Activity 4: 3D Objects and Transformation

Contributors include Keane Dalisay, Nel Alanan, and Prince Alexander Malatuba.

Cheers!

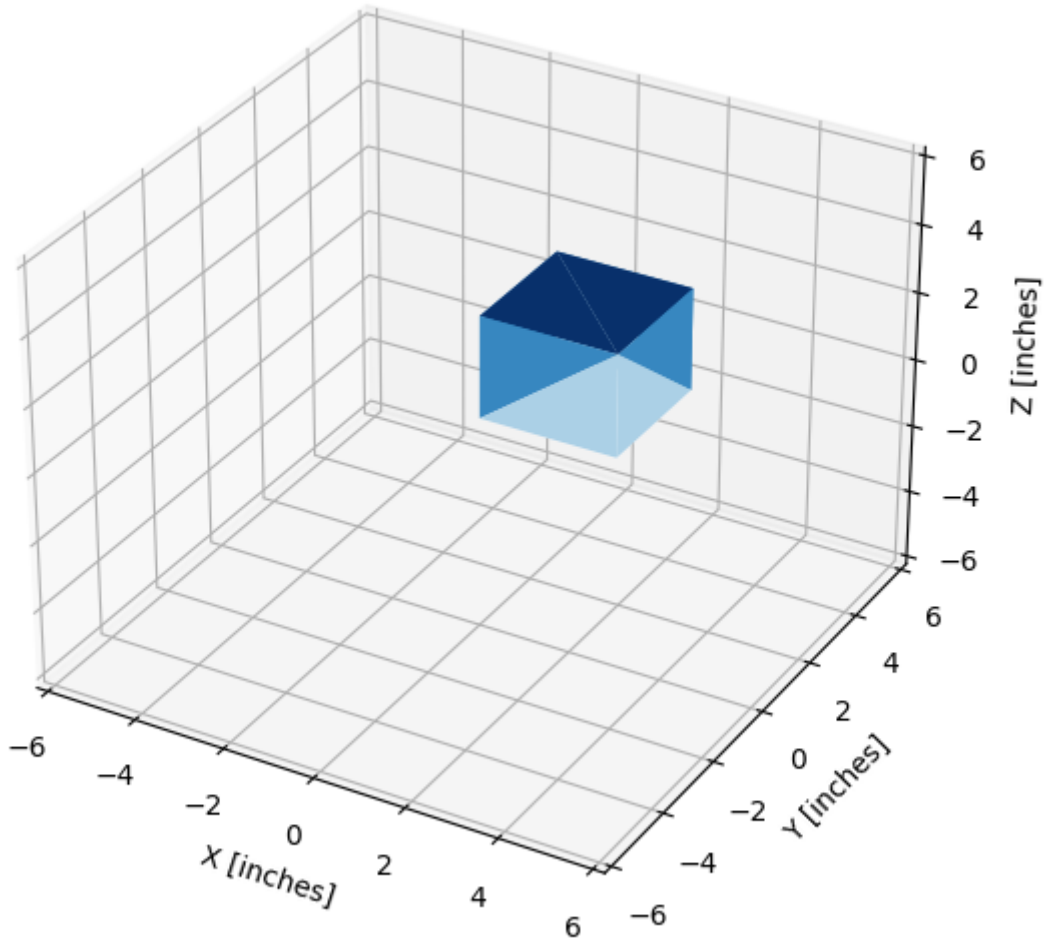
Inheritance

```
class Cube(Transform):
    def __init__(self):
         super().__init__()
         self.create()

    ...
```

