INDIVIDUAL PROJECT

Benjamin Whelan

Fontys ICT
Cybersecurity Advanced

Contents

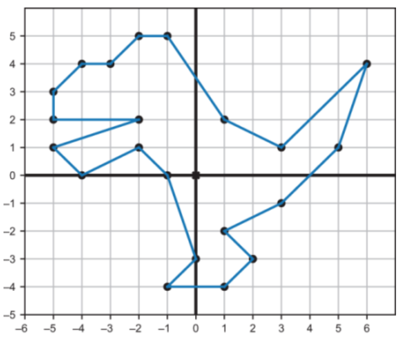
Version

Version (v)	Date	Changes
1	12/09/2022	Introduction Added
2	14/09/2022	Exercises 2.1
3	20/09/2022	Exercises 2.2/2.3
4	26/09/2022	Exercises 2.4
5	03/10/2022	Exercises & Theory Chapter 3
6	10/10/2022	Theory 4.1
7	17/10/2022	Answers Chapter 4 & 5.1

Introduction

I have decided to focus my individual project on learning math topics related to programming, since I feel like those areas have been lacking in my HBO study thus far. More importantly, through my work (which involves robots and embedded developers) I can feel my lack of knowledge in Math being something I wish to fill up. I will be working from a book whereby my proof of work will consist of exercises and mini projects. Below are exercises 2 till 5.1. Remaining exercises can be found on the repository itself.

2D Vectors

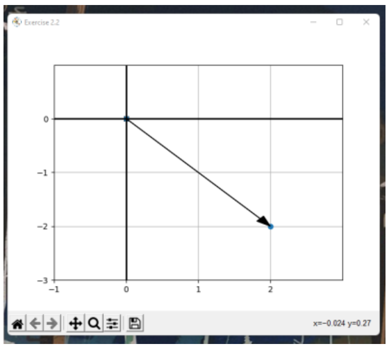


Exercise 2.1: What is the x- and y-coordinates of the point at the tip of the dinosaur's toe?

Answer: (-1, -4)

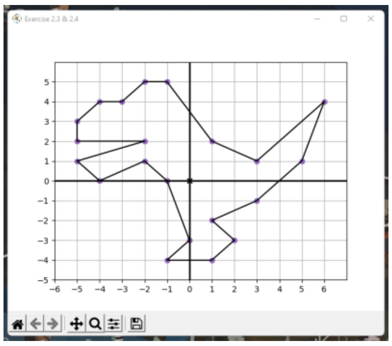
Exercise 2.2: Draw the point in the plane and the arrow corresponding to the point (2, −2).

Answer:

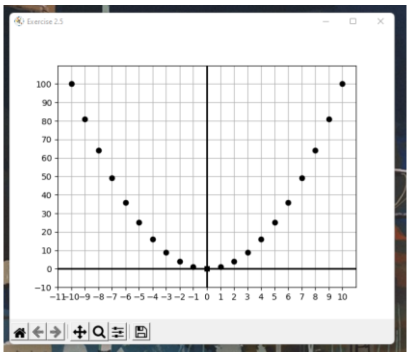


Exercise 2.3/2.4: Complete Dino and Segment Lines.

Answer:



Exercise 2.5: Draw the vectors (x,x**2) for *x* in the range from *x* = −10 to *x* = 11) as points (dots) using the draw function. What is the result?

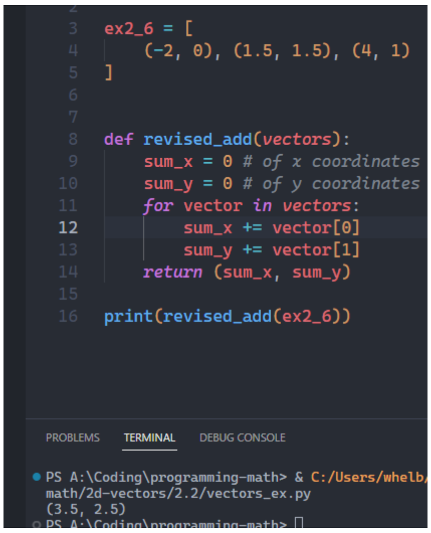


Exercise 2.6: If the vector **u** = (−2, 0), the vector **v** = (1.5, 1.5), and the vector **w** = (4, 1), what are the results of **u** + **v**, **v** + **w**, and **u** + **w**? What is the result of **u** + **v** + **w**?

Answer:

- u + v = (-0.5, 1.5)
- v + w = (5.5, 2.5)
- u + w = (2, 1)
- u + v + w = (3.5, 2.5)

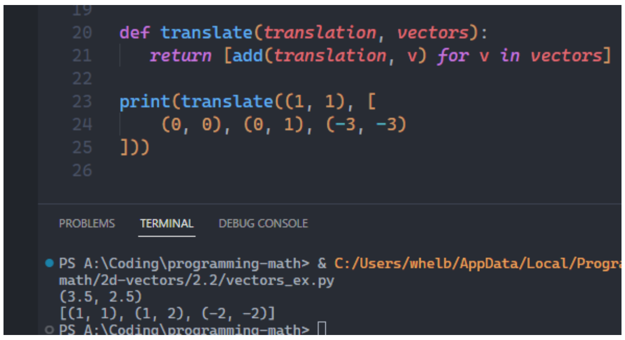
Exercise 2.7-Mini Project: Implement a revised add function that takes any number of vectors as arguments.



```python
ex2_6 = [
    (-2, 0), (1.5, 1.5), (4, 1)
]


def revised_add(vectors):
    sum_x = 0 # of x coordinates
    sum_y = 0 # of y coordinates
    for vector in vectors:
        sum_x += vector[0]
        sum_y += vector[1]
    return (sum_x, sum_y)

print(revised_add(ex2_6))
```

```
PS A:\Coding\programming-math> & C:/Users/whelb/
math/2d-vectors/2.2/vectors_ex.py
(3.5, 2.5)
PS A:\Coding\programming-math> 
```

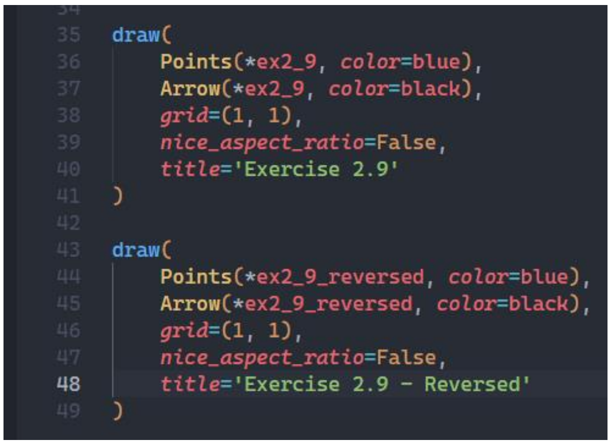
Exercise 2.8: Write a function translate (translation, vectors) that takes a translation vector and a list of input vectors, and returns a list of the input vectors all translated by the translation vector:



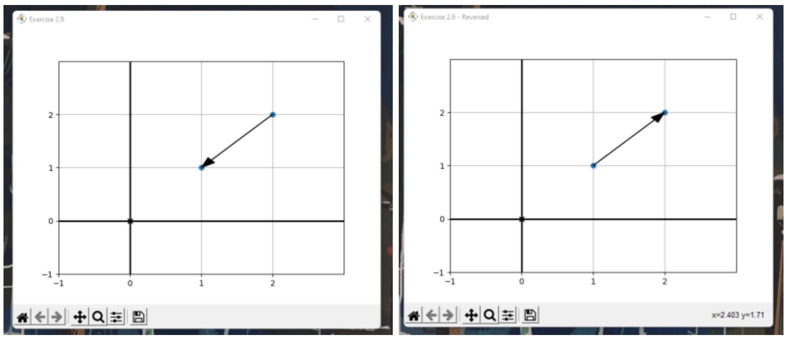
```python
def translate(translation, vectors):
    return [add(translation, v) for v in vectors]

print(translate((1, 1), [
    (0, 0), (0, 1), (-3, -3)
]))
```

```
PS A:\Coding\programming-math> & C:/Users/whelb/AppData/Local/Progra
math/2d-vectors/2.2/vectors_ex.py
(3.5, 2.5)
[(1, 1), (1, 2), (-2, -2)]
PS A:\Coding\programming-math> 
```

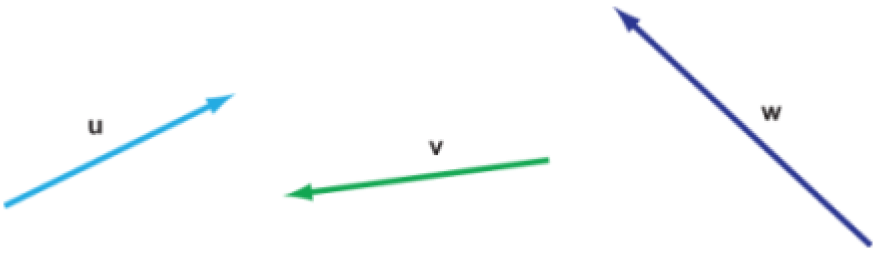
Exercise 2.9–Mini Project: Any sum of vectors **v** + **w** gives the same result as **w** + **v**. Explain why this is true using the definition of the vector sum on coordinates. Also, draw a picture to show why it is true geometrically.



```python
draw(
    Points(*ex2_9, color=blue),
    Arrow(*ex2_9, color=black),
    grid=(1, 1),
    nice_aspect_ratio=False,
    title='Exercise 2.9'
)

draw(
    Points(*ex2_9_reversed, color=blue),
    Arrow(*ex2_9_reversed, color=black),
    grid=(1, 1),
    nice_aspect_ratio=False,
    title='Exercise 2.9 - Reversed'
)
```

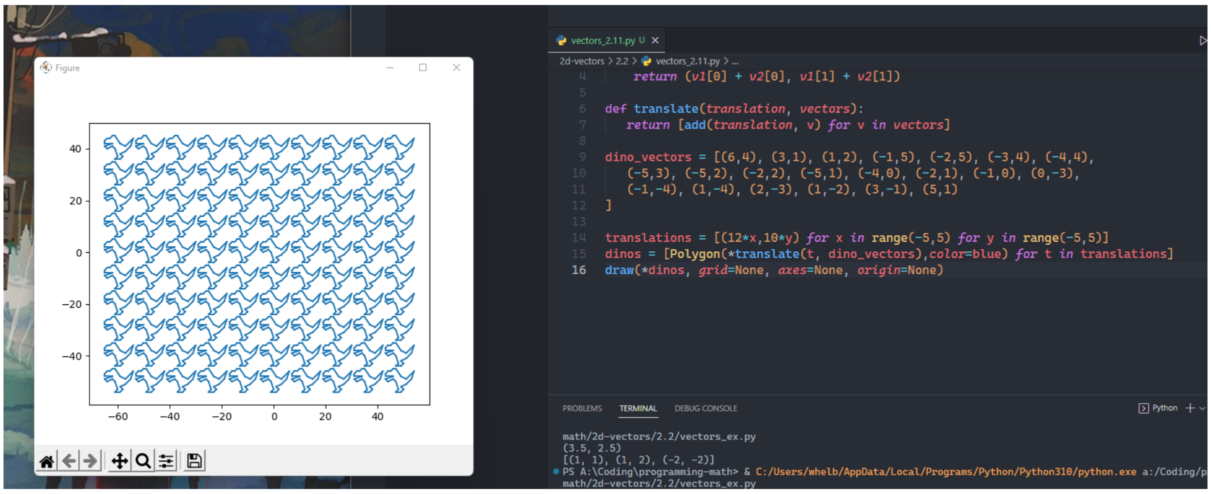


__**Exercise 2.10**: Among the following three arrow vectors (labelled **u**, **v**, and **w**), which pair has the sum that gives the longest arrow? Which pair sums to give the shortest arrow?__



Answer: If vectors are placed tip-to-tail we can measure the sums and see which is shortest.

__**Exercise 2.11-Mini Project**: Write a Python function using vector addition to show 100 simultaneous and non-overlapping copies of the dinosaur.__



```python
    return (v1[0] + v2[0], v1[1] + v2[1])

def translate(translation, vectors):
    return [add(translation, v) for v in vectors]

dino_vectors = [(6,4), (3,1), (1,2), (-1,5), (-2,5), (-3,4), (-4,4),
    (-5,3), (-5,2), (-2,2), (-5,1), (-4,0), (-2,1), (-1,0), (0,-3),
    (-1,-4), (1,-4), (2,-3), (1,-2), (3,-1), (5,1)
]

translations = [(12*x,10*y) for x in range(-5,5) for y in range(-5,5)]
dinos = [Polygon(*translate(t, dino_vectors),color=blue) for t in translations]
draw(*dinos, grid=None, axes=None, origin=None)
```

```
math/2d-vectors/2.2/vectors_ex.py
(3.5, 2.5)
[(1, 1), (1, 2), (-2, -2)]
PS A:\Coding\programming-math> & C:/Users/whelb/AppData/Local/Programs/Python/Python310/python.exe a:/Coding/p
math/2d-vectors/2.2/vectors_ex.py
```

__**Exercise 2.12**: Which is longer, the *x* or y component of (3, −2) + (1, 1) + (−2, −2)?__

The result is (2, -3), thus the y component is longer by 1 unit.

