In [1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("assignment3.ipynb")
```

Table of Contents

- Assignment 4: Modeling and Optimization
- Questions 1-3: Resource Allocation Problem
  - Question 1: Resource Constraints
    - Question 1.a: Modeling Resource Usage
    - Question 1.b: Resource Usage vs. Total Resource Constraint
    - Question 1.c: Feasible Region Boundary
    - Question 1.d: Interior of Feasible Region
    - Question 1.e: Visualizing the Feasible Region
  - Question 2: Objective Function
    - Question 2.a: Defining Objective Function
    - Question 2.b: Direction of Steepest Increase
  - Question 3: Putting Pieces Together
    - Question 3.a: Standard Form of a Linear Programming Problem
    - Question 3.b: Computing the Numerical Solution
    - Question 3.c: Plotting the optimal solution
- Question 4: Nutrition Problem
  -
    - Question 4.a: Define Constraints
    - Question 4.b: Create Python Variables
    - Question 4.c: Solve the Problem
    - Question 4.d: Interpreting the Results

Assignment 3: Modeling and Optimization

Mathematical modeling of a problem at hand give us a systematic way of finding a solution. For example, a maximum likelihood estimator (assuming it exists), $\hat{\theta}$, is a method for finding the parameter that maximizes the likelihood of the data L_n:

$$L_n(\hat{\theta}; x_1, x_2, \ldots, x_n) = \max_{\theta \in \Theta} L_n(\theta; x_1, x_2, \ldots, x_n)$$

Data $x_1, x_2, \ldots$, set of feasible parameters Θ, and likelihood function L_n are given. We find the parameter that "best" describe the data in the context of the likelihood function.

Many other applications of optimization exists, and this assignment will give a hands-on introduction to a simple linear programming problem.

In [2]:
```
import cvxpy as cp
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
```

Questions 1-3: Resource Allocation Problem

Hint: refer to Chapter 2.B of Introduction to Linear and Matrix Algebra for examples.

You are in charge of a company that makes two hot sauces: x_1 liters of Kapatio and x_2 liters of Zriracha. We will use optimization technique to find the "best" manufacturing strategy given our resource constraints.

First, we need to define what we mean by "best" strategy. In this scenario, the goal is to obtain the highest revenue possible. While doing so, there are resource constraints we must satisfy.

For example, in order to manufacture these two hot sauce products, different amount of peppers and vineger are needed. Also, we have only so much total resource available.

Ingridients	Kapatio	Zriracha	Total Available
Pepper	5	7	30
Vineger	4	2	12

Question 1: Resource Constraints

Question 1.a: Modeling Resource Usage

What is the equation for the amount of pepper needed to manufacture x_1 and x_2. What is the equation for the amount of vinegar?

$$Pepper = 5x_1 + 4x_2$$

$$Vinegar = 4x_1 + 2x_2$$

Question 1.b: Resource Usage vs. Total Resource Constraint

Total amount of pepper needed cannot exceed total available. Write down the inequality expressing this relationship. Do the same for vinegar. These inequalities are your resource constraints. Additinally, variables x_1 and x_2 are non-negative: i.e. amount of manufactured goods cannot be negative.

Rewrite the system of constraint inequalities into a matrix inequality: $Ax \leq b$, where $x = (x_1, x_2)^T$. Arrange rows of A and b such that:

- Row 1: total pepper amount constraint
- Row 2: total vinegar amount constraint
- Row 3: Kapatio non-negativity constraint
- Row 4: Zriracha non-negativity constraint

Less than symbol in $Ax \leq b$ means element-wise.

$$0 \leq 5x_1 + 7x_2 \leq 30$$

$$0 \leq 4x_1 + 2x_2 \leq 12$$

Define matrix `A1` and vector `b1` according to matrix inequality above.

```
In [3]: A1 = np.array([[5, 7], [4, 2], [-1, 0], [0, -1]])
        b1 = np.array([30, 12, 0, 0])
```

```
In [4]: grader.check("q1b2")
```

```
Out[4]:
```

q1b2 passed! 🙌

Visualizing Feasible Region

In a 2-dimensional plot, we will visualize the area that satisfies both of the resource constraints. Draw x_1 on the horizontal axis and x_2 on the vertical axis.

There will be two main components to the plot:

- **Lines** indicating constraint boundaries:
  e.g. the constraint $x_2 \geq 0$ has boundary at $x_2 = 0$.
- **Shaded area** indicating feasible regions:
  e.g., the whole region $x_2 > 0$ is to be shaded *if* $x_2 \geq 0$ was the only constraint. We will use shading to indicate the region where *all* constraints are satisfied.