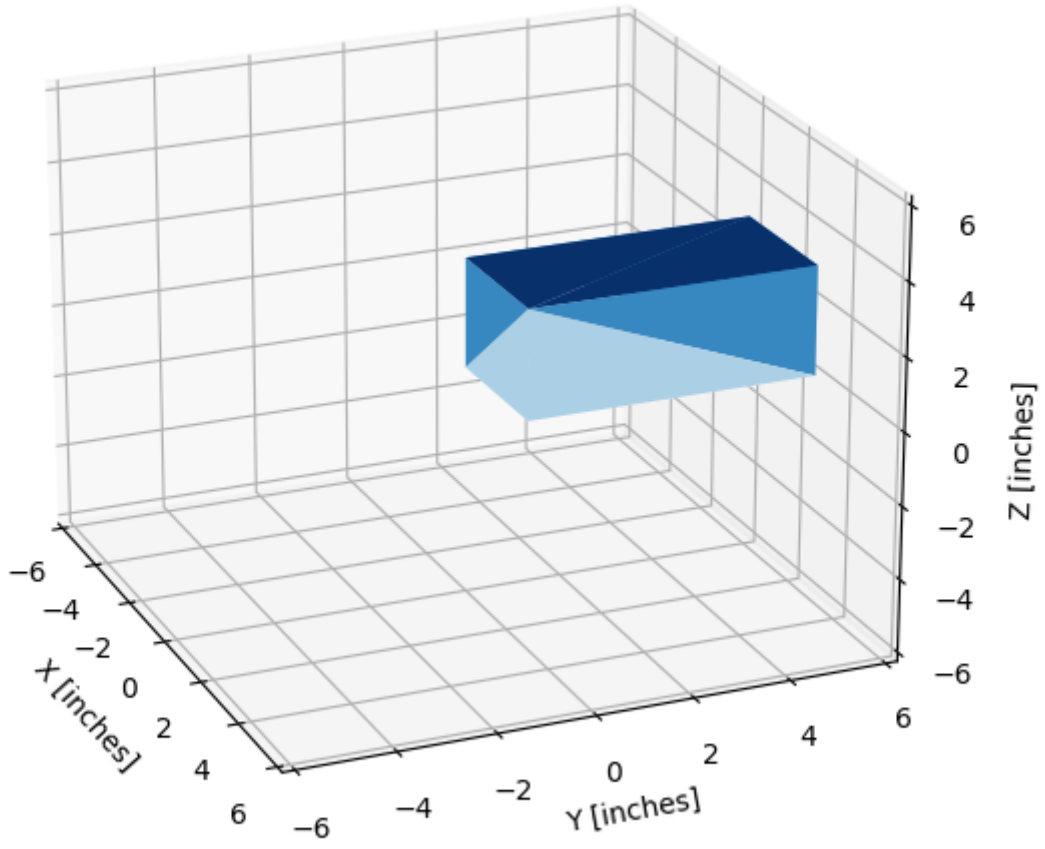
```
class Transform():

    def __init__(self):
        self.points = []
        self.pos = {'x': 0, 'y': 0, 'z': 0}
        self.size = 0

    ...
```

Every object class like `Cube()` inherits from a `Transform()` class for reusability. No matter the object, the same transformation functions from the class like `rotate` and `shear` apply.