__**Exercise 2.13**: What are the components and lengths of the vectors (−6, −6) and (5, −12)?__

Components:

- (-6, 0)
- (0, -6)
- (5, 0)
- (0, -12)

Lengths: (Pythagoras theorem)

- 8.48528137423857
- 13

<u>**Exercise 2.14**: Suppose I have a vector **v** that has a length of 6 and an *x* component (1, 0). What are the possible coordinates of **v**?</u>

1 + y**2 = 36

y**2 = 35

y = 5.916079783099616

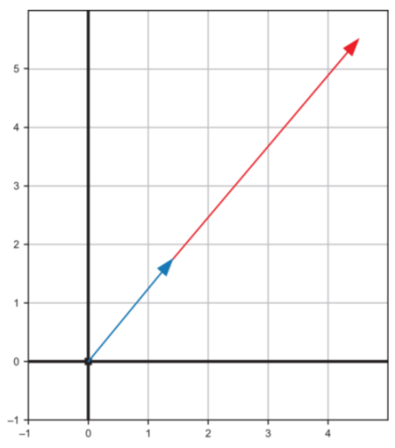
The vector's y component is either 5.9161 or -5.9161

<u>**Exercise 2.15**: What vector in the dino vectors list has the longest length?</u>

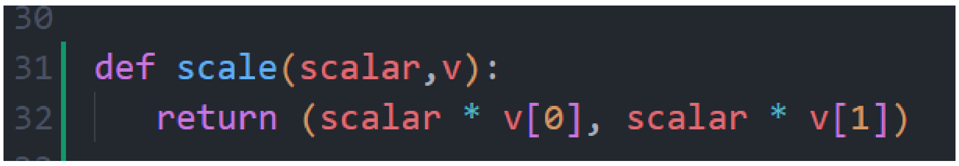
(6, 4)

<u>**Exercise 2.16**: Suppose a vector **w** has the coordinates (√2, √3). What are the approximate coordinates of the scalar multiple π · **w**? Draw an approximation of the original vector and the new vector.</u>

(4.442882938158366, 5.441398092702653)

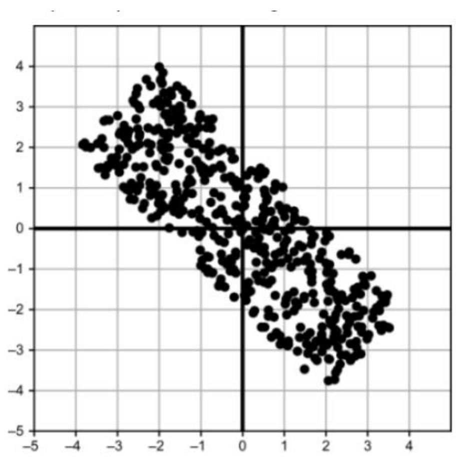


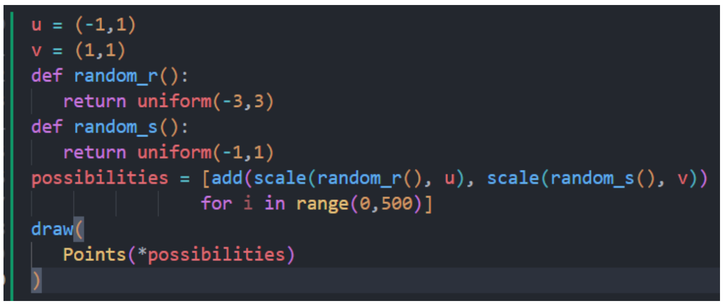
<u>**Exercise 2.17**: Write a Python function scale(s,v) that multiplies the input vector **v** by the input scalar *s*.</u>



```python
def scale(scalar,v):
    return (scalar * v[0], scalar * v[1])
```

<u>**Exercise 2.19–Mini Project**: Suppose *z* = (−1, 1) and **v** = (1, 1), and suppose *r* and *s* are real numbers. Specifically, let's assume −3 < *r* < 3 and −1 < *s* < 1. Where are the possible points on the plane where the vector *r* · **u** + *s* · **v** could end up?</u>



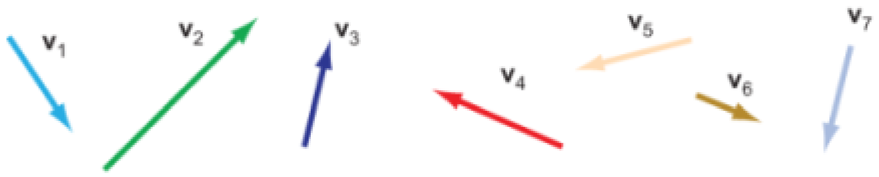


```
u = (-1,1)
v = (1,1)
def random_r():
    return uniform(-3,3)
def random_s():
    return uniform(-1,1)
possibilities = [add(scale(random_r(), u), scale(random_s(), v))
                for i in range(0,500)]
draw(
    Points(*possibilities)
)
```

Exercise 2.20: Show algebraically why a vector and its opposite have the same length.

$$\sqrt{(-a)^2 + (-b)^2} = \sqrt{(-a) \cdot (-a) + (-b) \cdot (-b)} = \sqrt{a^2 + b^2}$$

Exercise 2.21: Of the following seven vectors, represented as arrows, which two are a pair of opposite vectors?



V_3 **and** $\mathbf{V_7}$

Exercise 2.22: Suppose **u** is any 2D vector. What are the coordinates of **u** + −**u**?

(0, 0)

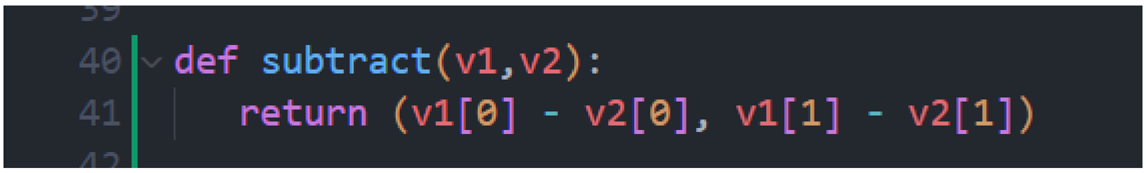
Exercise 2.23: For vectors u = (−2, 0), v = (1.5, 1.5), and w = (4, 1), what are the results of the vector subtractions v − w, u − v, and w − v?

v − **w** = (−2.5, 0.5)

u − **v** = (−3.5, −1.5)

w − **v** = (2.5, −0.5)

Exercise 2.24: Write a Python function subtract (v1, v2) that returns the result of v1 - v2, taking two 2D vectors as inputs and returning a 2D vector as an output.

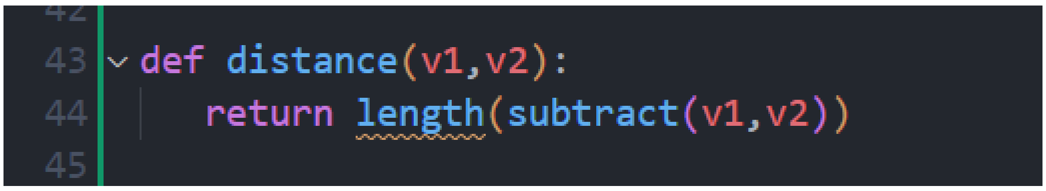


```
def subtract(v1,v2):
    return (v1[0] - v2[0], v1[1] - v2[1])
```

Exercise 2.25: Write a Python function distance (v1, v2) that returns the distance between two input vectors. (Note that the subtract function from the previous exercise already gives the *displacement*.)

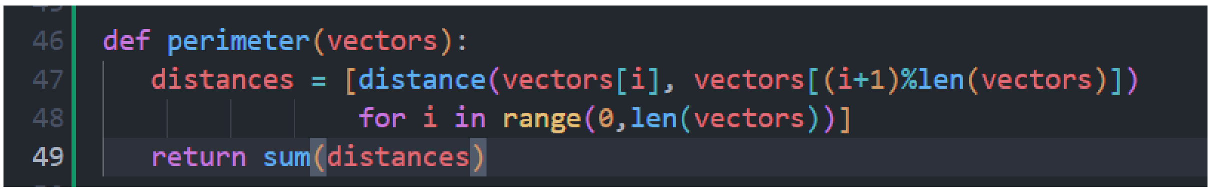
Write another Python function perimeter(vectors) that takes a list of vectors as an argument and returns the sum of distances from each vector to the next, including the distance from the last vector to the first. What is the perimeter of the dinosaur defined by dino_vectors?

Distance:



```python
def distance(v1,v2):
    return length(subtract(v1,v2))
```

Perimeter:

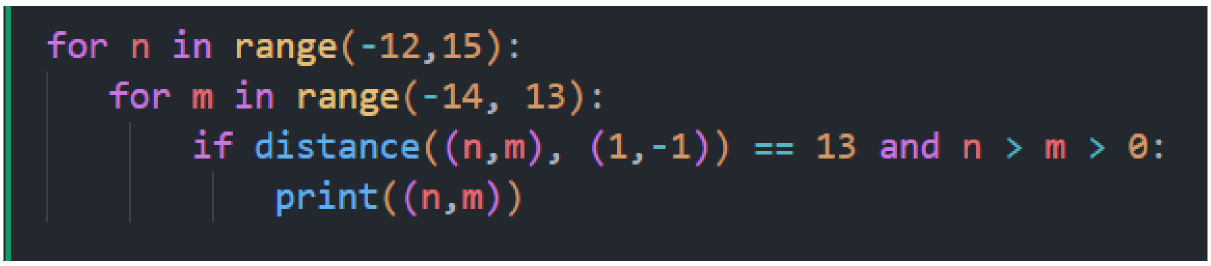


```python
def perimeter(vectors):
    distances = [distance(vectors[i], vectors[(i+1)%len(vectors)])
                 for i in range(0,len(vectors))]
    return sum(distances)
```

Perimeter of dinosaur:

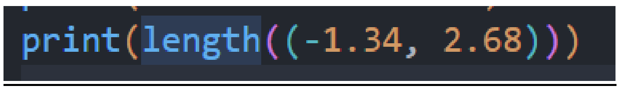
44.77115093694563

Exercise 2.26–Mini Project: Let **u** be the vector (1, −1). Suppose there is another vector **v** with positive integer coordinates (*n*, *m*) such that *n* > *m* and has a distance of 13 from **u**. What is the displacement from **u** to **v**?



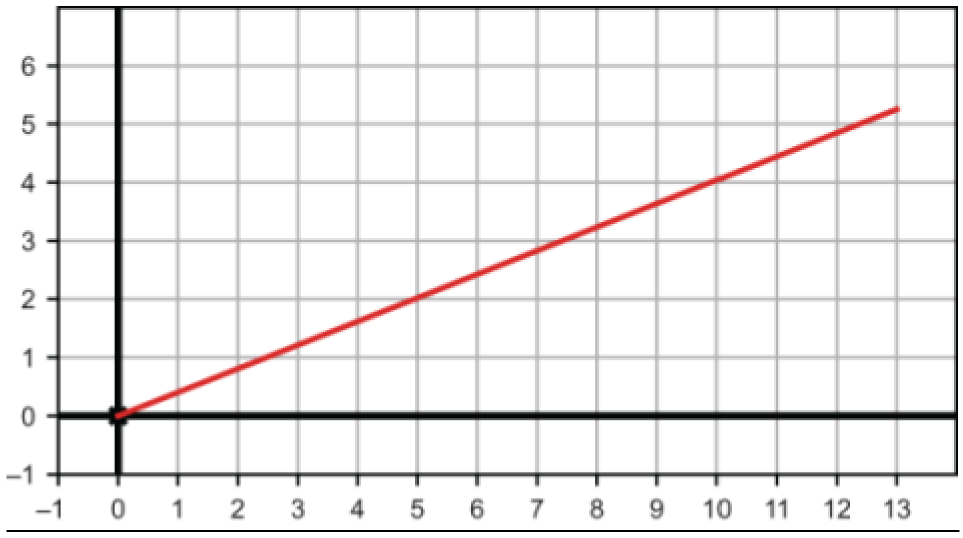
```python
for n in range(-12,15):
    for m in range(-14, 13):
        if distance((n,m), (1,-1)) == 13 and n > m > 0:
            print((n,m))
```

Exercise 2.27: Confirm that the vector given by Cartesian coordinates (−1.34, 2.68) has a length of approximately 3 as expected.



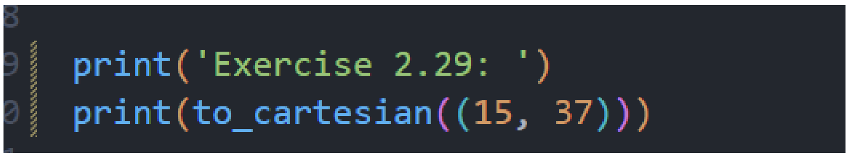
```python
print(length((-1.34, 2.68)))
```

Exercise 2.28: The figure shows a line that makes a 22° angle in the counterclockwise direction from the positive *x*-axis. Based on the following picture, what is the approximate value of tan(22°)?



4 / 10 = **0.4**

Exercise 2.29: What are the *x* and *y* components of a vector with length 15 pointing at a 37° angle?



```
print('Exercise 2.29: ')
print(to_cartesian((15, 37)))
```

(11.48121077918015, -9.653072000354992)

Exercise 2.30: Suppose I travel 8.5 units from the origin at an angle of 125°, measured counter clockwise from the positive *x*-axis. Given that sin (125°) = 0.819 and cos (125°) = −0.574, what are my final coordinates?

$x = r \cdot \cos(\vartheta) = 8.5 \cdot -0.574 = -4.879$

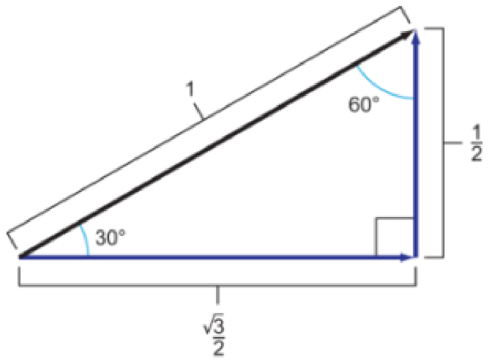
$y = r \cdot \sin(\vartheta) = 8.5 \cdot 0.819 = 6.962$

Exercise 2.31: What are the sine and cosine of 0°? Of 90°? Of 180°?

Input	Value
Sine 0°	0
Sine 90°	1
Sine 180°	0
Cosine 0°	1
Cosine 90°	0
Cosine 180°	-1

Exercise 2.32: The following diagram gives some exact measurements for a right triangle:

First, confirm that these lengths are valid for a right triangle because they satisfy the Pythagorean theorem. Then, calculate the values of sin (30°), cos(30°), and tan(30°) to three decimal places using the measurements in the diagram.



Sin (30) = 0.5/1 = 0.5

Cos (30) = 3^0.5/2 = 0.866 (approx.)

Tan (30) = 0.5 / (3^0.5/2) = 0.577 (approx.)

Exercise 2.33: Looking at the triangle from the previous exercise from a different perspective, use it to calculate the values of sin (60°), cos (60°), and tan (60°) to three decimal places.

Sin (60) = (3^0.5/2)/1 = 0.866 (approx.)

Cos (60) = 0.5/1 = 0.5

Tan (60) = (3^0.5/2)/0.5 = 1.732 (approx.)

Exercise 2.34: The cosine of 50° is 0.643. What is sin (50°) and what is tan (50°)?

Sin (50) = 0.766
Tan (50) = 1.192

Exercise 2.35: What is 116.57° in radians? Use Python to compute the tangent of this angle and confirm that it is close to −2 as we saw previously.