```
In [5]: x1_line = np.linspace(-1, 10, 500)
        x2_line = np.linspace(-1, 10, 500)
```

Question 1.c: Feasible Region Boundary

In a list named `boundary`, create four data frames for each equality in $Ax = b$. These lines indicate where the feasible area ends. Set column names as

- `$x_1$`
- `$x_2$`

- `constraints`

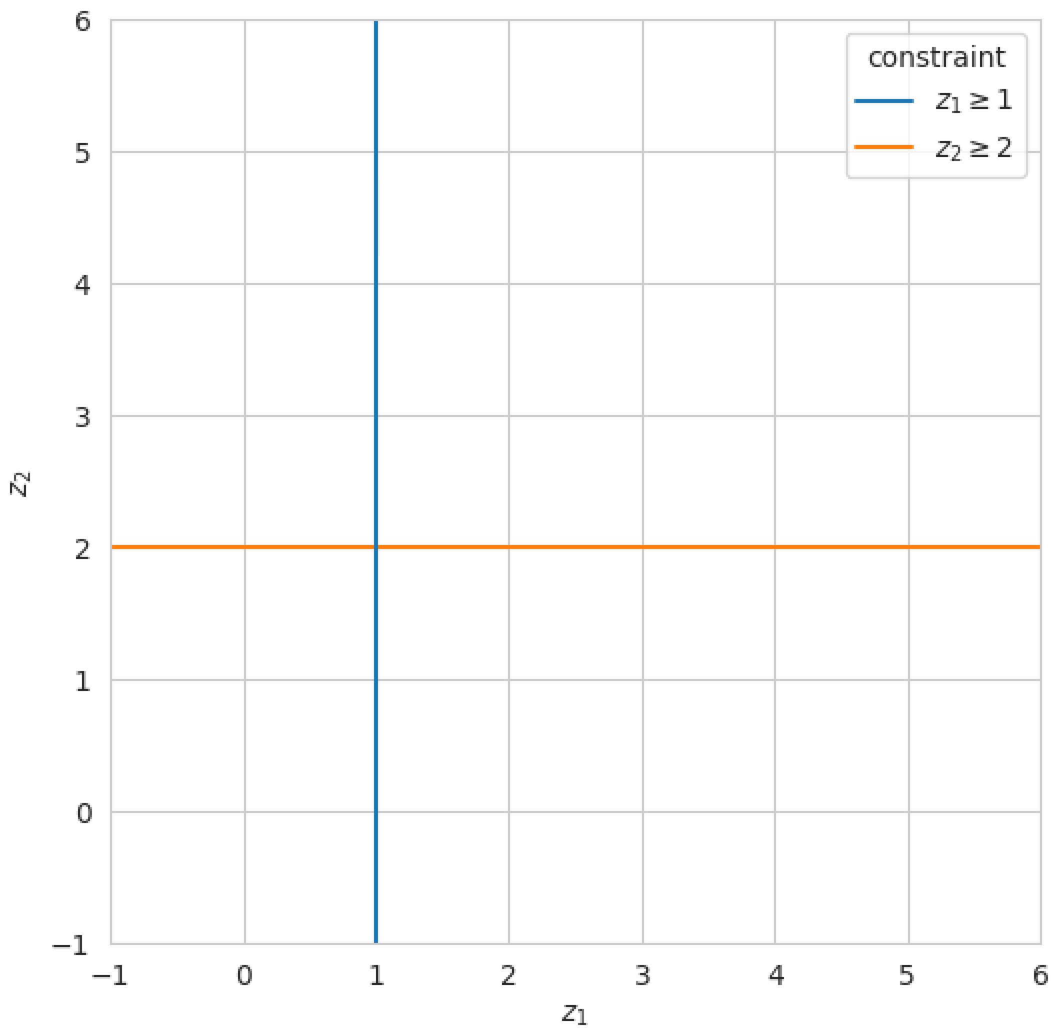
Note the use of latex codes.

Toy Example: Drawing Boundaries

Here is a **toy example** of drawing two constraint boundaries by constructing data frames:

In [6]:
```
z1_line = np.linspace(-1, 10, 500)
z2_line = np.linspace(-1, 10, 500)

boundary = [
    pd.DataFrame({
        '$z_1$': np.ones_like(z2_line)*1,
        '$z_2$': z2_line,
        'constraint': '$z_1\geq 1$'
    }),
    pd.DataFrame({
        '$z_1$': z1_line,
        '$z_2$': np.ones_like(z1_line)*2,
        'constraint': '$z_2\geq 2$'
    }),
]
fig, ax = plt.subplots(figsize=(6, 6))
sns.lineplot(x='$z_1$', y='$z_2$', hue='constraint', data=pd.concat(boundary
plt.xlim(-1, 6)
plt.ylim(-1, 6)
plt.show()
```



Sometimes, things just do not work as expected.

In the toy example code,

```
sns.lineplot(x='$z_1$', y='$z_2$', hue='constraint',
data=pd.concat(boundary), ax=ax).axvline(1)
```

what seems strange about the plotting command? Why was the strange code necessary?

Type your answer here, replacing this text.

Example: Resource Constraint Boundary

Now, create a data frame for the non-negativity constraint $x_2 \geq 0$ as follows:

```
In [7]: pd.DataFrame({'$x_1$':x1_line,              ## x_1 can take on any value
                      '$x_2$':x1_line * 0.0,         ## x_2 = 0
                      'constraint':'$x_2 \geq 0$'}), ## constraint equation for labe
```

```
Out[7]: (         $x_1$  $x_2$    constraint
         0    -1.000000   -0.0  $x_2 \geq 0$
         1    -0.977956   -0.0  $x_2 \geq 0$
         2    -0.955912   -0.0  $x_2 \geq 0$
         3    -0.933868   -0.0  $x_2 \geq 0$
         4    -0.911824   -0.0  $x_2 \geq 0$
         ..         ...    ...           ...
         495   9.911824    0.0  $x_2 \geq 0$
         496   9.933868    0.0  $x_2 \geq 0$
         497   9.955912    0.0  $x_2 \geq 0$
         498   9.977956    0.0  $x_2 \geq 0$
         499  10.000000    0.0  $x_2 \geq 0$

         [500 rows x 3 columns],)
```

Create a list named `boundary` containing four data frames (each corresponding to a constraint). Concatenate data frames in `boundary` to one data frame named `hull`.

```
In [8]: boundary = [
            pd.DataFrame({'$x_1$': x1_line, '$x_2$': (30 - 5 * x1_line) / 7, 'constr
            pd.DataFrame({'$x_1$': x1_line, '$x_2$': (12 - 4 * x1_line) / 2, 'constr
            pd.DataFrame({'$x_1$': x1_line, '$x_2$': np.zeros_like(x1_line), 'constr
            pd.DataFrame({'$x_1$': np.zeros_like(x1_line), '$x_2$': x1_line, 'constr
        ]
        hull = pd.concat(boundary)
```