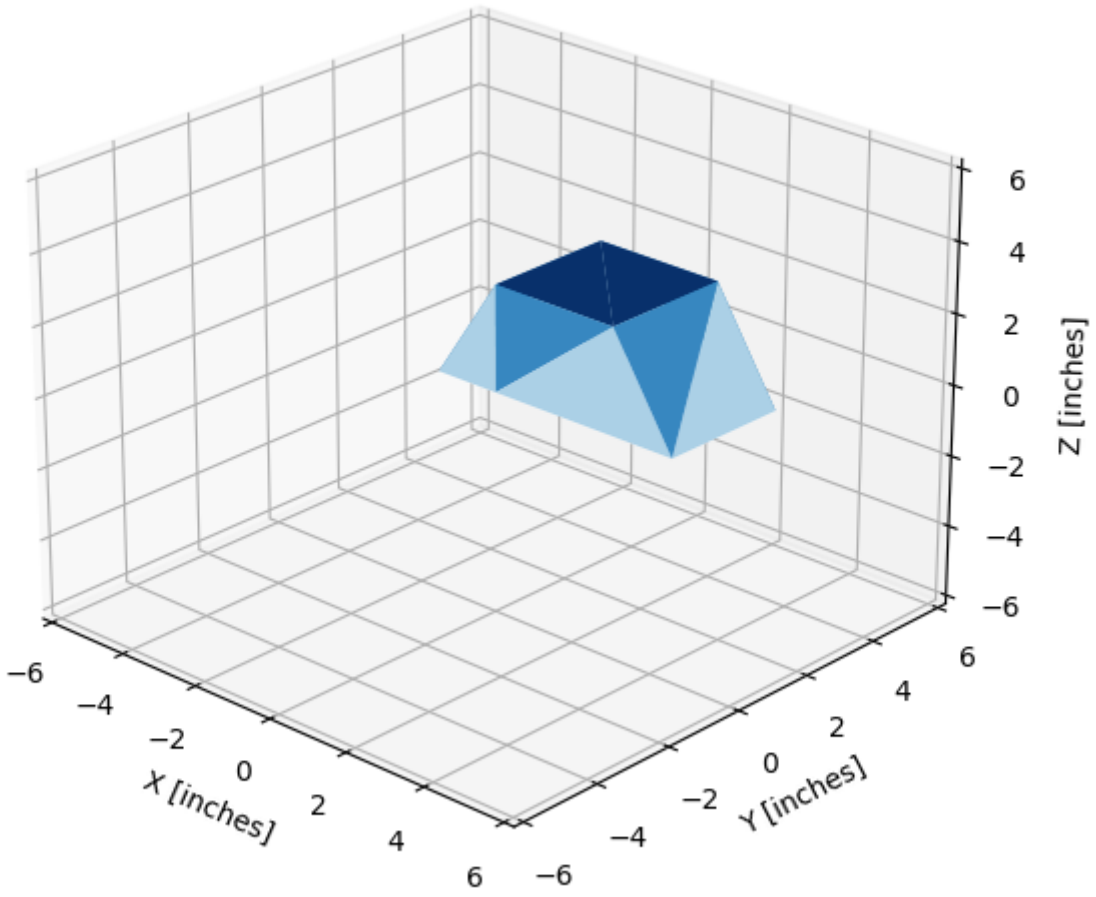
Five Objects, Five Classes



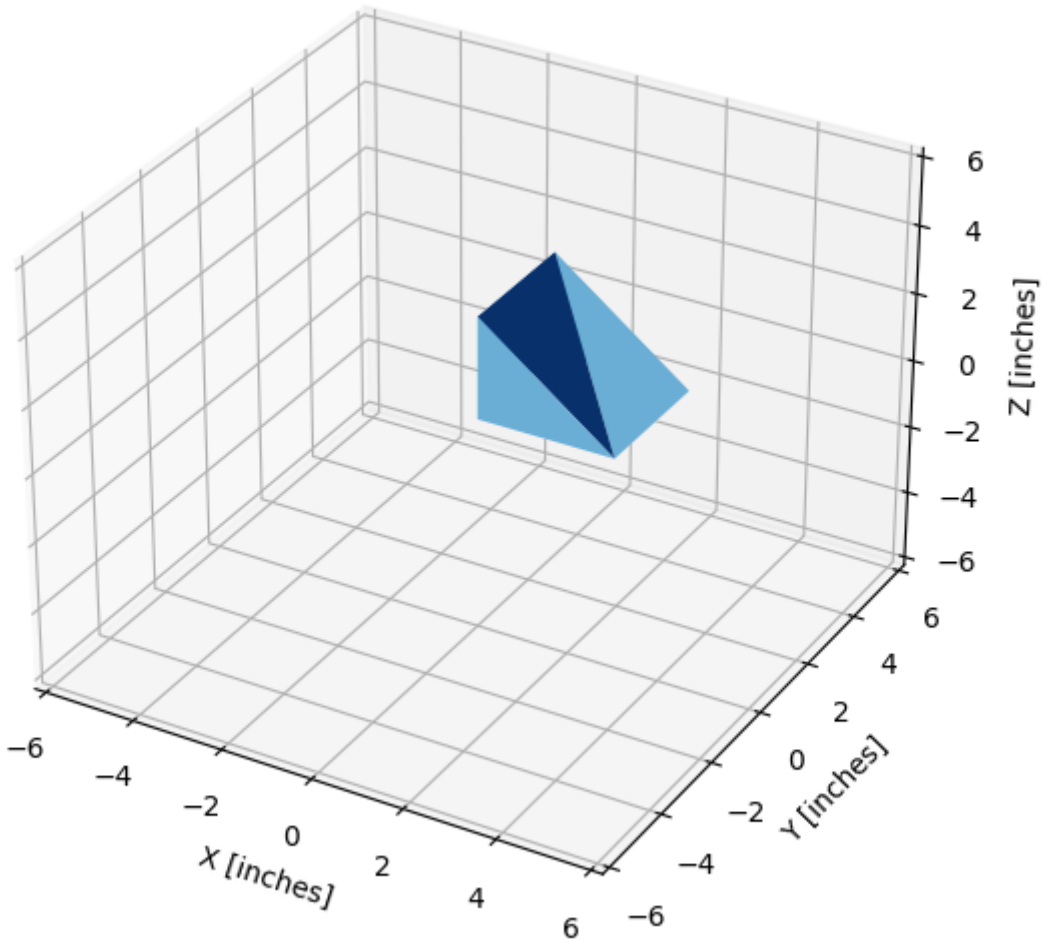
A 3 x 3 blue cube.