116.56 / 57.296 = **2.035**

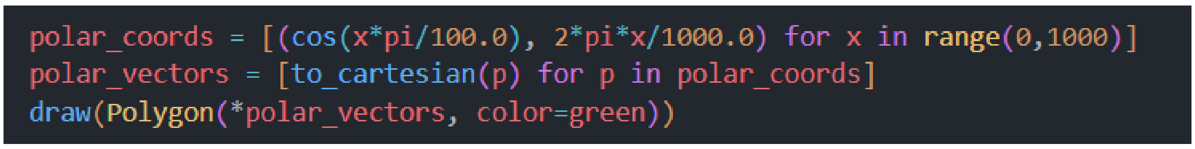
Exercise 2.36: Locate the angle 10π/6

1π/6 (1 12th of circle) -> 10/12 of circle -> 360 / 12 = 30 -> 10 * 30 = 300 degrees

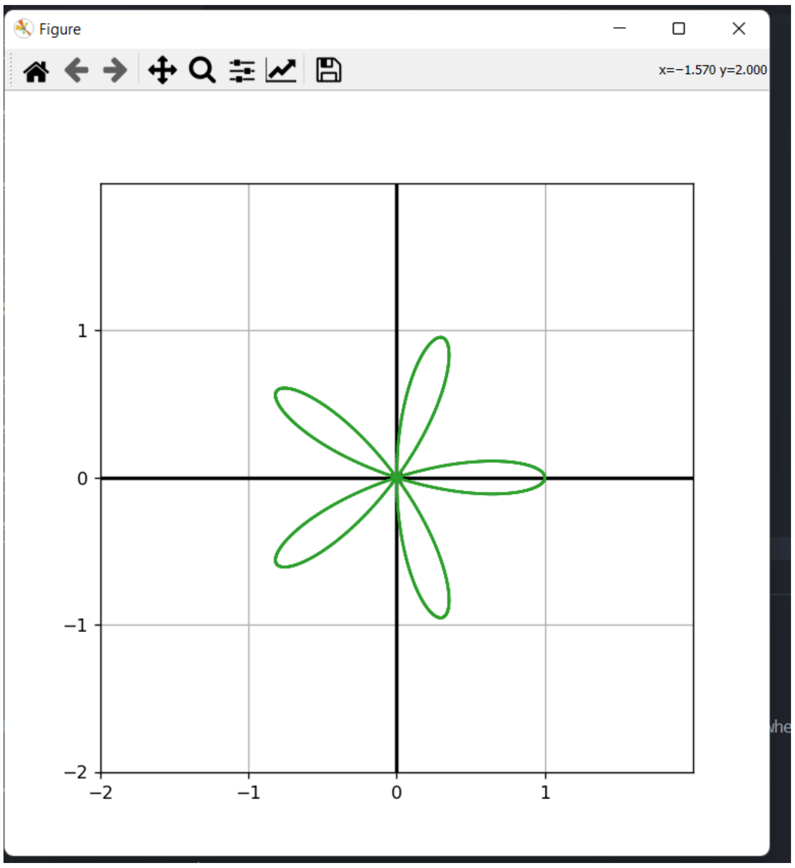
Exercise 2.37: The following list comprehension creates 1,000 points in polar coordinates:

```
[(cos(5*x*pi/500.0), 2*pi*x/1000.0) for x in range(0,1000)]
```

In Python code, convert these to Cartesian coordinates and connect them in a closed loop with line segments to draw a picture.



```
polar_coords = [(cos(x*pi/100.0), 2*pi*x/1000.0) for x in range(0,1000)]
polar_vectors = [to_cartesian(p) for p in polar_coords]
draw(Polygon(*polar_vectors, color=green))
```



Exercise 2.38: Find the angle to get to the point (−2, 3) by "guess-and-check."

Tan (angle) = -(3/2)

Tan (2.159rad)

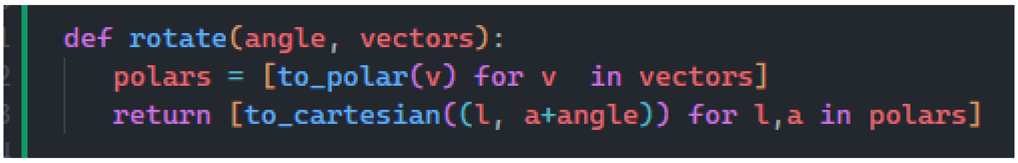
Exercise 2.39: Find another point in the plane with the same tangent as *ϑ*, namely −3/2. Use Python's implementation of the arctangent function, math. Tan, to find the value of this angle.

Arctan (-3/2) = **−0.982793723247329**

Exercise 2.40: What are the polar coordinates corresponding to the Cartesian coordinates (1, 1) and (1, −1)?

((0.5403, 0.8414), (0.5403, -0.8414))

Exercise 2.42: Create a rotate (angle, vectors) function that takes an array of input vectors in Cartesian coordinates and rotates those by the specified angle (counterclockwise or clockwise, according to whether the angle is positive or negative).

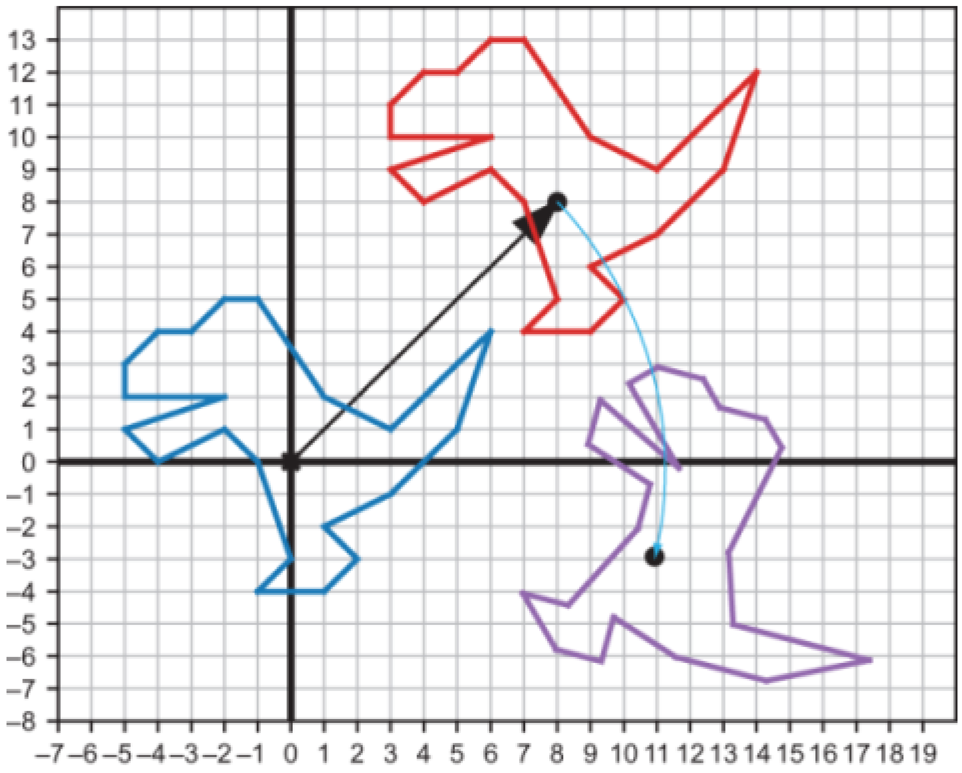


```python
def rotate(angle, vectors):
    polars = [to_polar(v) for v  in vectors]
    return [to_cartesian((l, a+angle)) for l,a in polars]
```

Exercise 2.43: Create a function regular polygon(n) that returns Cartesian coordinates for the vertices of a regular *n* -sided polygon (that is, having all angles and side lengths equal).

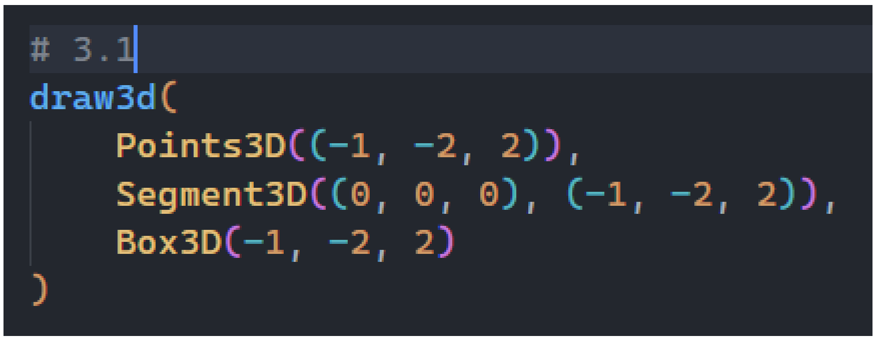
```
def regular_polygon(n):  # 2.43
    return [to_cartesian((1, 2*pi*k/n)) for k in range(0, n)]
```

Exercise 2.44: What is the result of first translating the dinosaur by the vector (8, 8) and then rotating it by 5π/3?

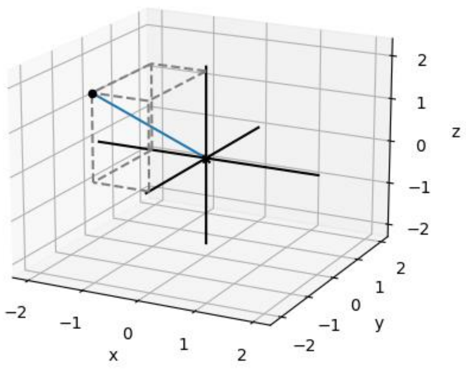


3D Vectors

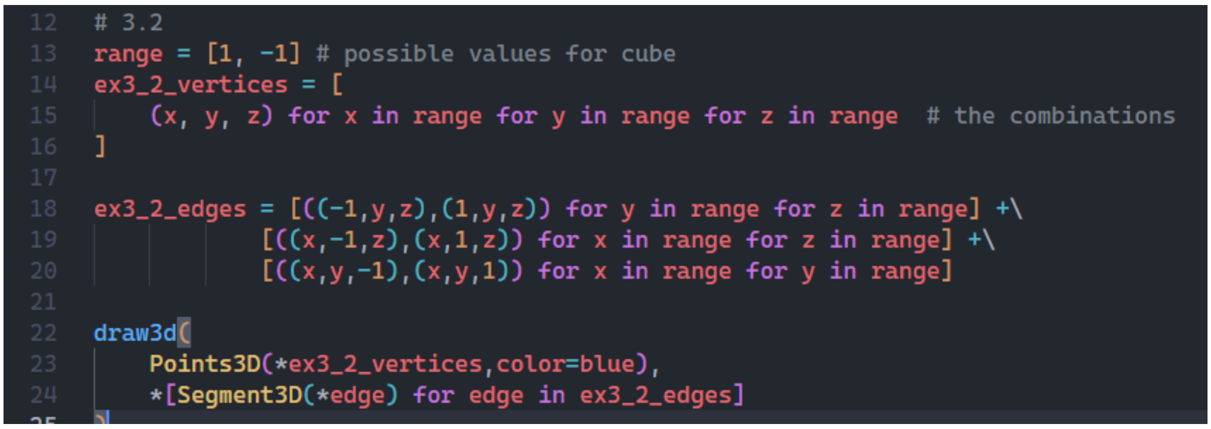
Exercise 3.1: Draw the 3D arrow and point representing the coordinates (−1, −2, 2) as well as the dashed box that makes the arrow look 3D.



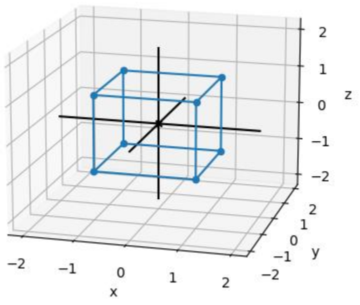
```
# 3.1
draw3d(
    Points3D((-1, -2, 2)),
    Segment3D((0, 0, 0), (-1, -2, 2)),
    Box3D(-1, -2, 2)
)
```



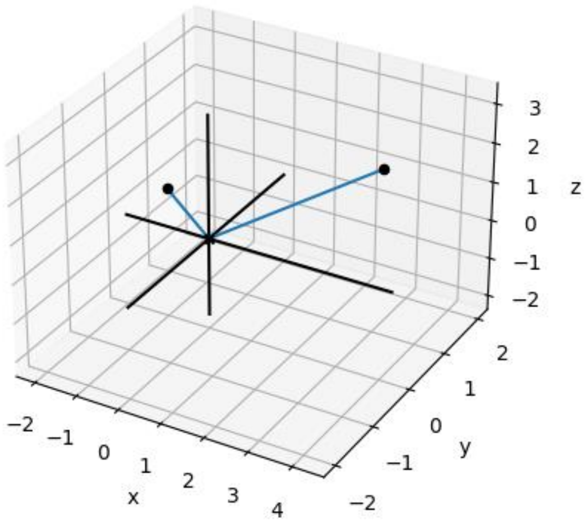
Exercise 3.2-Mini Project: There are exactly eight 3D vectors whose coordinates are all either +1 or −1. For instance, (1, −1, 1) is one of these. Plot all these eight vectors as points. Then figure out how to connect them with line segments using Segment3D objects to form the outline of a cube.



```
# 3.2
range = [1, -1] # possible values for cube
ex3_2_vertices = [
    (x, y, z) for x in range for y in range for z in range  # the combinations
]

ex3_2_edges = [((-1,y,z),(1,y,z)) for y in range for z in range] +\
              [((x,-1,z),(x,1,z)) for x in range for z in range] +\
              [((x,y,-1),(x,y,1)) for x in range for y in range]

draw3d(
    Points3D(*ex3_2_vertices,color=blue),
    *[Segment3D(*edge) for edge in ex3_2_edges]
)
```



Draw (4, 0, 3) and (−1, 0, 1) as Arrow3D objects, such that they are placed tip-to-tail in both orders in 3D. What is their vector sum?



```
(3, 0, 4)
```

Suppose we set vectors1= [(1,2,3,4,5), (6,7,8,9,10)] and vectors2= [(1,2), (3,4), (5,6)]. Without evaluating in Python, what are the lengths of zip(*vectors1) and zip(*vectors2)?

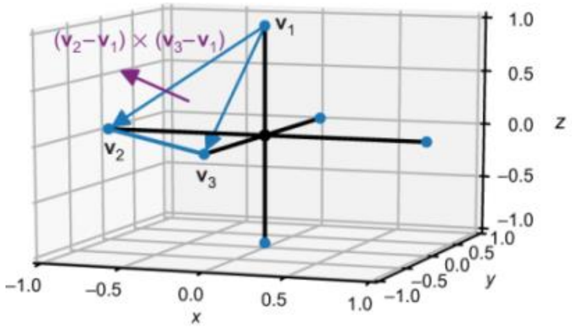
Length vectors1 = 5

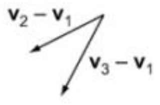
Length vectors2 = 2

Defining 3D Objects with Vectors

1. Define vertices (coordinates)
2. Draw the direct vectors
3. Generate all triangular faces:
4. Shade triangles

This is the core of game development.