```
In [9]: grader.check("q1c2")
```

```
Out[9]:
```

q1c2 passed! ✨

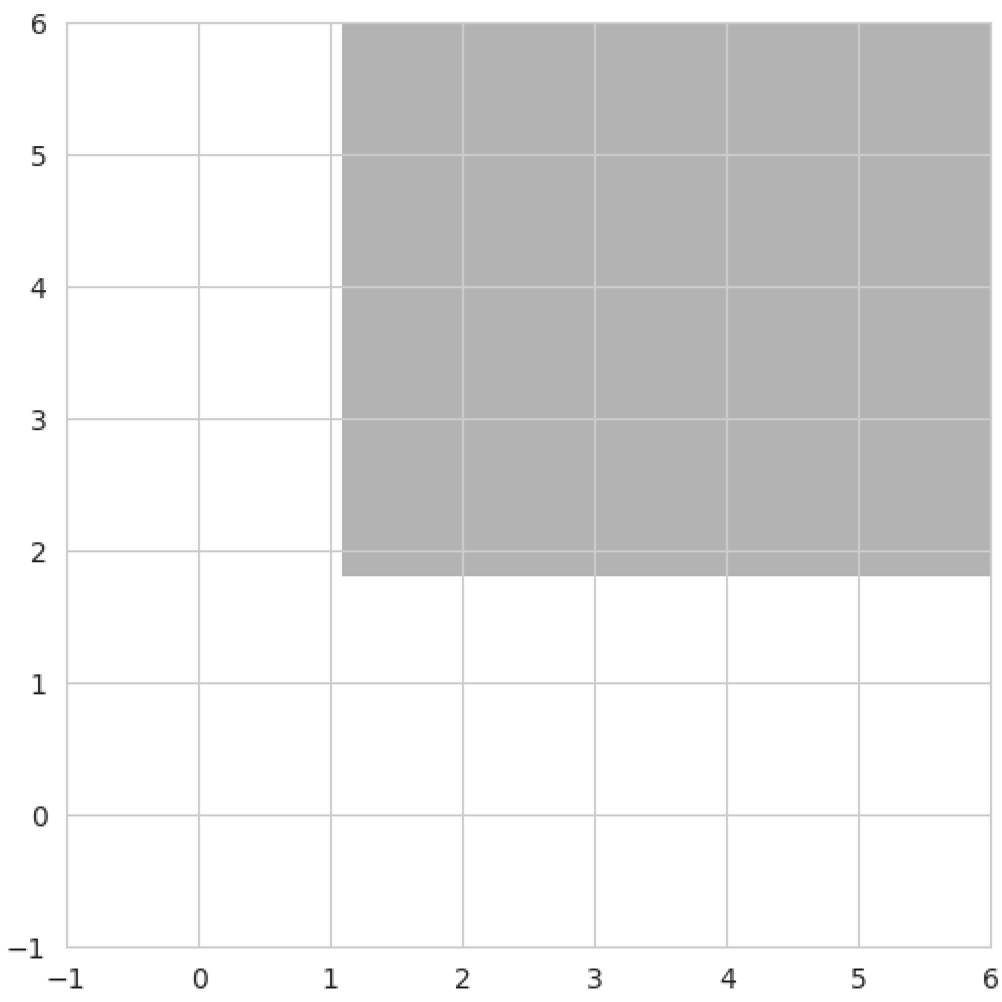
Question 1.d: Interior of Feasible Region

Previous question prepared constraint boundaries, $Ax = b$. In this question, we calculate the interior of the feasible region, which will be shaded in the visualization. First, create a 2-d array of x_1 and x_2 values. If a point (x_1, x_2) satisfies *every* constraint, the point will be colored grey.

For example, in order to shade $\{x_1 : x_1 \geq 1\} \cap \{x_2 : x_2 \geq 2\}$, we can use the `imshow` method.

```
In [10]: z1_line = np.linspace(-1, 6, 10)
         z2_line = np.linspace(-1, 6, 10)
         z1_grid, z2_grid = np.meshgrid(z1_line, z2_line)