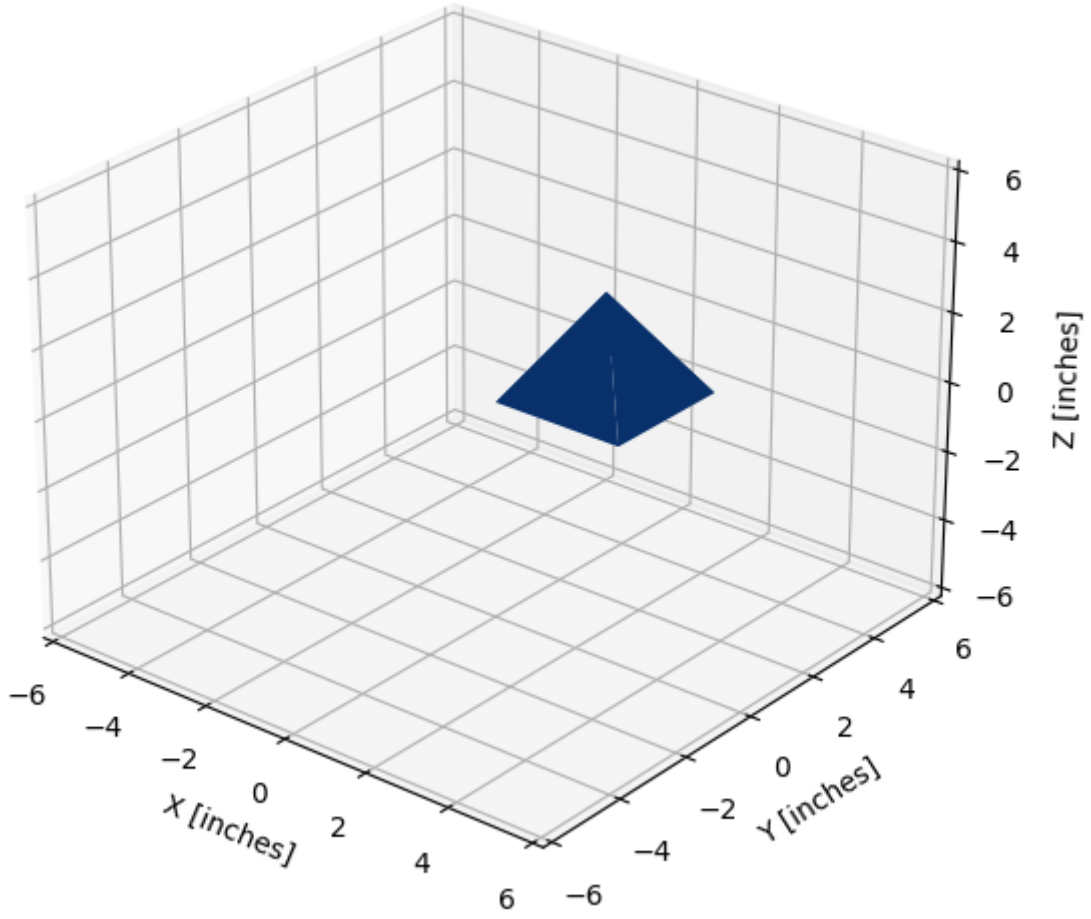
A 3 x 6 blue cuboid.



A 6 x 3 blue trapezoid.



A 3D blue right triangle.