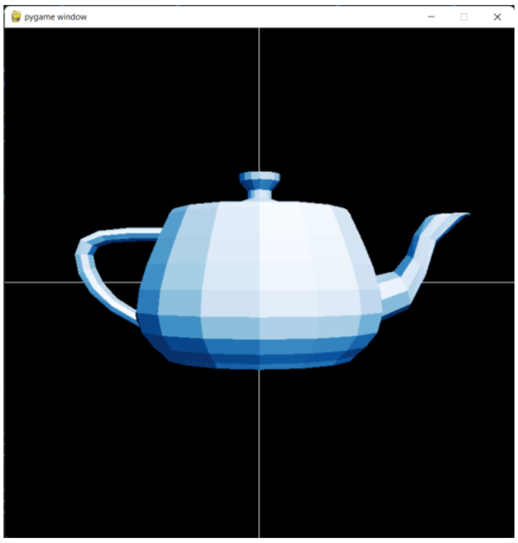




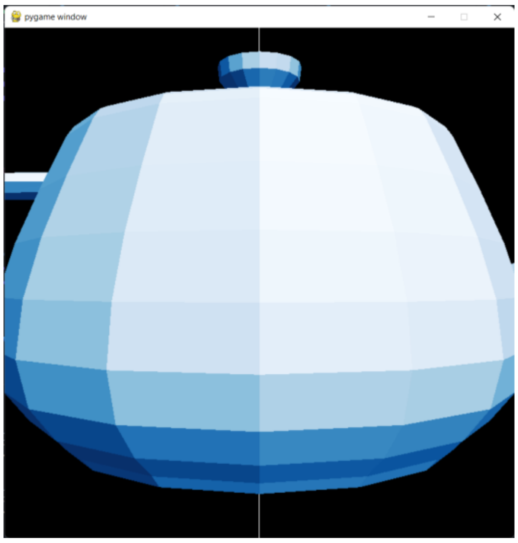
3D Graphics

The goal of this chapter is to be able to create three dimensional objects that change over time (with Python), while understanding the key concepts behind it. These topics include mathematical functions, transformations to vector graphics, identifying linear transformations as well as computing the effects of linear transformations on vectors and 3D models. I will be using **OpenGL** a more powerful drawing tool (industry standard library for high-performance graphics). Usually OpenGL is done through C/C++, but we can use a wrapper (PyOpenGL) to able to use Python. Additionally, PyGame is a useful library for any video game development, in our case, helping with the rendering of successive images into an animation.

Initially we can create a simple three-dimensional teapot made up of a series of vectors:

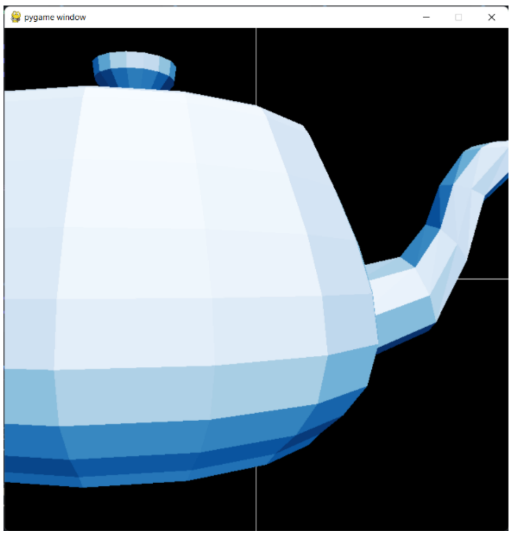


We can modify all the vectors to blow up or shrink the teapot:

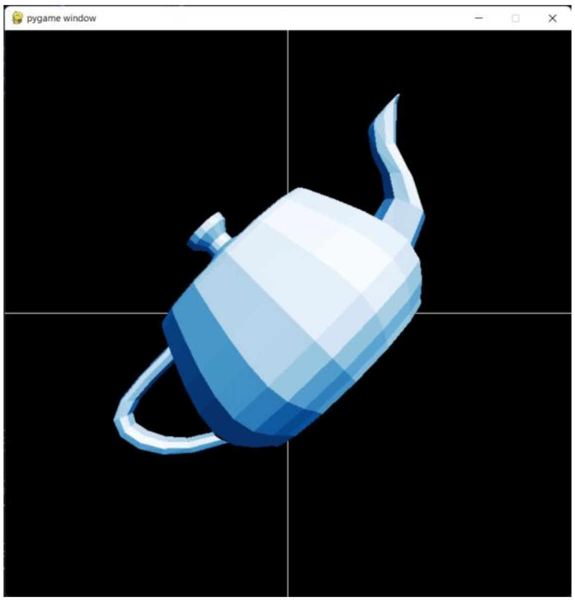


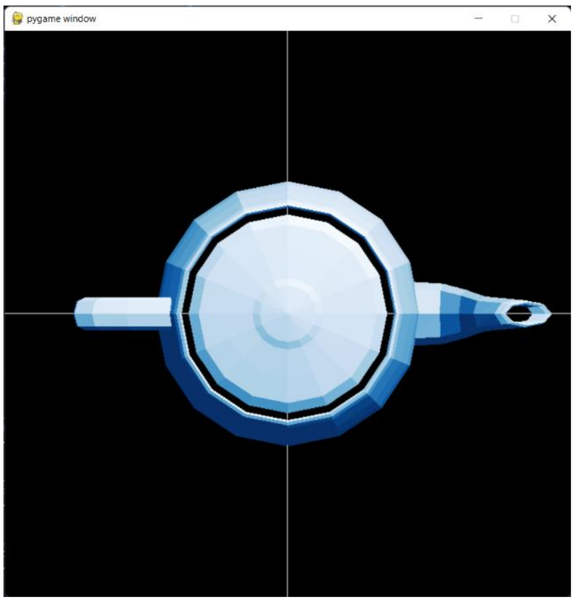
In this case we multiplied all vectors by a scale of 2.

We can also transform the image through displacing the vectors on the x axis by -1.

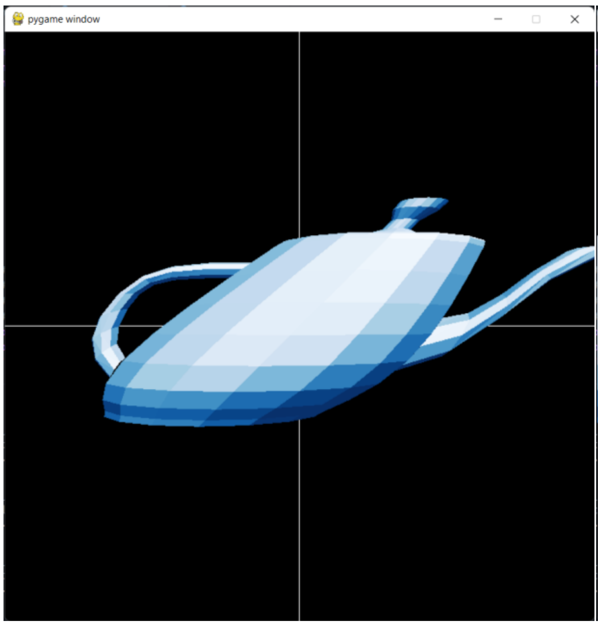


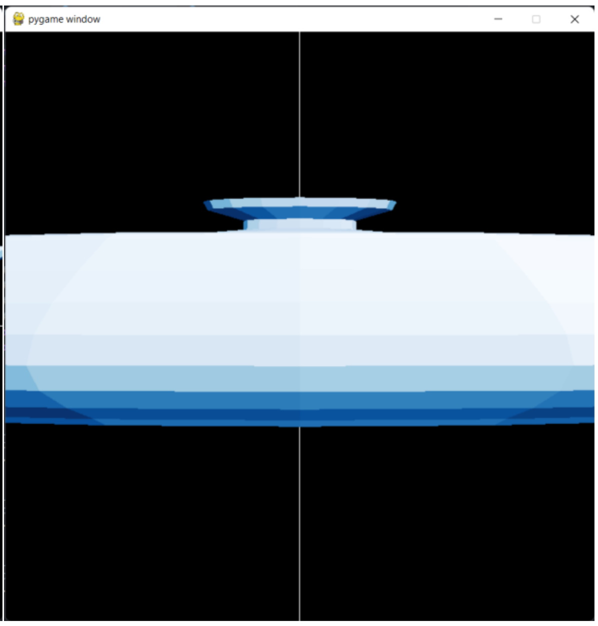
We can also rotate the teapot.



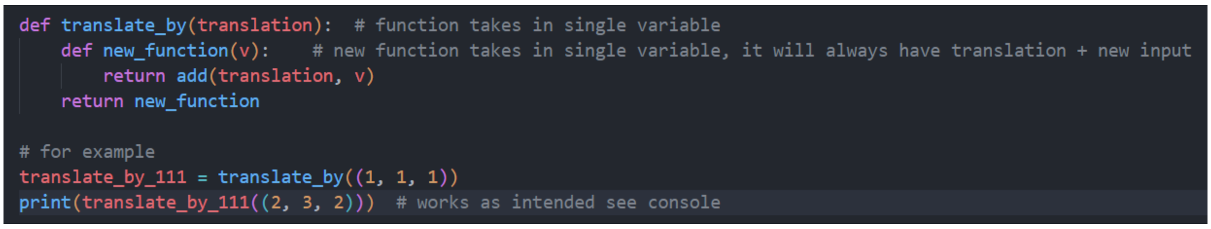


There are many other transformations we can make. Here are some examples:

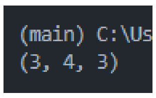




Exercise 4.1: Implement a translate_by function taking a translation vector as input and returning a translation function as output.

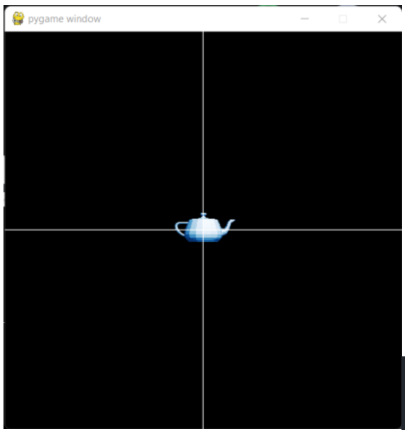


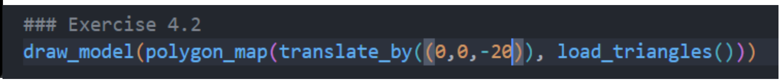
```python
def translate_by(translation):  # function takes in single variable
    def new_function(v):    # new function takes in single variable, it will always have translation + new input
        return add(translation, v)
    return new_function

# for example
translate_by_111 = translate_by((1, 1, 1))
print(translate_by_111((2, 3, 2)))  # works as intended see console
```



```
(main) C:\Us
(3, 4, 3)
```

Exercise 4.2: Render the teapot translated by 20 units in the negative *z* direction. What does the resulting image look like?



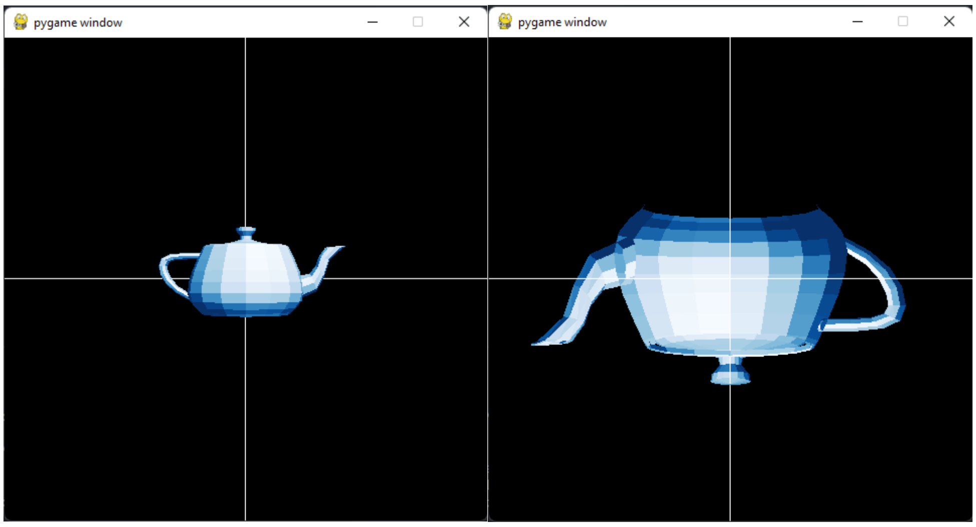


```
### Exercise 4.2
draw_model(polygon_map(translate_by((0,0,-20)), load_triangles()))
```

Draw_model initiates the drawing, polygon_map allows for multiple vectors to be given simultaneously while the translate_by function from the previous exercise will translate every vector. Load_triangles load in the original teapot vectors.

Exercise 4.3-Mini Project: What happens to the teapot when you scale every vector by a scalar between 0 and 1? What happens when you scale it by a factor of −1?

Any value below 1 will shrink the teapot while -1 will rotate the teapot by 180 degrees:



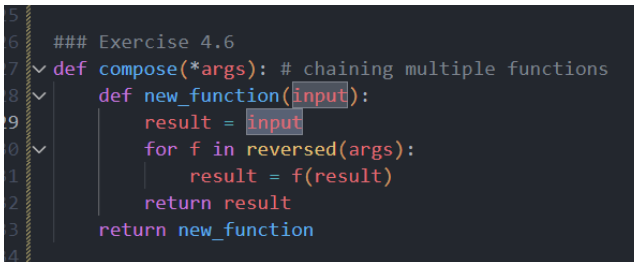
Exercise 4.4: First apply translate1left to the teapot and then apply scale2. How is the result different from the opposite order of composition? Why?

Translate followed by scale will be further from the center than scale followed by translation, since the scale will build upon the already translated vectors. Scaling for example a (2, 0) vector by 2 will be (4, 0) while a (8, 0) vector will be (16, 0) a far greater translation.

Exercise 4.5: What does the effect of the transformation *compose (scale_by (0.4), scale_by (1.5))*?

0.4 * 1.5 = 0.6, thus the overall scale will be 0.6, 60% of the original object.

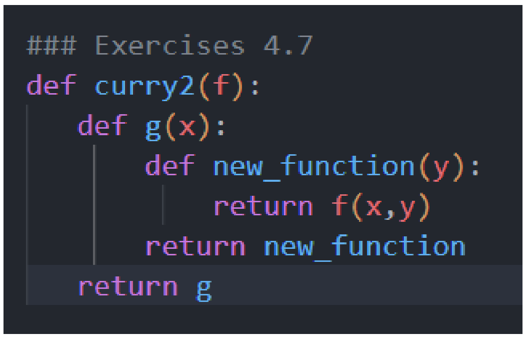
Exercise 4.6: Modify the compose (f, g) function to compose(*args), which takes several functions as arguments and returns a new function that is their composition.



```python
### Exercise 4.6
def compose(*args): # chaining multiple functions
    def new_function(input):
        result = input
        for f in reversed(args):
            result = f(result)
        return result
    return new_function
```

This will append every function return to each other when calling the compose parent function.

Exercise 4.7: Write a curry2(f) function that takes a Python function f (x, y) with two arguments and returns a curried version. For instance, once you write g = curry2(f), the two expressions f (x, y) and g(x)(y) should return the same result.