         fig, az = plt.subplots(figsize=(6, 6))
         az.imshow(
             ((z1_grid >= 1) & (z2_grid >= 2)).astype(int),
             origin='lower',
             extent=(z1_grid.min(), z1_grid.max(), z2_grid.min(), z2_grid.max()),
             cmap="Greys", alpha=0.3, aspect='equal'
         )
         plt.xlim(-1, 6)
         plt.ylim(-1, 6)
         plt.show()
```



By dissecting the command below and reading the documentation, report what each of the following lines does:

- `((y1_grid >= 1) & (y2_grid >= 2)).astype(int)` (What is the output of running this command?)
- `origin='lower'`
- `extent=(y1_grid.min(), y1_grid.max(), y2_grid.min(), y2_grid.max())`
- `cmap='Greys'`
- `alpha=0.3`
- `aspect='equal'`

1. The line icreates a boolean mask checking if each element is greater than or equal to the value we set. True conditions will be satisified and false wont. The astype(int) converts it to an array of 1 for true and 0 for false.

2. The line sets the origin of the image to the lower corner

3. This specifies how far the image is on the grid. this defines the boundaries specified with the minimum and maximum values.
4. This is the grey scale coloring
5. This is the transparancy
6. This makes the x and y axis have equal scales

Question 1.e: Visualizing the Feasible Region

Finally, create a figure that shows constraint boundaries and the interior region shaded with a light grey color.

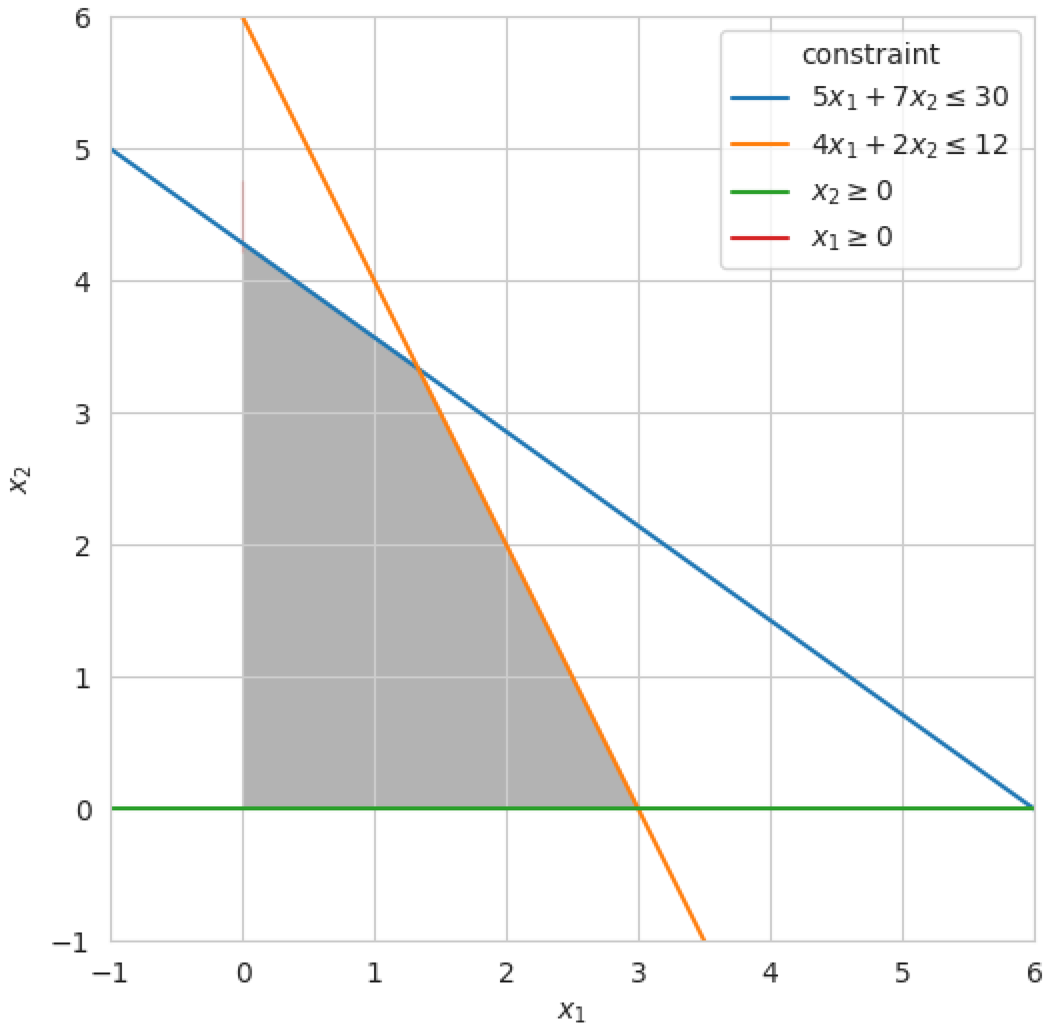
Your output will look like this:
hull

In [11]:
```
fig, ax = plt.subplots(figsize=(6, 6))
x1_grid, x2_grid = np.meshgrid(x1_line, x2_line)


ax.imshow(
    (
    (5 * x1_grid + 7 * x2_grid <= 30) &
    (4 * x1_grid + 2 * x2_grid <= 12) &
    (x2_grid >= 0) &
    (x1_grid >= 0)
    ),
    origin='lower',
    extent=(x1_grid.min(), x1_grid.max(), x2_grid.min(), x2_grid.max()),
    cmap="Greys", alpha = 0.3, aspect='equal')


# ax = sns.lineplot(???).axvline(???)
ax = sns.lineplot(x='$x_1$', y='$x_2$', hue='constraint', data=hull, ax=ax)

plt.xlim(-1, 6)
plt.ylim(-1, 6)
plt.show()
```



In the context of linear programming, $Ax \leq b$ is called the *feasible region* (including the appropriate sections of the boundaries). Denote the (shaded) feasible region as set C. Points $(x_1, x_2) \in C$ satisfy all of the constraints.

Describe in plain words the feasible region in the context of hot sauce manufacturing. Specifically, which constraint is violated (if any) by a point at:

- $(x_1, x_2) = (4, 1)$
- $(x_1, x_2) = (0, 5)$
- $(x_1, x_2) = (3, 4)$

Type your answer here, replacing this text.

Question 2: Objective Function

Question 2.a: Defining Objective Function

Suppose the hot sauces are sold at the same price: $5 per liter.

What is the equation $f(x)$ for the total revenue as a function of $x_1$ and $x_2$?

The function $f(x)$ is called the objective function.

Type your answer here, replacing this text.

Objective function $f(x)$ is a linear function in x. Therefore, $f(x)$ is a 2-dimesional hyperplane. Note that each value of $f(x)$ defines a line in (x_1, x_2) plane.

For example, $f(x) = 0 = c_1x_1 + c_2x_2$ defines a line. A subspace of equal function value is sometimes referred to as a *level set* or a *contour line* when visualized.

First, create a numpy array of prices `c` for the two hot sauces, $x_1$ and $x_2$. Then, create a list `f_vals` containing four data frames of contour lines, $f(x) \in \{0, 10, 20, 30\}$. by creating one data frame for each contour line.

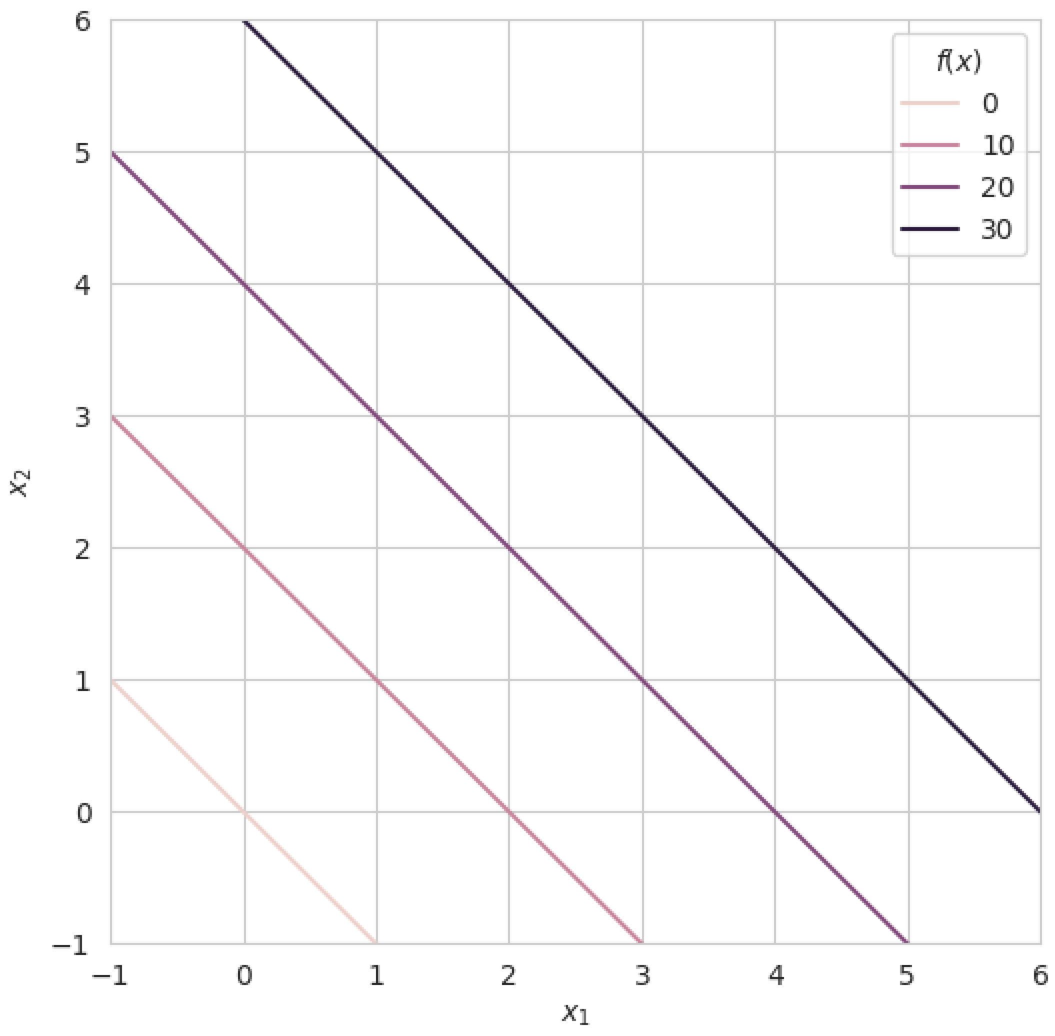
In [12]:
```python
c = np.array([5, 5])

