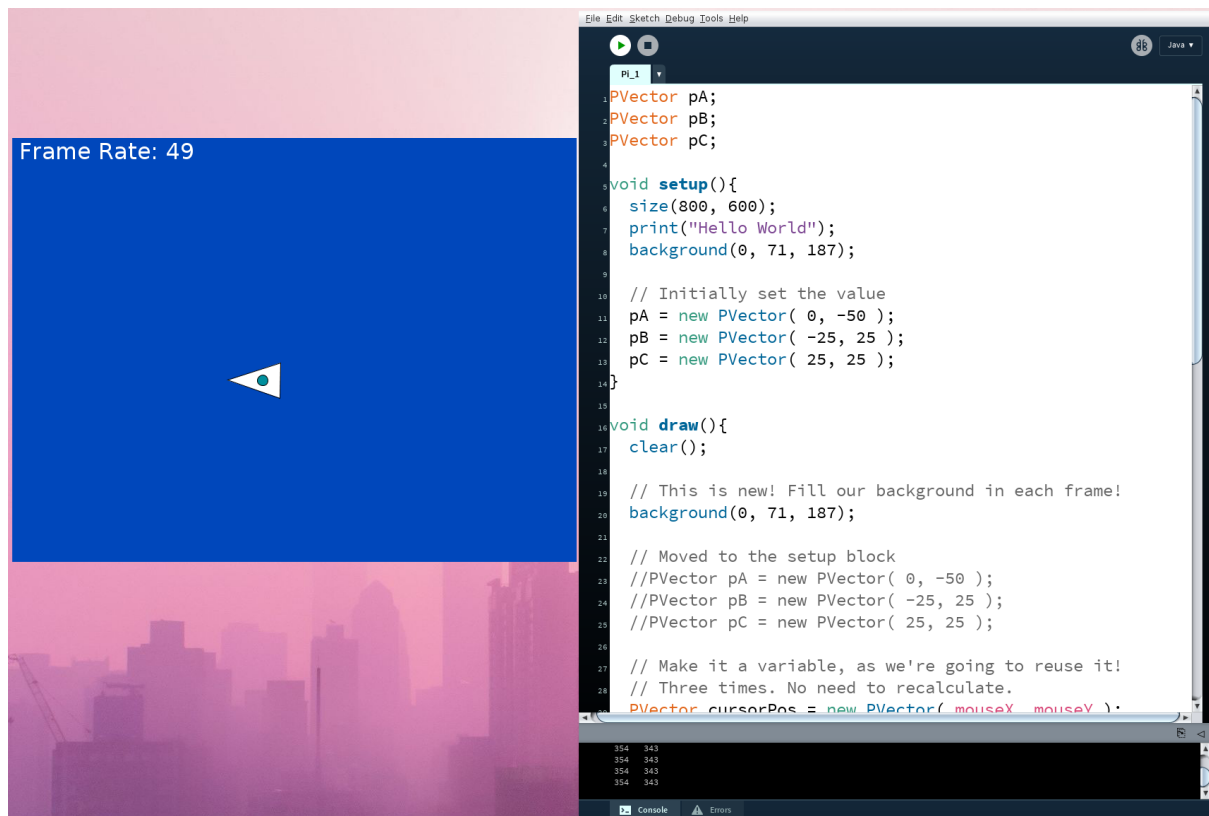
```
    /// Rest of draw  
}
```

I have put comments where I've left out some code, make sure your versions have that code in there such as `size()` and `background()` and `clear()`, etc.

At the very top of the file we have declared there will be a variable `pA`, which is a `PVector`. But it won't yet have a value at that point. Once `setup()` is called, then we say this `pA` `PVector` which is currently null and doesn't have any data, is equal to `(0, -50)`. Now we get to each draw frame, `pA` is not only able to be accessed, but it actually has a value. We then rotate it.

As we are not constantly defining `pA` as `(0, -50)`, we're not getting our upwards triangle each frame. `pA` at any point will have different values (as we're rotating the whole triangle).

If you run your code now you should see a very fast spinning triangle (maybe a little too fast), which is beginning to look suspiciously like a shuriken - which I cannot capture in a still image.



Try experimenting with different amounts of rotation. Turning 90 degrees every frame, at ~60 FPS means we're spinning at 15 rotations per second! Maybe changing the `HALF_PI` to some smaller value will make it appear smoother...

Full Code thus far:

```
PVector pA;
PVector pB;
PVector pC;

void setup(){
  size(800, 600);
  print("Hello World");
  background(0, 71, 187);

  // Initially set the value
  pA = new PVector( 0, -50 );
  pB = new PVector( -25, 25 );
  pC = new PVector( 25, 25 );
}

void draw(){
  clear();

  // This is new! Fill our background in each frame!
  background(0, 71, 187);

  // Make it a variable, as we're going to reuse it!
  // Three times. No need to recalculate.
  PVector cursorPos = new PVector( mouseX, mouseY );

  // We still have access!
  pA.rotate( HALF_PI );
  pB.rotate( HALF_PI );
  pC.rotate( HALF_PI );

  // We need to make a copy of our triangle here.
  PVector moved_pA = pA.copy();
  moved_pA.add( cursorPos );

  PVector moved_pB = pB.copy();
  moved_pB.add( cursorPos );

  PVector moved_pC = pC.copy();
  moved_pC.add( cursorPos );

  triangle(moved_pA.x, moved_pA.y,
    moved_pB.x, moved_pB.y,
```

```
moved_pC.x, moved_pC.y);

fill(0, 143, 157);
circle(mouseX, mouseY, 15);

// Let's make some FPS counter.
// We use String to refer to text.
// Just like how int was for counting numbers.
String my_text = "Frame Rate: " + (int)frameRate;
textSize(32); // 32pt Size.
fill(255); // White.
text(my_text, 10, 30); // Use my_text value, at (10,30)

println(mouseX, " ", mouseY);
}
```

We covered quite a lot in this sheet, both conceptually and programmatically. Feel free to change some of the values we've used here, such as our three triangle points. We can make any triangle we want! It's your program, you decide. If you want a wonky spaceship, you can!

We'll be building upon our spinning (rotating) ship in our next sheet.