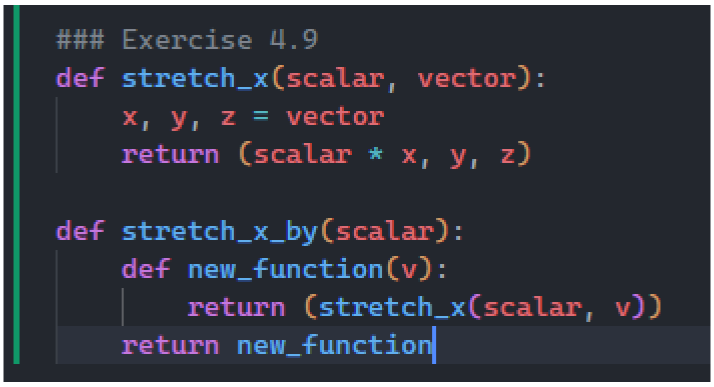
```python
### Exercises 4.7
def curry2(f):
    def g(x):
        def new_function(y):
            return f(x,y)
        return new_function
    return g
```

https://stackoverflow.com/questions/36314/what-is-currying

Exercise 4.8: Without running it, what is the result of applying the transformation compose(rotate_z_by(pi/2), rotate_x_by(pi/2))? What if you switch the order of the composition?

A clockwise rotation by 90 degrees about the y-axis (since we are rotating the two other axis). Switching the order of the composition (first x, then z, will cause the rotation to be anti-clockwise).

Exercise 4.9: Write a function stretch_x (scalar, vector) that scales the target vector by the given factor but only in the *x* direction. Also write a curried version stretch_x_by so that stretch_x_by(scalar)(vector) returns the same result.



```python
### Exercise 4.9
def stretch_x(scalar, vector):
    x, y, z = vector
    return (scalar * x, y, z)

def stretch_x_by(scalar):
    def new_function(v):
        return (stretch_x(scalar, v))
    return new_function
```

Exercise 4.10: Considering *S* again, the vector transformation that squares all coordinates, show algebraically that *S*(*s***v**) = *sS*(**v**) does not hold for all choices of scalars *s* and 2D vectors **v**.

S = vector transformation ($x^{2,} y^2$)
s = a scalar (e.g., 3)
v = a vector (e.g., 1, 2, 1)

S(sv) = S(3, 6, 3) = (9, 36, 9)
sS(v) = s(1, 4, 1) = (3, 12, 3)
Therefore S(sv) != sS(v)

Exercise 4.11: Suppose *T* is a vector transformation and *T*(**o**) ≠ o, where **o** represents the vector with all coordinates equal to zero. Why is *T* not linear according to the definition?

o = (0, 0, 0)
v = (1, 1, 1)
T = unknown vector transformation
T(o + v) = T(o) + T(v)
Since we know o is (0, 0, 0) the statement can be:
T(o + v) = T(v) = T(o) + T(v)
0 = T(o) meaning the statement *T*(**o**) ≠ o is wrong.
?

Exercise 4.12: The *identity transformation* is the vector transformation that returns the same vector it is passed. It is denoted with a capital *I*, so we could write its definition as *I*(**v**) = **v** for all vectors **v**. Explain why *I* is a linear transformation.

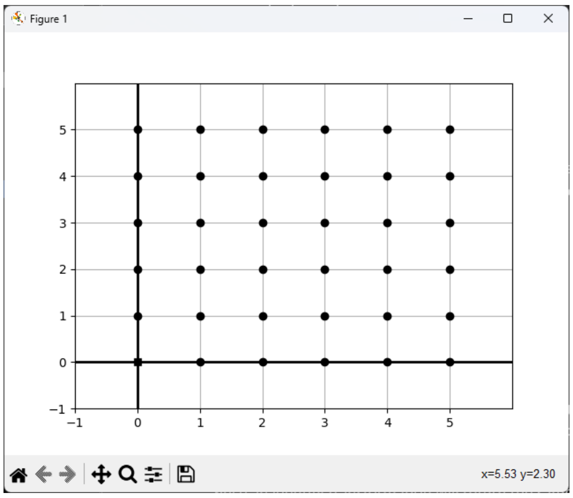
Since an identify transformation will return the same vector that is passed, it will definitely preserve vector sums/scalar multiples, thus by definition is a linear transformation.

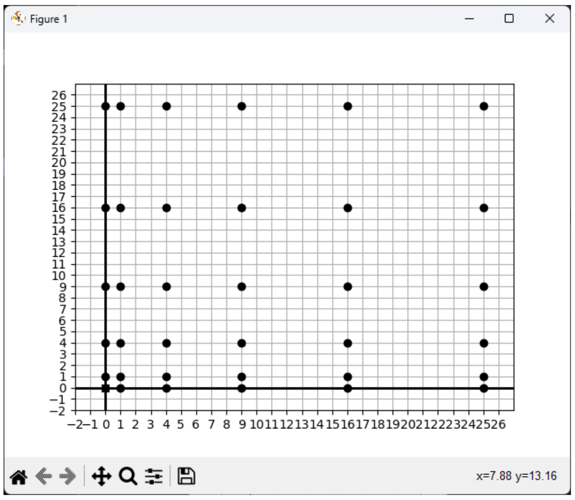
Exercise 4.13: What is the midpoint between (5, 3) and (−2, 1)?
Using linear transformation 0.5.
u = (5, 3)
v = (-2, 1)
u + v = (3, 4)

(1.5, 2) = 0.5(5, 3) + 0.5(-2, 1)
(1.5, 2) = (2.5, 1.5) + (-1, 0.5)
(1.5, 2) = (1.5, 2)
Thus, the midpoint is (1.5, 2)

Exercise 4.14: Consider again the non-linear transformation *S*(**v**) sending **v** = (*x*, *y*) to (x^2, y^2). Plot all 36 vectors **v** with integer coordinates 0 to 5 as points using the drawing code from chapter 2 and then plot *S*(**v**) for each of them. What happens geometrically to vectors under the action of *S*?

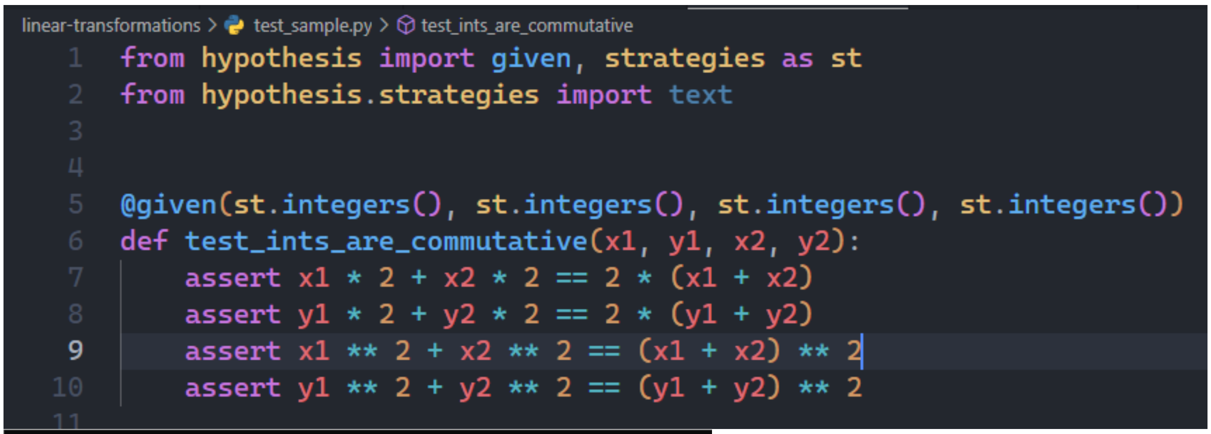
The uniform distance between each point is lost. No linear transformation in this case.



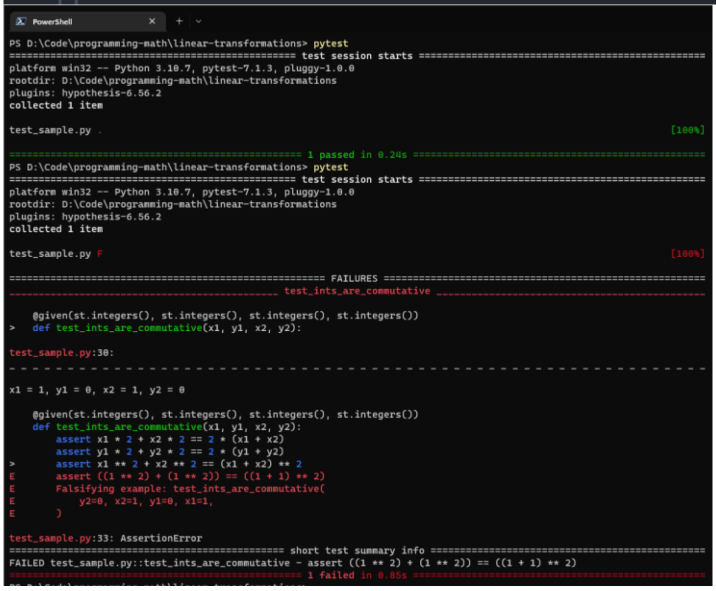


Exercise 4.15-Mini Project: Using your library of choice, implement property-based tests that check if a vector transformation is linear.

Specifically, given a vector transformation *T* implemented as a Python function, generate many pairs of random vectors and assert for all of those that their sum is preserved by *T*. Then, do the same thing for pairs of a scalar and a vector, and ensure that *T* preserves scalar multiples. You should find that linear transformations like rotate_x_by(pi/2) pass the test, but non-linear transformations like the coordinate-squaring transformation do not pass.


linear-transformations > test_sample.py > test_ints_are_commutative

```python
from hypothesis import given, strategies as st
from hypothesis.strategies import text


@given(st.integers(), st.integers(), st.integers(), st.integers())
def test_ints_are_commutative(x1, y1, x2, y2):
    assert x1 * 2 + x2 * 2 == 2 * (x1 + x2)
    assert y1 * 2 + y2 * 2 == 2 * (y1 + y2)
    assert x1 ** 2 + x2 ** 2 == (x1 + x2) ** 2
    assert y1 ** 2 + y2 ** 2 == (y1 + y2) ** 2
```

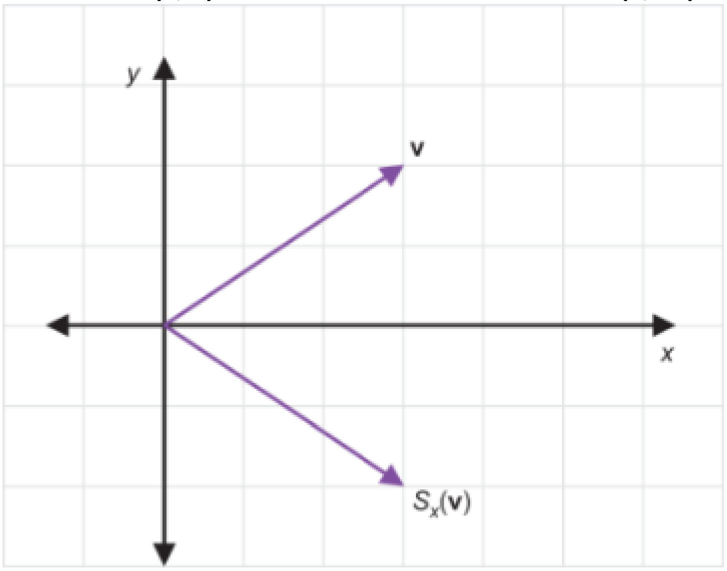


```
PS D:\Code\programming-math\linear-transformations> pytest
================================================ test session starts ================================================
platform win32 -- Python 3.10.7, pytest-7.1.3, pluggy-1.0.0
rootdir: D:\Code\programming-math\linear-transformations
plugins: hypothesis-6.56.2
collected 1 item

test_sample.py .                                                                                              [100%]

================================================= 1 passed in 0.24s =================================================
PS D:\Code\programming-math\linear-transformations> pytest
================================================ test session starts ================================================
platform win32 -- Python 3.10.7, pytest-7.1.3, pluggy-1.0.0
rootdir: D:\Code\programming-math\linear-transformations
plugins: hypothesis-6.56.2
collected 1 item

test_sample.py F                                                                                              [100%]

===================================================== FAILURES ======================================================
____________________________________________ test_ints_are_commutative _____________________________________________

    @given(st.integers(), st.integers(), st.integers(), st.integers())
>   def test_ints_are_commutative(x1, y1, x2, y2):

test_sample.py:30:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

x1 = 1, y1 = 0, x2 = 1, y2 = 0

    @given(st.integers(), st.integers(), st.integers(), st.integers())
    def test_ints_are_commutative(x1, y1, x2, y2):
        assert x1 * 2 + x2 * 2 == 2 * (x1 + x2)
        assert y1 * 2 + y2 * 2 == 2 * (y1 + y2)
>       assert x1 ** 2 + x2 ** 2 == (x1 + x2) ** 2
E       assert ((1 ** 2) + (1 ** 2)) == ((1 + 1) ** 2)
E       Falsifying example: test_ints_are_commutative(
E           y2=0, x2=1, y1=0, x1=1,
E       )

test_sample.py:33: AssertionError
============================================== short test summary info ==============================================
FAILED test_sample.py::test_ints_are_commutative - assert ((1 ** 2) + (1 ** 2)) == ((1 + 1) ** 2)
================================================= 1 failed in 0.85s =================================================
```

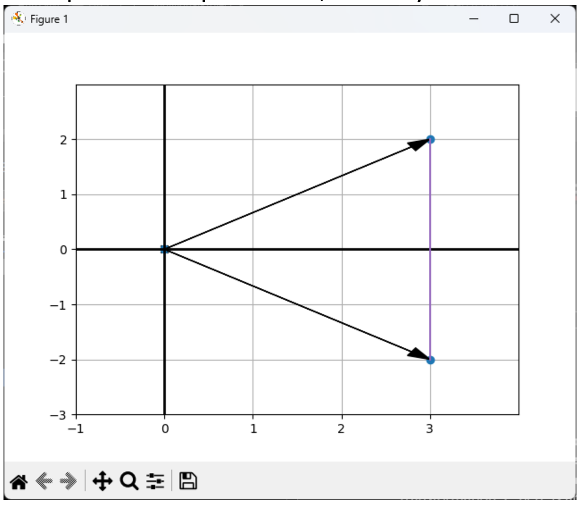
The scalar linear transformations work while the non-linear squared transformations fail.

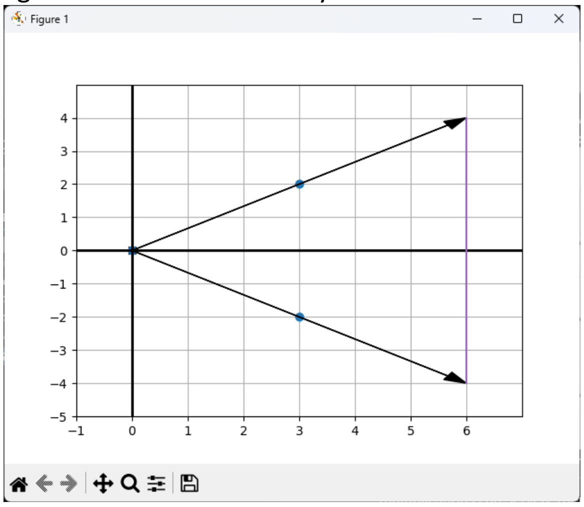
Exercise 4.16: One 2D vector transformation is *reflection* across the *x* -axis. This transformation takes a vector and returns another one, which is the mirror image with respect to the *x*-axis. Its *x*-coordinate should be unchanged, and its *y*-coordinate should change its sign. Denoting this transformation S_x, here is an image of a vector **v** = (3, 2) and the transformed vector S_x(**v**).