fig, ax = plt.subplots(figsize=(6, 6))
contours = [
    pd.DataFrame({
        '$x_1$': x1_line,
        '$x_2$': (0 - c[0]*x1_line) / c[1],
        '$f(x)$': 0
    }),
    pd.DataFrame({
        '$x_1$': x1_line,
        '$x_2$': (10 - c[0]*x1_line) / c[1],
        '$f(x)$': 10
    }),
    pd.DataFrame({
        '$x_1$': x1_line,
        '$x_2$': (20 - c[0]*x1_line) / c[1],
        '$f(x)$': 20
    }),
    pd.DataFrame({
        '$x_1$': x1_line,
        '$x_2$': (30 - c[0]*x1_line) / c[1],
        '$f(x)$': 30
    })
]

f_vals = pd.concat(contours)

# ax = sns.lineplot(???)
ax = sns.lineplot(x='$x_1$', y='$x_2$', hue='$f(x)$', data=f_vals, ax=ax)

plt.xlim(-1, 6)
plt.ylim(-1, 6)
plt.show()
```



```
In [13]: grader.check("q2a2")
```

```
Out[13]:
```

q2a2 passed! 🙌

Question 2.b: Direction of Steepest Increase

Since we want to maximize revenue, we want to increase our objective function as much as possible. Analogous to the minimization example given in a previous lecture, we can repeatedly move in the direction of function increase. In order to determine such direction, compute the gradient of $f(x)$ at $x = (0,0)^T$:

$$\nabla_x f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \end{pmatrix}$$

Regardless of what each variable is, the gradient is (5,5)

Question 3: Putting Pieces Together

Question 3.a: Standard Form of a Linear Programming Problem

Write down the so-called the *standard form* of a linear programming problem:

$$\max_x f(x)$$
$$\text{subject to } Ax \le b$$

Specifically, write the obejective as an inner product of two vectors: $f(x) = c^T x$, and write the constraint as a vector inequality involving a matrix-vector prduct: $Ax \le b$, where A is a 4-by-2 matrix.

$$\max_x (c1, c2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ subject to } \begin{pmatrix} 5 & 7 \\ 4 & 2 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \le \begin{pmatrix} 30 \\ 12 \\ 0 \\ 0 \end{pmatrix}$$

Question 3.b: Computing the Numerical Solution

Therefore, *maximizing* the revenue is a search over the feasible region for the best point $x^* = (x_1^*, x_2^*)$ that gives the largest revenue. On the otherhand, any *infeasible* point *not* in the feasible region cannot be a solution to the constrained optimization problem.

Notationally, the following expression means the same thing:

$$x^* = \arg \max_{\{x : Ax \le b\}} f(x)$$

Using CVXPY, solve for the resource allocation problem with constraints.

In [14]:
```
# define variables
x = cp.Variable(2)