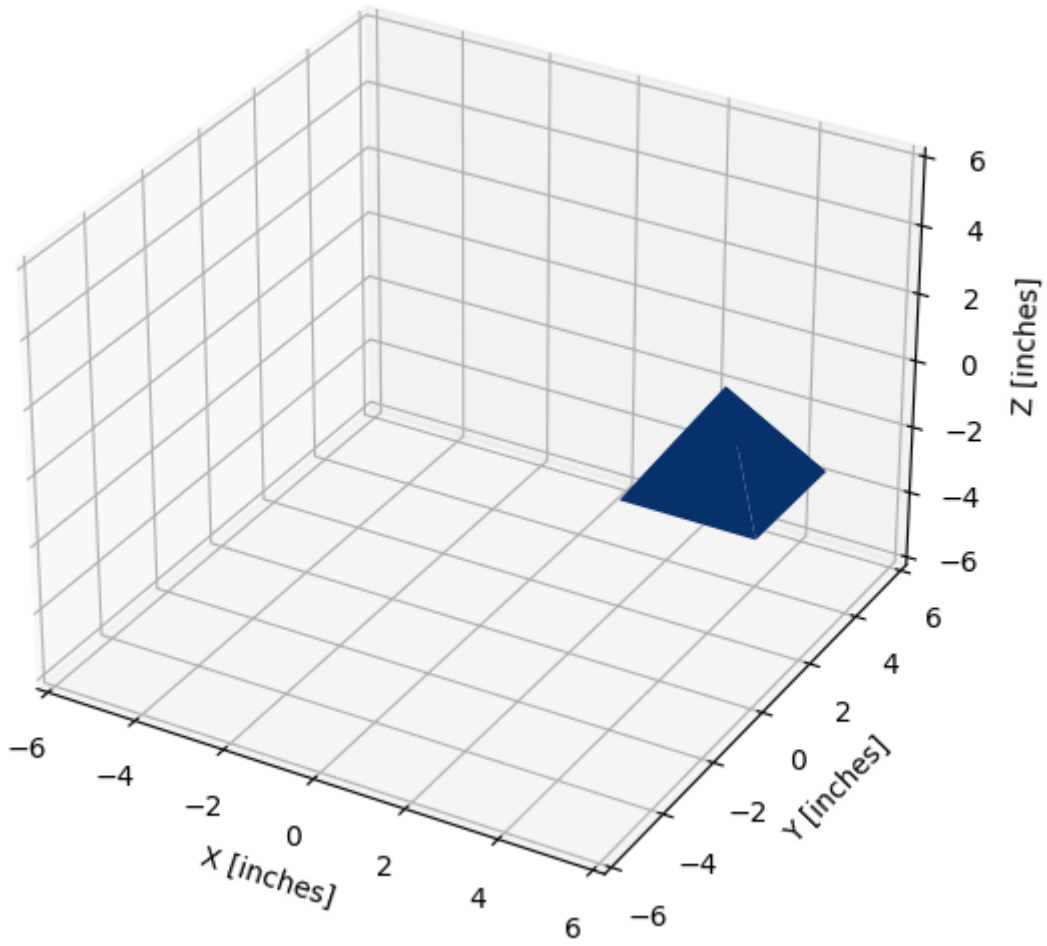


A 3D blue pyramid.

Transformations

All transformation functions work on the five object classes we've created. The results of the pictures below are replicable to any of them.