A vector v = (3, 2) and its reflection over the *x*-axis (3, −2)



Draw two vectors and their sum, as well as the reflection of these three vectors to demonstrate that this transformation preserves vector addition. Draw another diagram to show similarly that scalar multiplication is preserved, thereby demonstrating both criteria for linearity.





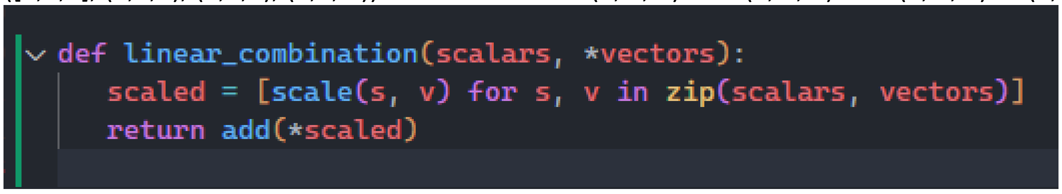
Exercise 4.17-Mini Project: Suppose *S* and *T* are both linear transformations. Explain why the composition of *S* and *T* is also linear.

Due to the rules of linearity, any addition or scalar multiple will always maintain linearity.

Exercise 4.18: Let *T* be the linear transformation done by the Python function rotate_x_by(pi/2), what are *T*($\mathbf{e}_1$), *T*($\mathbf{e}_2$), and *T*($\mathbf{e}_3$)?

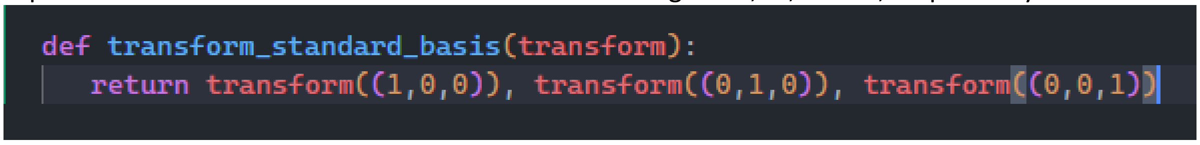
Let vector = (1, 1, 1)
Let e_1 = (1, 0, 0)
Transformation of 90 degrees is (0, 1, 0)
So e_2 = (0, 1, 0)
And e_3 = (0, 0, 1)

Exercise 4.19: Write a linear_combination (scalars, *vectors) that takes a list of scalars and the same number of vectors and returns a single vector. For example, linear_combination ([1,2,3], (1,0,0), (0,1,0), (0,0, 1)) should return 1 · (1, 0, 0) + 2 · (0, 1, 0) + 3 · (0, 0, 1) or (1, 2, 3).



```python
def linear_combination(scalars, *vectors):
    scaled = [scale(s, v) for s, v in zip(scalars, vectors)]
    return add(*scaled)
```

Exercise 4.20: Write a function transform_standard_basis(transform) that takes a 3D vector transformation as an input and outputs the effect it has the standard basis. It should output a tuple of 3 vectors that are the results of transform acting on $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, respectively.



```python
def transform_standard_basis(transform):
    return transform((1,0,0)), transform((0,1,0)), transform((0,0,1))
```

Exercise 4.21: Suppose *B* is a linear transformation, with $B(\mathbf{e}_1) = (0, 0, 1)$, $B(\mathbf{e}_2) = (2, 1, 0)$, $B(\mathbf{e}_3) = (-1, 0, -1)$, and $\mathbf{v} = (-1, 1, 2)$. What is $B(\mathbf{v})$?

B(v) = B(-e$_1$, e$_2$, 2e$_3$)
= -(0, 0, 1) + (2, 1, 0) + 2(-1, 0, -1)
= (0, 1, -3)

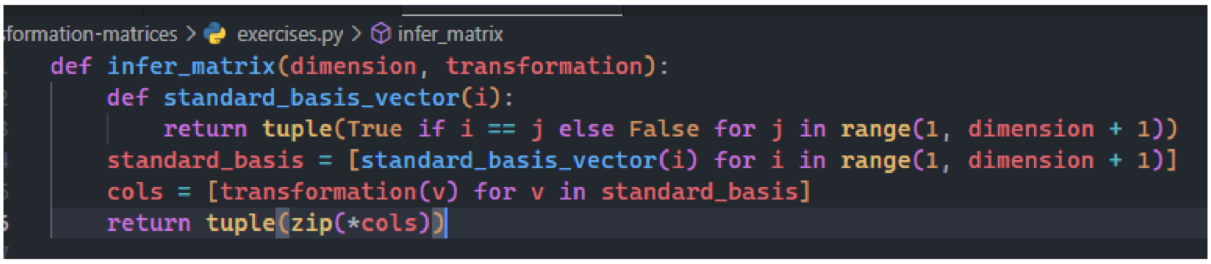
Exercise 4.22: Suppose *A* and *B* are both linear transformations with $A(\mathbf{e}_1) = (1, 1, 1)$, $A(\mathbf{e}_2) = (1, 0, -1)$, and $A(\mathbf{e}_3) = (0, 1, 1)$, and $B(\mathbf{e}_1) = (0, 0, 1)$, $B(\mathbf{e}_2) = (2, 1, 0)$, and $B(\mathbf{e}_3) = (-1, 0, -1)$. What is $A(B(\mathbf{e}_1))$, $A(B(\mathbf{e}_2))$, and $A(B(\mathbf{e}_3))$?

$A(\mathbf{e}_1) = (1, 1, 1)$
$A(\mathbf{e}_2) = (1, 0, -1)$
$A(\mathbf{e}_3) = (0, 1, 1)$
$B(\mathbf{e}_1) = (0, 0, 1)$
$B(\mathbf{e}_2) = (2, 1, 0)$
$B(\mathbf{e}_3) = (-1, 0, -1)$

$A(B(\mathbf{e}_1)) = A(0, 0, 1)$

Matrices

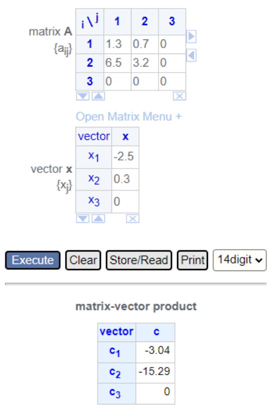
Exercise 5.1: Write *a* function infer_matrix(n, transformation) that takes a dimension (like 2 or 3) and a function that is a vector transformation assumed to be linear.



```python
def infer_matrix(dimension, transformation):
    def standard_basis_vector(i):
        return tuple(True if i == j else False for j in range(1, dimension + 1))
    standard_basis = [standard_basis_vector(i) for i in range(1, dimension + 1)]
    cols = [transformation(v) for v in standard_basis]
    return tuple(zip(*cols))
```

Exercise 5.2: What is the result of the following product of a 2-by−2 matrix with a 2D vector?

$$\begin{pmatrix}1.3 & 0.7\\ 6.5 & 3.2\end{pmatrix}\begin{pmatrix}-2.5\\ 0.3\end{pmatrix}$$


matrix A {aij}

i \ j	1	2	3
1	1.3	0.7	0
2	6.5	3.2	0
3	0	0	0

Open Matrix Menu +

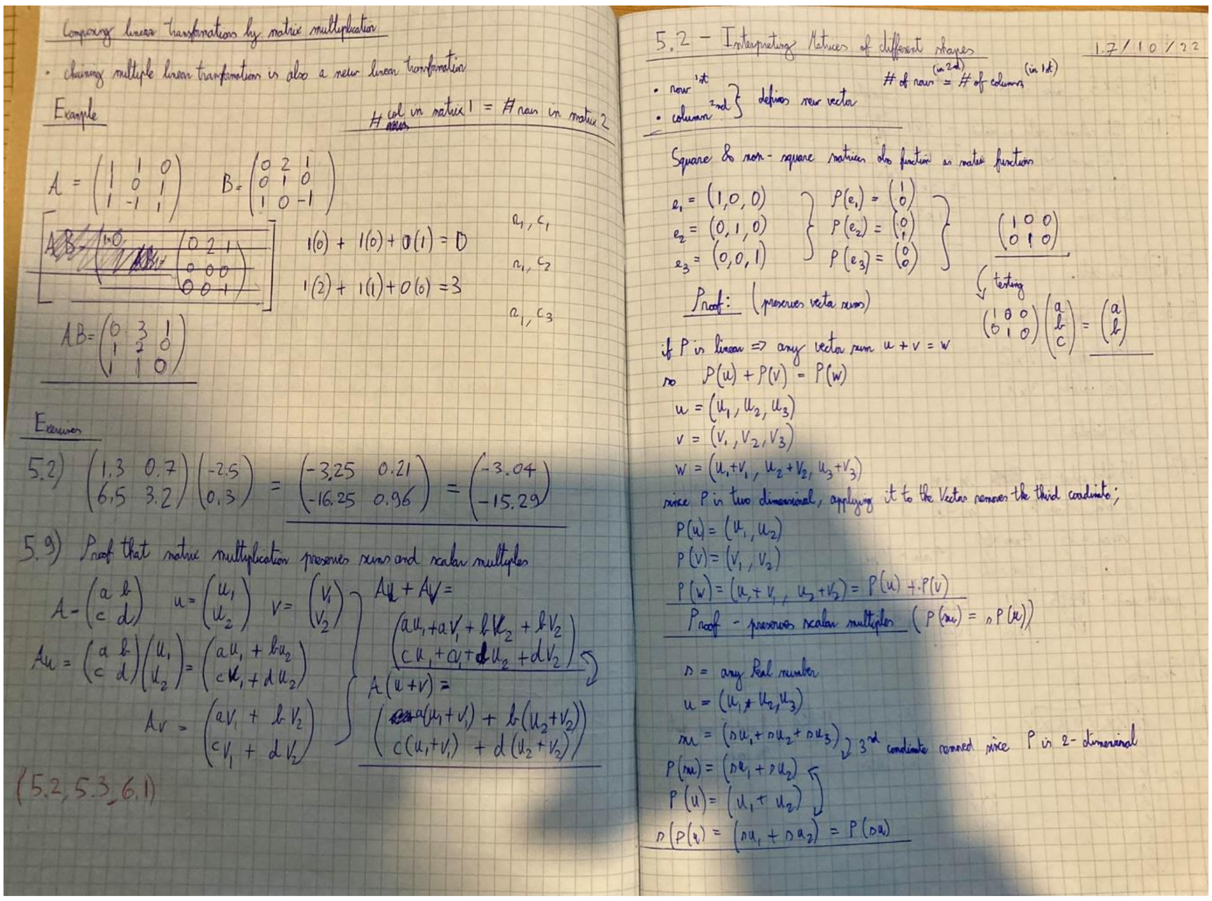
vector x {xj}

vector	x
x_1	-2.5
x_2	0.3
x_3	0

matrix-vector product

vector	c
c_1	-3.04
c_2	-15.29
c_3	0

Exercises 5.2, 5.9


Composing linear transformations by matrix multiplication

- chaining multiple linear transformation is also a new linear transformation

Example

~~rows~~ col in matrix 1 = # rows in matrix 2

$A = \begin{pmatrix}1 & 1 & 0\\ 1 & 0 & 1\\ 1 & -1 & 1\end{pmatrix}$ $B = \begin{pmatrix}0 & 2 & 1\\ 0 & 1 & 0\\ 1 & 0 & -1\end{pmatrix}$

~~$AB = \begin{pmatrix}0 & 2 & 1\\ 0 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$~~

$1(0) + 1(0) + 0(1) = 0$ r_1, c_1

$1(2) + 1(1) + 0(0) = 3$ r_1, c_2

r_1, c_3

$AB = \begin{pmatrix}0 & 3 & 1\\ 1 & 2 & 0\\ 1 & 1 & 0\end{pmatrix}$

Exercises

5.2) $\begin{pmatrix}1.3 & 0.7\\ 6.5 & 3.2\end{pmatrix}\begin{pmatrix}-2.5\\ 0.3\end{pmatrix} = \begin{pmatrix}-3.25 & 0.21\\ -16.25 & 0.96\end{pmatrix} = \begin{pmatrix}-3.04\\ -15.29\end{pmatrix}$

5.9) Proof that matrix multiplication preserves sums and scalar multiples

$A = \begin{pmatrix}a & b\\ c & d\end{pmatrix}$ $u = \begin{pmatrix}u_1\\ u_2\end{pmatrix}$ $v = \begin{pmatrix}v_1\\ v_2\end{pmatrix}$

$Au = \begin{pmatrix}a & b\\ c & d\end{pmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix} = \begin{pmatrix}au_1 + bu_2\\ cu_1 + du_2\end{pmatrix}$

$Av = \begin{pmatrix}av_1 + bv_2\\ cv_1 + dv_2\end{pmatrix}$

$Au + Av = \begin{pmatrix}au_1 + av_1 + bu_2 + bv_2\\ cu_1 + cv_1 + du_2 + dv_2\end{pmatrix}$

$A(u+v) = \begin{pmatrix}a(u_1+v_1) + b(u_2+v_2)\\ c(u_1+v_1) + d(u_2+v_2)\end{pmatrix}$

(5.2, 5.3, 6.1)

5.2 - Interpreting Matrices of different shapes 17/10/22

- row 1st
- column 2nd } defines new vector

# of rows (in 2nd) = # of columns (in 1st)

Square & non-square matrices do function as matrix functions

$e_1 = (1,0,0)$, $e_2 = (0,1,0)$, $e_3 = (0,0,1)$ } $P(e_1) = \begin{pmatrix}1\\ 0\end{pmatrix}$, $P(e_2) = \begin{pmatrix}0\\ 1\end{pmatrix}$, $P(e_3) = \begin{pmatrix}0\\ 0\end{pmatrix}$ } $\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{pmatrix}$

testing

$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{pmatrix}\begin{pmatrix}a\\ b\\ c\end{pmatrix} = \begin{pmatrix}a\\ b\end{pmatrix}$

Proof: (preserves vector sums)

if P is linear ⇒ any vector sum $u + v = w$

so $P(u) + P(v) = P(w)$

$u = (u_1, u_2, u_3)$

$v = (v_1, v_2, v_3)$

$w = (u_1+v_1, u_2+v_2, u_3+v_3)$

since P is two dimensional, applying it to the Vector removes the third coordinate;

$P(u) = (u_1, u_2)$

$P(v) = (v_1, v_2)$

$P(w) = (u_1+v_1, u_2+v_2) = P(u) + P(v)$

Proof - preserves scalar multiples ($P(su) = sP(u)$)

s = any Real number

$u = (u_1, u_2, u_3)$

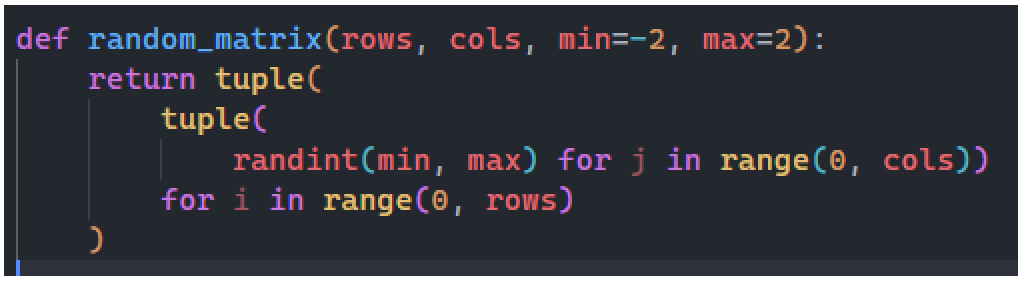
$su = (su_1 + su_2 + su_3)$ } 3rd coordinate removed since P is 2-dimensional

$P(su) = (su_1 + su_2)$

$P(u) = (u_1 + u_2)$

$s(P(u)) = (su_1 + su_2) = P(su)$

Exercise 5.3-Mini Project: Write a random_matrix function that generates matrices of a specified size with random whole number entries. Use the function to generate five pairs of 3-by–3 matrices. Multiply each of the pairs together by hand (for practice) and then check your work with the matrix_multiply function.



```python
def random_matrix(rows, cols, min=-2, max=2):
    return tuple(
        tuple(
            randint(min, max) for j in range(0, cols))
        for i in range(0, rows)
    )
```

Exercise 5.4: For each of your pairs of matrices from the previous exercise, multiply them in the opposite order. Do you get the same result?