# define the linear program
problem = cp.Problem(
    cp.Maximize(5 * sum(x)),
    [A1 @ x <= b1]
)

fstar1 = problem.solve()                                                   # n
xstar1 = pd.DataFrame(x.value.reshape(1, 2), columns=['$x_1$', '$x_2$']) # n
```

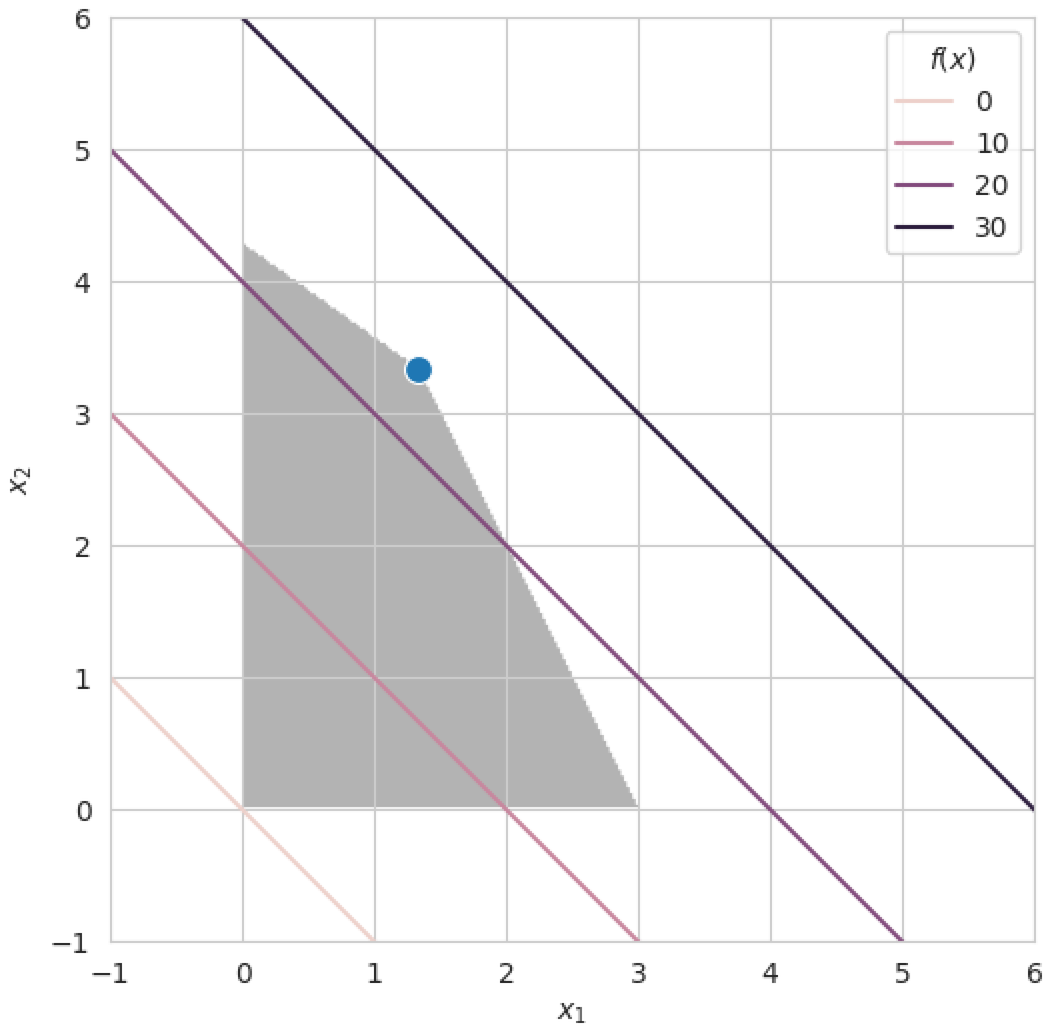
In [15]:
```
grader.check("q3b")
```

Out[15]:

q3b passed! 🌈

Question 3.c: Plotting the optimal solution

In [16]:
```
fig, ax = plt.subplots(figsize=(6, 6))
x1_grid, x2_grid = np.meshgrid(x1_line, x2_line)
ax.imshow(
    (
        (A1[0,0]*x1_grid + A1[0,1]*x2_grid <= b1[0]) & # Pepper constraints
        (A1[1,0]*x1_grid + A1[1,1]*x2_grid <= b1[1]) & # Vinegar constraints
        (A1[2,0]*x1_grid + A1[2,1]*x2_grid <= b1[2]) &
        (A1[3,0]*x1_grid + A1[3,1]*x2_grid <= b1[3])   # non-negativity cons
    ),
    origin='lower',
    extent=(x1_grid.min(), x1_grid.max(), x2_grid.min(), x2_grid.max()),
    cmap="Greys", alpha = 0.3, aspect='equal'
)
sns.scatterplot(x='$x_1$', y='$x_2$', data=xstar1, ax=ax, s=100)
sns.lineplot(x='$x_1$', y='$x_2$', hue='$f(x)$', data=f_vals.reset_index(),
plt.xlim(-1, 6)
plt.ylim(-1, 6)
plt.show()
```



Question 4: Nutrition Problem

During the second world war, the US Army set out to save money without damaging the nutritional health of members of the armed forces.

> According to this source, the following problem is a *simple variation of the well-known diet problem that was posed by George Stigler and George Dantzig: how to choose foods that satisfy nutritional requirements while minimizing costs or maximizing satiety.*
>
> *Stigler solved his model "by hand" because technology at the time did not yet support more sophisticated methods. However, in 1947, Jack Laderman, of the US National Bureau of Standards, applied the simplex method (an algorithm that was recently proposed by George Dantzig) to Stigler's model. Laderman and his team of nine linear programmers, working on desk calculators, showed that Stigler's heuristic approximation was very close to optimal (only 24 cents per year over the optimum found by the simplex method) and thus demonstrated the practicality of the simplex method on large-scale, real-world problems.*
>
> *The problem that is solved in this example is to minimize the cost of a diet that satisfies certain nutritional constraints.*

The file `foods.csv` contains calorie, nutritional content, serving size, and price per serving information about 64 foods. Read it into a data frame named `foods`.

In [17]:
```
foods = pd.read_csv("foods.csv")
print(foods)
```