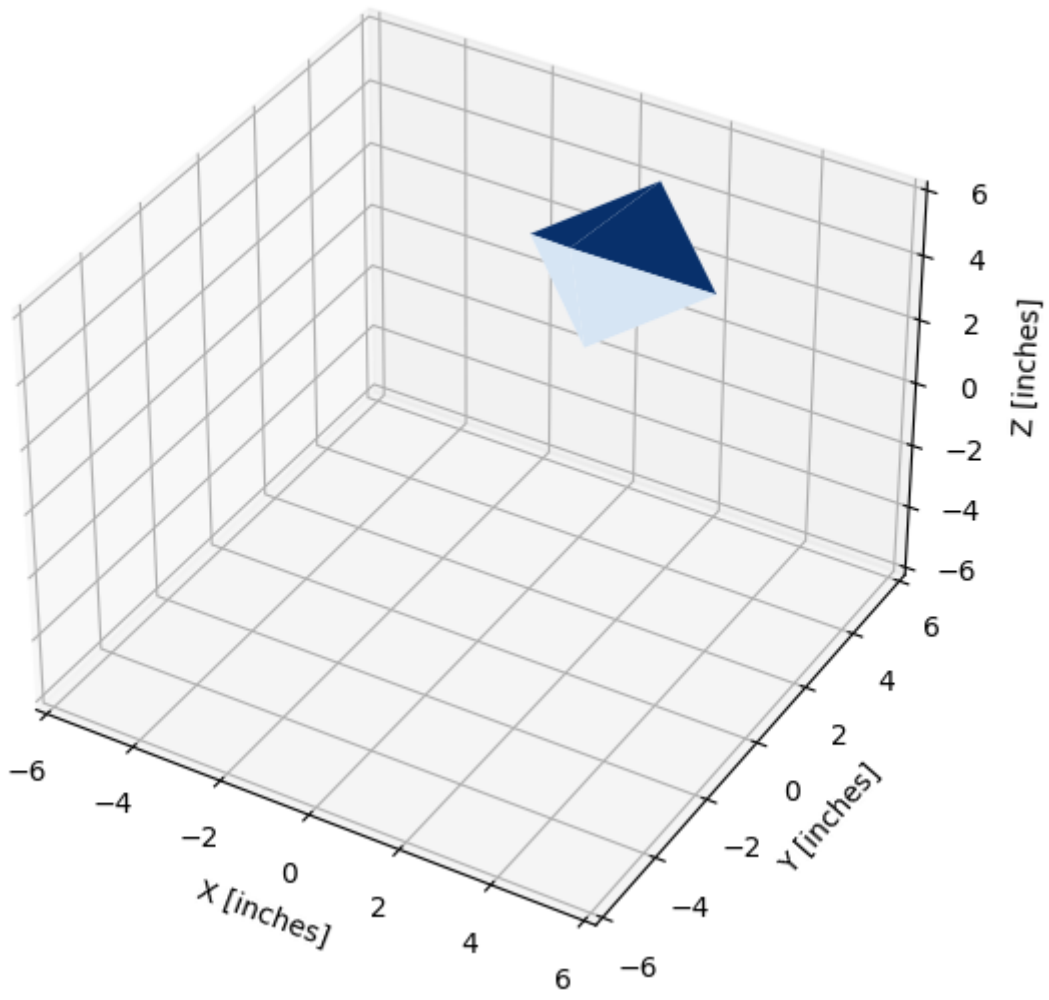
Translate



A blue pyramid translated 2 inches on the *X* and *Y* axis, and 2 inches less on the *Z* axis.

```
def translate(self):
    print('\n>>> Starting translation...\n')

    x = float(input('Inches to translate X: '))
    y = float(input('Inches to translate Y: '))
    z = float(input('Inches to translate Z: '))

    trans_am = tf.constant([x, y, z], dtype = tf.float64)
    trans_obj = tf.add(self.points, trans_am)
    self.points = runSession(trans_obj)
```

Rotate



A blue pyramid rotated 45 degrees on the *X* and *Z* axis.