I will get different results.

What are the matrices representing the identity transformation in 2D and 3D, respectively?

For 2d:

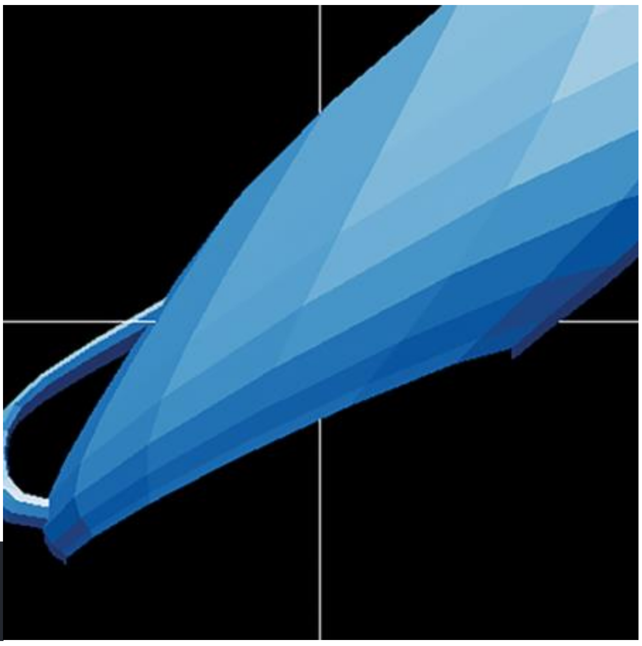
1	0
0	1

For 3d:

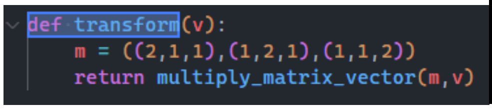
1	0	0
0	1	0
0	0	1

Multiplying by these matrices will return the same vector.

Exercise 5.6: Apply the matrix ((2,1,1), (1,2,1), (1,1,2)) to all the vectors defining the teapot. What happens to the teapot and why?

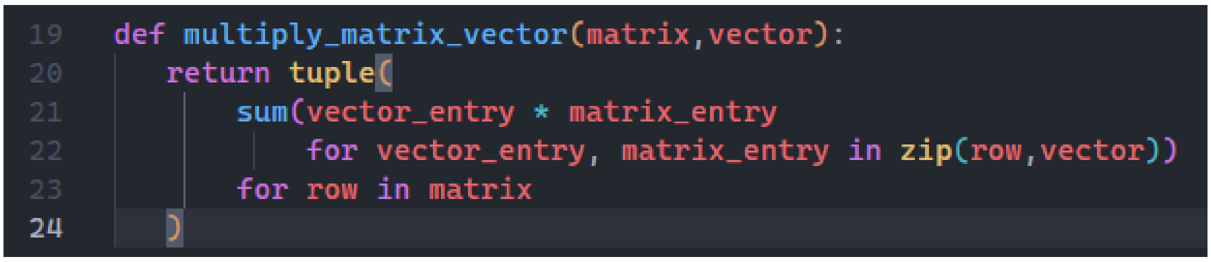


Using



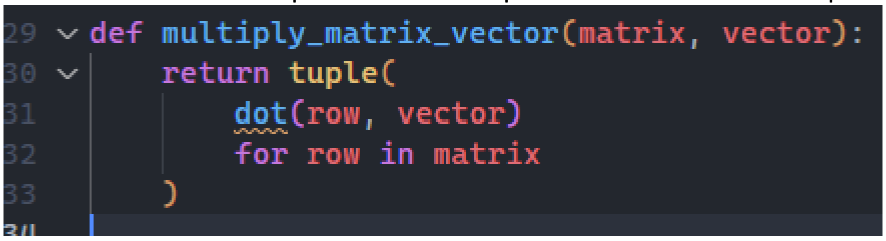
```python
def transform(v):
    m = ((2,1,1),(1,2,1),(1,1,2))
    return multiply_matrix_vector(m,v)
```

Exercise 5.7: Implement multiply_matrix_vector in a different way by using two nested comprehensions: one traversing the rows of the matrix and one traversing the entries of each row.



```python
def multiply_matrix_vector(matrix,vector):
    return tuple(
        sum(vector_entry * matrix_entry
            for vector_entry, matrix_entry in zip(row,vector))
        for row in matrix
    )
```

Exercise 5.8: Implement multiply_matrix_vector yet another way using the fact that the output coordinates are the dot products of the input matrix rows with the input vector.



```python
def multiply_matrix_vector(matrix, vector):
    return tuple(
        dot(row, vector)
        for row in matrix
    )
```

Exercise 5.9–Mini Project: Prove that all matrices represent linear transformations. Starting with the explicit formulas for multiplying a 2D vector by a 2-by–2 matrix or multiplying a 3D vector by a 3-by–3 matrix, prove that algebraically. That is, show that matrix multiplication preserves sums and scalar multiples.

For sums:

$$A\mathbf{u} + A\mathbf{v} = \begin{pmatrix} au_1 + bu_2 \\ cu_1 + du_2 \end{pmatrix} + \begin{pmatrix} av_1 + bv_2 \\ cv_1 + dv_2 \end{pmatrix} = \begin{pmatrix} au_1 + av_1 + bu_2 + bv_2 \\ cu_1 + cv_1 + du_2 + dv_2 \end{pmatrix}$$

$$A(\mathbf{u}+\mathbf{v}) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} u_1 + v_1 \\ u_2 + v_2 \end{pmatrix} = \begin{pmatrix} a(u_1 + v_1) + b(u_2 + v_2) \\ c(u_1 + v_1) + d(u_2 + v_2) \end{pmatrix} = \begin{pmatrix} au_1 + av_1 + bu_2 + bv_2 \\ cu_1 + cv_1 + du_2 + dv_2 \end{pmatrix}$$

For scalar multiples:

$$s\mathbf{v} = \begin{pmatrix} sv_1 \\ sv_2 \end{pmatrix}$$

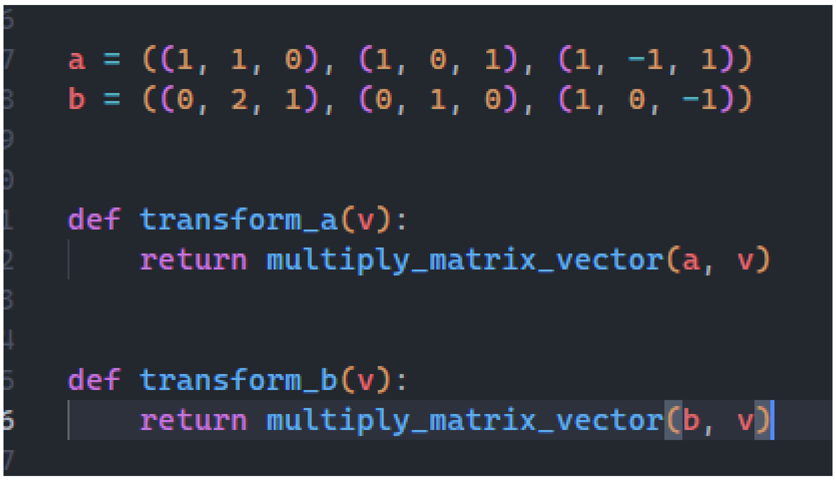
$$s(A\mathbf{v}) = \begin{pmatrix} s(av_1 + bv_2) \\ s(cv_1 + dv_2) \end{pmatrix} = \begin{pmatrix} sav_1 + sbv_2 \\ scv_1 + sdv_2 \end{pmatrix}$$

$$A(s\mathbf{v}) = \begin{pmatrix} a(sv_1) + b(sv_2) \\ c(sv_1) + d(sv_2) \end{pmatrix} = \begin{pmatrix} sav_1 + sbv_2 \\ scv_1 + sdv_2 \end{pmatrix}$$

Exercise 5.10: Once again, let's use the two matrices from section 5.1.3:

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & -1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

Write a function compose_a_b that executes the composition of the linear transformation for *A* and the linear transformation for *B*. Then use the infer _matrix function from a previous exercise in this section to show that infer_matrix(3, compose_a_b) is the same as the matrix product *AB*.



```python
a = ((1, 1, 0), (1, 0, 1), (1, -1, 1))
b = ((0, 2, 1), (0, 1, 0), (1, 0, -1))


def transform_a(v):
    return multiply_matrix_vector(a, v)


def transform_b(v):
    return multiply_matrix_vector(b, v)
```

Exercise 5.11-Mini Project: Find two, 2-by−2 matrices, neither of which is the identity matrix I_2, but whose product *is* the identity matrix.

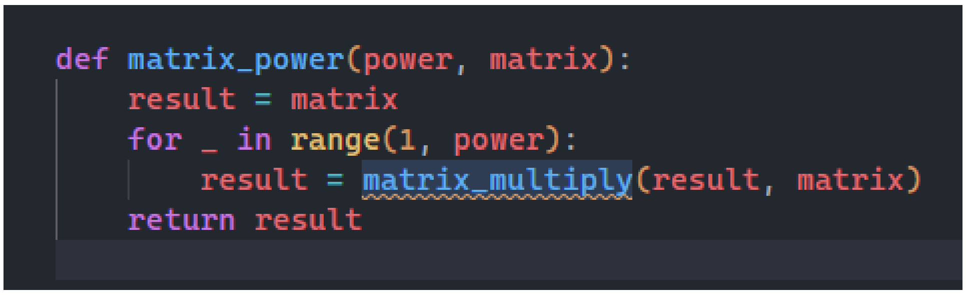
Matrix 1

0	1
-1	0

Matrix 2

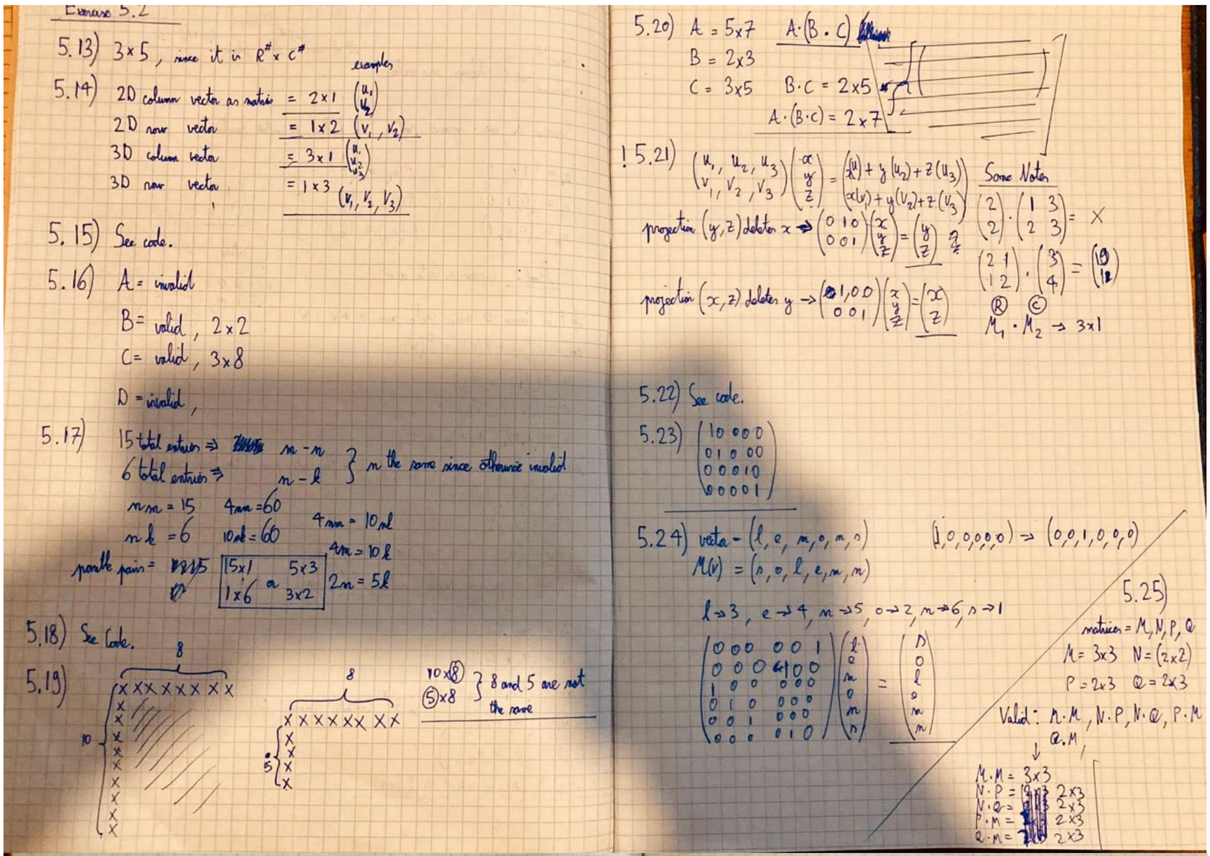
0	-1
1	0

Exercise 5.12: We can multiply a square matrix by itself any number of times. We can then think of successive matrix multiplications as "raising a matrix to a power." For a square matrix *A*, we can write *AA* as A^2; we can write *AAA* as A^3; and so on. Write a matrix_power (power, matrix) function that raises a matrix to the specified (whole number) power.



```python
def matrix_power(power, matrix):
    result = matrix
    for _ in range(1, power):
        result = matrix_multiply(result, matrix)
    return result
```

Exercises 5.13 – 15.29



Exercise 5.2

5.13) 3×5, since it is $R^{\#} \times C^{\#}$

5.14) examples

2D column vector as matrix	$= 2 \times 1$	$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$
2D row vector	$= 1 \times 2$	(v_1, v_2)
3D column vector	$= 3 \times 1$	$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$
3D row vector	$= 1 \times 3$	(v_1, v_2, v_3)

5.15) See code.