```
                      Name  Calories  Cholesterol  Total_Fat  Sodium  \
0           Frozen Broccoli      73.8          0.0        0.8    68.2
1              Carrots, Raw      23.7          0.0        0.1    19.2
2               Celery, Raw       6.4          0.0        0.1    34.8
3               Frozen Corn      72.2          0.0        0.6     2.5
4       Lettuce, Iceberg,Raw       2.6          0.0        0.0     1.8
..                      ...       ...          ...        ...     ...
59          New Eng Clam Chwd     175.7         10.0        5.0  1864.9
60              Tomato Soup     170.7          0.0        3.8  1744.4
61  New Eng Clam Chwd, w/Mlk     163.7         22.3        6.6   992.0
62     Crm Mshrm Soup, w/Mlk     203.4         19.8       13.6  1076.3
63   Bean Bacon Soup, w/Watr     172.0          2.5        5.9   951.3

    Carbohydrates  Dietary_Fiber  Protein    Vit_A  Vit_C  Calcium  Iron  \
0            13.6            8.5      8.0   5867.4  160.2    159.0   2.3
1             5.6            1.6      0.6  15471.0    5.1     14.9   0.3
2             1.5            0.7      0.3     53.6    2.8     16.0   0.2
3            17.1            2.0      2.5    106.6    5.2      3.3   0.3
4             0.4            0.3      0.2     66.0    0.8      3.8   0.1
..            ...            ...      ...      ...    ...      ...   ...
59           21.8            1.5     10.9     20.1    4.8     82.8   2.8
60           33.2            1.0      4.1   1393.0  133.0     27.6   3.5
61           16.6            1.5      9.5    163.7    3.5    186.0   1.5
62           15.0            0.5      6.1    153.8    2.2    178.6   0.6
63           22.8            8.6      7.9    888.0    1.5     81.0   2.0

              Serving  Price/Serving ($)
0           10 Oz Pkg               0.16
1   1/2 Cup Shredded               0.07
2             1 Stalk               0.04
3             1/2 Cup               0.18
4              1 Leaf               0.02
..                ...                ...
59     1 C (8 Fl Oz)               0.75
60     1 C (8 Fl Oz)               0.39
61     1 C (8 Fl Oz)               0.99
62     1 C (8 Fl Oz)               0.65
63     1 C (8 Fl Oz)               0.67

[64 rows x 14 columns]
```

The file `nutritional_constraints.csv` contains healthy nutritional range constraints. Minimum and maximum allowed nutritional contents can be found in this file. Name the variable `requirements`.

In [18]:
```
requirements= pd.read_csv("nutritional_constraints.csv")
print(requirements)
```

```
             Name Unit   Min    Max
0        Calories  cal  2000   2250
1     Cholesterol   mg     0    300
2       Total_Fat    g     0     65
3          Sodium   mg     0   2400
4   Carbohydrates    g     0    300
5   Dietary_Fiber    g    25    100
6         Protein    g    50    100
7           Vit_A   IU  5000  50000
8           Vit_C   IU    50  20000
9         Calcium   mg   800   1600
10           Iron   mg    10     30
```

Extract the nutritional content of foods into a 2-d array named `ncontent`.

In [19]:
```
ncontent = foods.loc[:, "Name":"Iron"].set_index("Name")
print(ncontent)
```

```
                          Calories  Cholesterol  Total_Fat  Sodium  \
Name
Frozen Broccoli               73.8          0.0        0.8    68.2
Carrots, Raw                  23.7          0.0        0.1    19.2
Celery, Raw                    6.4          0.0        0.1    34.8
Frozen Corn                   72.2          0.0        0.6     2.5
Lettuce, Iceberg,Raw           2.6          0.0        0.0     1.8
...                            ...          ...        ...     ...
New Eng Clam Chwd            175.7         10.0        5.0  1864.9
Tomato Soup                  170.7          0.0        3.8  1744.4
New Eng Clam Chwd, w/Mlk     163.7         22.3        6.6   992.0
Crm Mshrm Soup, w/Mlk        203.4         19.8       13.6  1076.3
Bean Bacon Soup, w/Watr      172.0          2.5        5.9   951.3

                          Carbohydrates  Dietary_Fiber  Protein    Vit_A  \
Name
Frozen Broccoli                    13.6            8.5      8.0   5867.4
Carrots, Raw                        5.6            1.6      0.6  15471.0
Celery, Raw                         1.5            0.7      0.3     53.6
Frozen Corn                        17.1            2.0      2.5    106.6
Lettuce, Iceberg,Raw                0.4            0.3      0.2     66.0
...                                 ...            ...      ...      ...
New Eng Clam Chwd                  21.8            1.5     10.9     20.1
Tomato Soup                        33.2            1.0      4.1   1393.0
New Eng Clam Chwd, w/Mlk           16.6            1.5      9.5    163.7
Crm Mshrm Soup, w/Mlk              15.0            0.5      6.1    153.8
Bean Bacon Soup, w/Watr            22.8            8.6      7.9    888.0

                          Vit_C  Calcium  Iron
Name
Frozen Broccoli           160.2    159.0   2.3
Carrots, Raw                5.1     14.9   0.3
Celery, Raw                 2.8     16.0   0.2
Frozen Corn                 5.2      3.3   0.3
Lettuce, Iceberg,Raw        0.8      3.8   0.1
...                         ...      ...   ...
New Eng Clam Chwd           4.8     82.8   2.8
Tomato Soup               133.0     27.6   3.5
New Eng Clam Chwd, w/Mlk    3.5    186.0   1.5
Crm Mshrm Soup, w/Mlk       2.2    178.6   0.6
Bean Bacon Soup, w/Watr     1.5     81.0   2.0