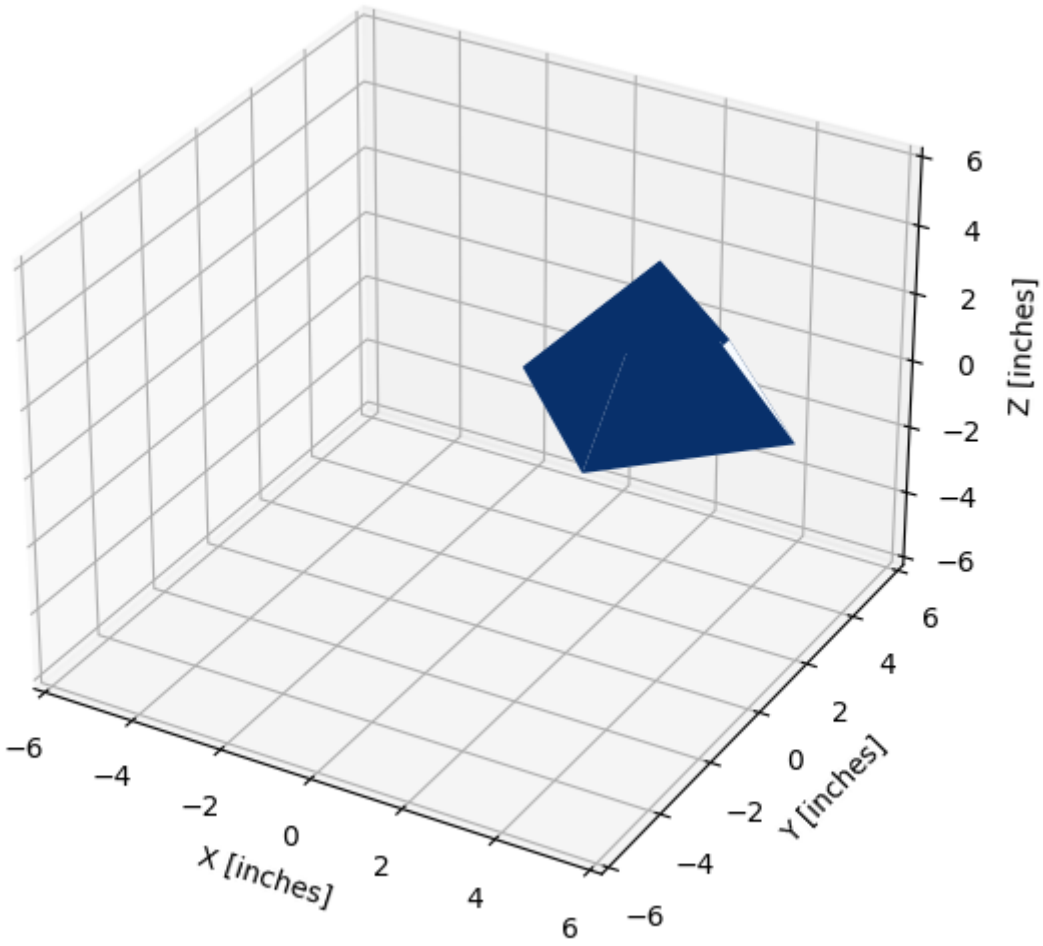
```
def rotate(self):
    print('\n>>> Starting rotation...\n')

    print('Rotate object in which axis?')
    print('\n(x) X-Axis', '\n(y) Y-Axis', '\n(z) Z-Axis')
    chc = input('\n: ')[0].lower()

    theta = math.radians(float(input('\nAngle to rotate object: ')))

    for i in range(len(self.points)):
        pos = {'x': self.points[i][0],
               'y': self.points[i][1],
               'z': self.points[i][2]}
        if (chc == 'x'):
            self.points[i] = rotateOnX(theta, pos)
        elif (chc == 'y'):
            self.points[i] = rotateOnY(theta, pos)
         elif (chc == 'z'):
            self.points[i] = rotateOnZ(theta, pos)
        else:
```

```
            exit()

    rotate_obj = tf.constant(self.points, dtype = tf.float64)
    self.points = runSession(rotate_obj)
```

```
def rotateOnX(theta, pos):
    return np.array([pos['x'],
            pos['y'] * math.cos(theta) - pos['z'] * math.sin(theta),
            pos['y'] * math.sin(theta) + pos['z'] * math.cos(theta)])
```

Function to calculate the angle of rotation per point on the *X* axis.