5.16) A = invalid

B = valid, 2×2

C = valid, 3×8

D = invalid,

5.17) 15 total entries ⇒ $m - n$

6 total entries ⇒ $n - l$ } n the same since otherwise invalid

$nm = 15$ $4nm = 60$ $4nm = 10nl$

$nl = 6$ $10nl = 60$ $4m = 10l$

possible pairs = $\boxed{\begin{matrix} 15 \times 1 & & 5 \times 3 \\ 1 \times 6 & \text{or} & 3 \times 2 \end{matrix}}$ $2m = 5l$

5.18) See Code.

5.19)

$$\overbrace{\text{X X X X X X X X}}^{8}$$
$$10 \left\{ \begin{matrix} X \\ X \\ X \\ X \\ X \\ X \\ X \\ X \\ X \\ X \end{matrix} \right.$$

$$\overbrace{\text{X X X X X X X X}}^{8}$$
$$5 \left\{ \begin{matrix} X \\ X \\ X \\ X \\ X \end{matrix} \right.$$

$10 \times \textcircled{8}$
$\textcircled{5} \times 8$ } 8 and 5 are not the same

5.20) $A = 5 \times 7$ $A \cdot (B \cdot C)$

$B = 2 \times 3$

$C = 3 \times 5$ $B \cdot C = 2 \times 5$

$A \cdot (B \cdot C) = 2 \times 7$

! 5.21) $\begin{pmatrix} u_1, & u_2, & u_3 \\ v_1, & v_2, & v_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x(u_1) + y(u_2) + z(u_3) \\ x(v_1) + y(v_2) + z(v_3) \end{pmatrix}$

projection (y, z) deletes x ⇒ $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ z \end{pmatrix}$

projection (x, z) deletes y → $\begin{pmatrix} 1, & 0, & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ z \end{pmatrix}$

Some Notes

$\begin{pmatrix} 2 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 3 \\ 2 & 3 \end{pmatrix} = X$

$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 10 \\ 11 \end{pmatrix}$

$\overset{\text{Ⓡ}}{M_1} \cdot \overset{\text{Ⓒ}}{M_2} \rightarrow 3 \times 1$

5.22) See code.

5.23) $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

5.24) vector = (l, e, m, o, n, s) $(1,0,0,0,0,0) \rightarrow (0,0,1,0,0,0)$

$M(v) = (s, o, l, e, m, n)$

$l \rightarrow 3$, $e \rightarrow 4$, $m \rightarrow 5$, $o \rightarrow 2$, $n \rightarrow 6$, $s \rightarrow 1$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} l \\ e \\ m \\ o \\ n \\ s \end{pmatrix} = \begin{pmatrix} s \\ o \\ l \\ e \\ m \\ n \end{pmatrix}$$

5.25)

matrices = M, N, P, Q

$M = 3 \times 3$ $N = (2 \times 2)$

$P = 2 \times 3$ $Q = 2 \times 3$

Valid: $M \cdot M$, $N \cdot P$, $N \cdot Q$, $P \cdot M$, $Q \cdot M$,

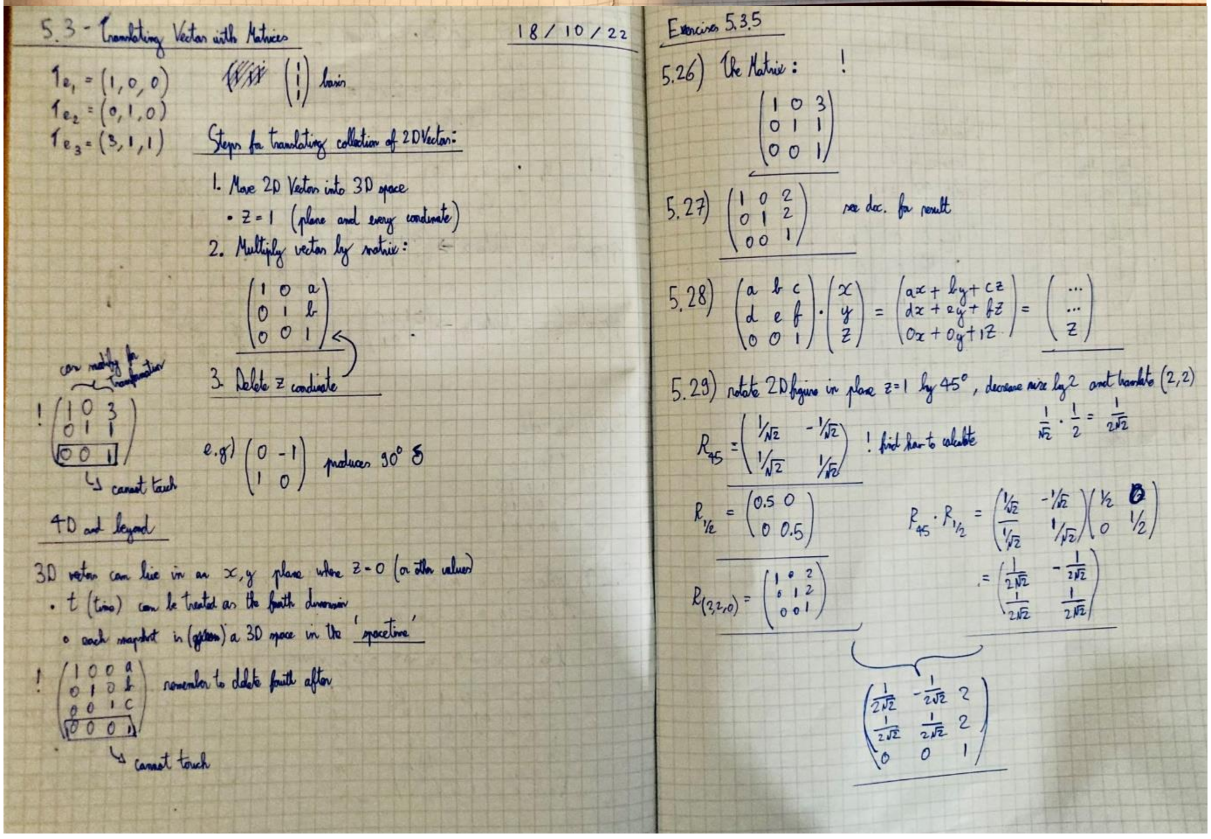
$M \cdot M = 3 \times 3$

$N \cdot P = 2 \times 3$

$N \cdot Q = 2 \times 3$

$P \cdot M = 2 \times 3$

$Q \cdot M = 2 \times 3$



5.3 - Translating Vector with Matrices 18 / 10 / 22

$1e_1 = (1, 0, 0)$ $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ basis

$1e_2 = (0, 1, 0)$

$1e_3 = (3, 1, 1)$

Steps for translating collection of 2D Vectors:

1. Move 2D Vectors into 3D space
   - $z = 1$ (plane and every coordinate)
2. Multiply vector by matrix:

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix}$$

3. Delete z coordinate

can modify for transformation

! $\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ \boxed{0 \quad 0 \quad 1} \end{pmatrix}$ ↳ cannot touch

e.g.) $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ produces 90° ↻

4D and beyond

3D vectors can live in an x, y plane where $z = 0$ (or other values)

- t (time) can be treated as the fourth dimension
  - each snapshot in (system) a 3D space in the 'spacetime'

! $\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ \boxed{0 \quad 0 \quad 0 \quad 1} \end{pmatrix}$ remember to delete fourth after

↳ cannot touch

Exercises 5.3.5

5.26) The Matrix: !

$$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

5.27) $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$ see doc. for result

5.28) $\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + by + cz \\ dx + ey + fz \\ 0x + 0y + 1z \end{pmatrix} = \begin{pmatrix} \dots \\ \dots \\ z \end{pmatrix}$

5.29) rotate 2D figure in plane $z = 1$ by 45°, decrease size by 2 and translate (2, 2)

$R_{45} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$! find how to calculate $\frac{1}{\sqrt{2}} \cdot \frac{1}{2} = \frac{1}{2\sqrt{2}}$

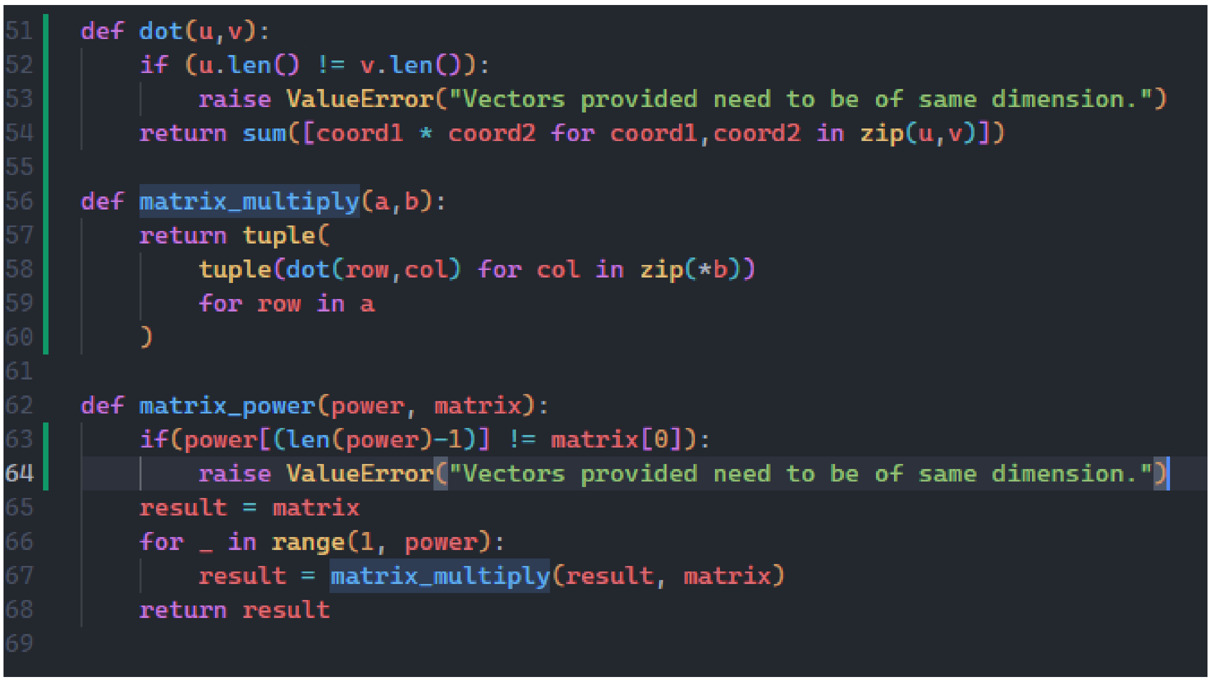
$R_{1/2} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$ $R_{45} \cdot R_{1/2} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$

$= \begin{pmatrix} \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \end{pmatrix}$

$R_{(2,2,0)} = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$

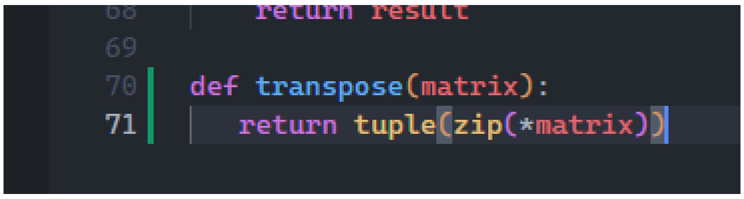
$$\begin{pmatrix} \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 2 \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Exercise 5.24−Mini Project: Consider the vector of six variables (*l*, *e*, *m*, *o*, *n*, *s*). Find the matrix for the linear transformation that acts on this vector to produce the vector (*s*, *o*, *l*, *e*, *m*, *n*) as a result.



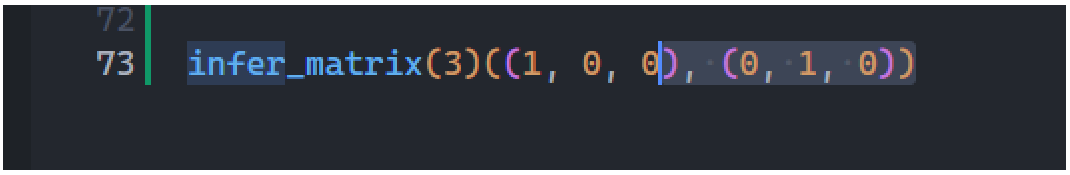
```python
def dot(u,v):
    if (u.len() != v.len()):
        raise ValueError("Vectors provided need to be of same dimension.")
    return sum([coord1 * coord2 for coord1,coord2 in zip(u,v)])

def matrix_multiply(a,b):
    return tuple(
        tuple(dot(row,col) for col in zip(*b))
        for row in a
    )

def matrix_power(power, matrix):
    if(power[(len(power)-1)] != matrix[0]):
        raise ValueError("Vectors provided need to be of same dimension.")
    result = matrix
    for _ in range(1, power):
        result = matrix_multiply(result, matrix)
    return result
```

Exercise 5.18: Write a function that turns a column vector into a row vector, or vice versa. Flipping a matrix on its side like this is called *transposition* and the resulting matrix is called the *transpose* of the original.



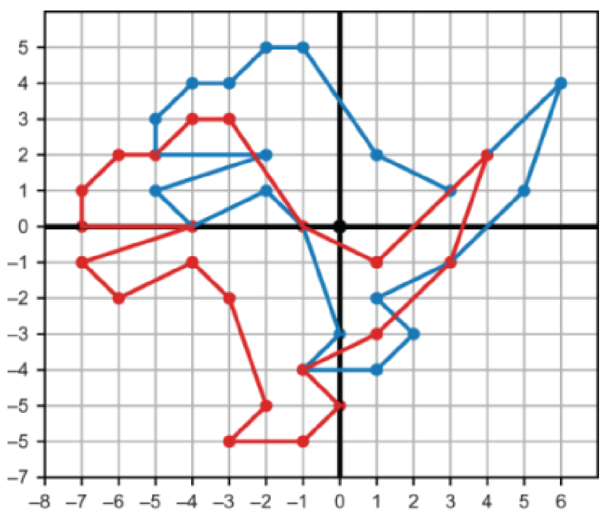
```python
    return result

def transpose(matrix):
    return tuple(zip(*matrix))
```

Exercise 5.22: Show by example that the infer_matrix function from a previous exercise can create matrices for linear functions whose inputs and outputs have different dimensions.



```python
infer_matrix(3)((1, 0, 0), (0, 1, 0))
```

Exercise 5.27: Come up with a matrix to translate the dinosaur by −2 units in the *x* direction and −2 units in the *y* direction. Execute the transformation and show the result.


5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
−7
−8
−7
−6
−5
−4
−3
−2
−1
0
1
2
3
4
5
6