Scale

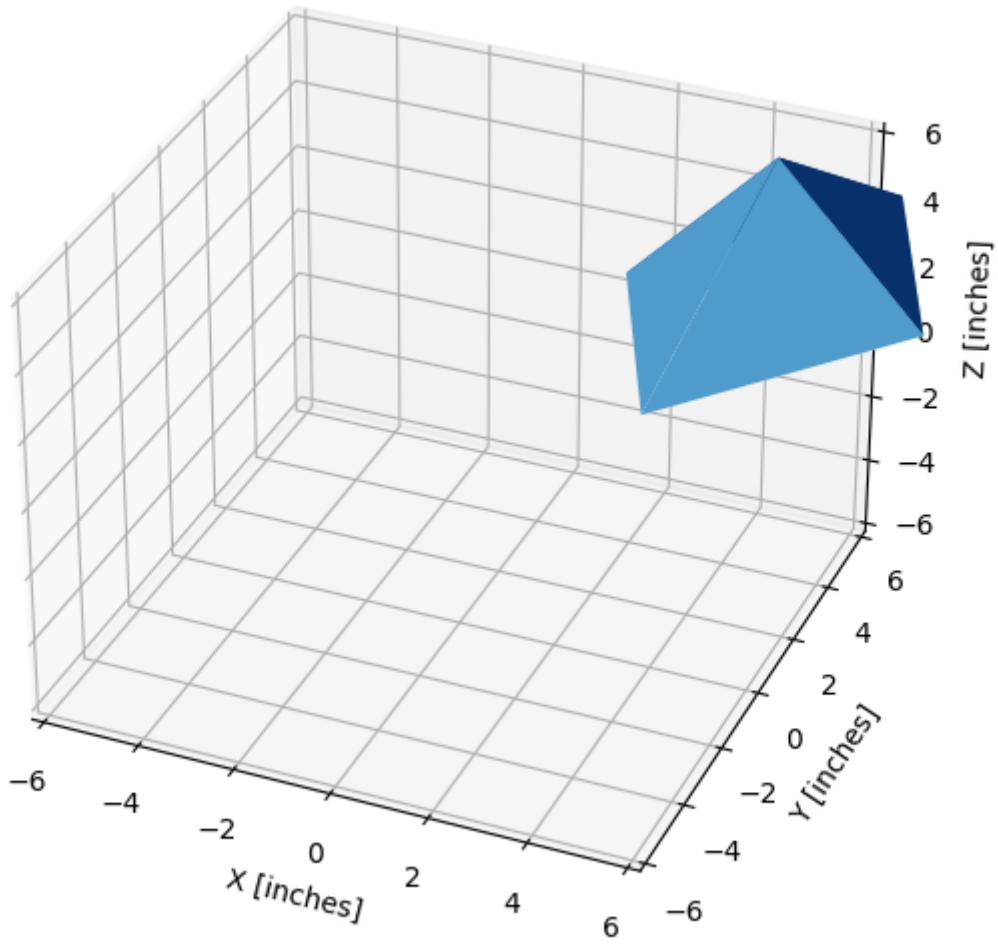


A blue pyramid scaled to 1.5 times its previous size on all axes.

```
def scale(self):
    print('\n>>> Starting scaling...\n')

    x = float(input('Multiplier to scale X: '))
```

```
    y = float(input('Multiplier to scale Y: '))
    z = float(input('Multiplier to scale Z: '))

    scale_am = tf.constant([x, y, z], dtype = tf.float64)
    scale_obj = tf.multiply(self.points, scale_am)
    self.points = runSession(scale_obj)
```

Shear



A blue pyramid sheared on the *Y* axis.

```
def shear(self):
    print('\n>>> Starting scaling...\n')

    print('Shear object in which axis?')
    print('\n(x) X-Axis', '\n(y) Y-Axis', '\n(z) Z-Axis')
    chc = input('\n: ')[0].lower()

    shr_am_one = 0
    shr_am_two = 0

    if (chc == 'x'):
```

```
        shr_am_one = getShearAm('y')
        shr_am_two = getShearAm('z')
    elif (chc == 'y'):
        shr_am_one = getShearAm('x')
        shr_am_two = getShearAm('z')
    elif (chc == 'z'):
        shr_am_one = getShearAm('x')
        shr_am_two = getShearAm('y')
    else:
        exit()

    for i in range(len(self.points)):
        pos = {'x': self.points[i][0],
                   'y': self.points[i][1],
                   'z': self.points[i][2]}
        if (chc == 'x'):
                self.points[i] = shearOnX(shr_am_one, shr_am_two, pos)
        elif (chc == 'y'):
                self.points[i] = shearOnY(shr_am_one, shr_am_two, pos)
        elif (chc == 'z'):
                self.points[i] = shearOnZ(shr_am_one, shr_am_two, pos)
        else:
                exit()

    shear_obj = tf.constant(self.points, dtype = tf.float64)
    self.points = runSession(shear_obj)
```

```
def shearOnX(shY, shZ, pos):
    return np.array([pos['x'],
                     pos['y'] + pos['x'] * shY,
                     pos['z'] + pos['x'] * shZ])
```

Function to calculate the shear amount per point on the *X* axis.

```
def getShearAm(ax):
    shear_am = float(input('\nMultiplier to ' + ax + ' for shearing: '))
    return shear_am
```

Function to get shear amount.