[64 rows x 11 columns]
```

Question 4.a: Define Constraints

To avoid eating the same foods, limit each food intake to be 2 or less. Also, one cannot consume less than zero servings. Furthermore, apply the nutritional constraints as specified in `nutritional_constraints.csv` (assume that the units are the same as food nutritional contents)

Note that a range constraints, e.g., $2000 \leq \text{total calories} \leq 2250$, can be written as two constraints: $\text{total calories} \leq 2250$ and $-\text{total calories} \leq -2000$. Hence, we can rewrite caloric intake constraints as

$$-(\text{calories in frozen broccoli})x_0 - (\text{calories in raw carrots})x_1 - \cdots \leq -2000$$
$$- (\text{calories in bean bacon soup, w/watr})x_{63} = -c^T x$$
$$\text{calories in frozen broccoli})x_0 + (\text{calories in raw carrots})x_1 + \cdots \quad \leq 2250$$
$$+ (\text{calories in bean bacon soup, w/watr})x_{63} = c^T x$$

where vector c contains calorie information for all 64 foods and x containts servings consumed of each food. Matrix U and vector w would be such that

$$U = \begin{pmatrix} -c^T \\ c^T \end{pmatrix} \text{ and } w = \begin{pmatrix} -2000 \\ 2250 \end{pmatrix},$$

and the matrix-vector inequality would be $Ux \leq w$. Range constraints of each food can be implemented similarly with identity matrices.

Denote nutritional content information from `foods` data frame as A and denote the `Min` and `Max` columns of `requirements` as vector b_L and b_U, respectively. Construct M and d in $Mx \leq d$ using I (identity matrix), A, b_L, b_U, and other constants, so that all the range constraints are expressed in $Mx \leq d$. (This is a theory question. No coding is involved)

$$Mx \leq d = \begin{pmatrix} -I \\ I \\ -A^T \\ A^T \end{pmatrix} x \leq \begin{pmatrix} 0 \\ 2 \\ -b_L \\ b_l \end{pmatrix}$$

Question 4.b: Create Python Variables

Denote the servings of each food as x_i where i is the row index of each food in `foods` data frame: i.e. x_0 indicates number of servings of frozen broccoli, x_1 indicates that of raw carrots, etc.

- Create cost vector `cost` that gives per serving cost.
- Create matrix `M` and `d` that lists nutritional content in the following order:
  - Non-negativity constraint of food consumed: i.e. 0 servings or more
  - Upper limit on food consumed: i.e. 2 servings or less
  - Lower limit on consumption of each nutrition: i.e. following `Min` column
  - Upper limit on consumption of each nutrition: i.e. following `Max` column

In [20]:
```
M = np.concatenate(
    (np.eye(64) * -1,
     np.eye(64),
     -ncontent.T,
     ncontent.T),
    axis=0
)

d = np.concatenate(
    (np.zeros((64, 1)),
    np.ones((64, 1)) * 2,
    -requirements['Min'].values.reshape(-1, 1),
    requirements['Max'].values.reshape(-1, 1)),
    axis=0
).reshape(-1,)

cost = foods['Price/Serving ($)'].values
```

In [21]:
```
grader.check("q4b")
```

Out[21]:

q4b passed! 🚀

Question 4.c: Solve the Problem

Create cvxpy variable `servings` to represent the number of servings of food, and use cvxpy to solve for the optimal solution.

Choose ECOS as your solver.

In [22]:
```
servings = cp.Variable(64)

# define the linear program
nutrition_problem = cp.Problem(
    cp.Minimize(servings @ cost.T),
    [M@servings <= d]

)

# solve linear programming problem with ECOS solver
# https://www.cvxpy.org/tutorial/advanced/index.html?highlight=osqp#choosing
fstar2 = nutrition_problem.solve(solver = "ECOS")
xstar2 = servings.value
```

In [23]:
```
grader.check("q4c")
```

Out[23]:

q4c passed! 💯

Question 4.d: Interpreting the Results

State the results in the context of the problem. How much of each food was consumed? List the foods and their calculated amounts. What is the total cost of feeding one soldier?

In [24]:
```
print('the average cost of food for one soldier is $', fstar2)
print(pd.DataFrame({'Food': foods['Name'], 'Price ($)': xstar2}))
```

```
the average cost of food for one soldier is $ 1.2484722937208912
                        Food     Price ($)
0            Frozen Broccoli  8.024228e-02
1               Carrots, Raw  2.240289e-01
2                Celery, Raw  1.401558e-11
3                Frozen Corn  3.148633e-12
4       Lettuce, Iceberg,Raw  3.018374e-11
..                       ...           ...
59          New Eng Clam Chwd  6.674082e-13
60                Tomato Soup  1.923350e-12
61  New Eng Clam Chwd, w/Mlk  3.453333e-13
62     Crm Mshrm Soup, w/Mlk  7.663001e-13
63   Bean Bacon Soup, w/Watr  7.326201e-13

[64 rows x 2 columns]
```

(PSTAT 234) Question 5.a: Assignment Problem

Given a set $\mathcal{S}$ of m people, a set $\mathcal{D}$ of m tasks, and for each $i \in \mathcal{S}, j \in \mathcal{D}$ a cost c_{ij} associated with assigning person i to task j, the assignment problem is to assign each person to one and only one task in such a manner that each task gets covered by someone and the total cost of the assignments is minimized. If we let

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned task } j, \\ 0 & \text{otherwise,} \end{cases}$$

then the objective function can be written as

$$\text{minimize} \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij}.$$

Note that x_{ij} values are constrained to be 0 or 1. This is also a constraint. Suppose that, additionally, we would like to build in the constraints that

1. Each person is assigned exactly one task.
2. Every task gets covered by someone.

What equality constraints can you add?

Type your answer here, replacing this text.

(PSTAT 234) Question 5.b: Transportation Problem

Transportation problem: i and j indicate pairs of locations, quantity of goods (x_{ij}), supply (s_i), demand (d_j), transportation network ($\mathcal{A}$)

$$\begin{aligned} \min_{x=(x_{ij})\geq 0} \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \\ \text{sugject to} \sum_{j:(i,j)\in\mathcal{A}} x_{ij} \leq s_i \quad \text{for all } i \\ \sum_{i:(i,j)\in\mathcal{A}} x_{ij} \geq d_j \quad \text{for all } j \end{aligned}$$

Describe the assignment problem as a special case of the transportation problem. Draw analogy of supply network ($\mathcal{A}$), supply (s_i), demand (d_j), and quantity of goods (x_{ij}) in the assignment problem context.

Type your answer here, replacing this text.

Cell intentionally blank

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [25]: grader.check_all()
```

```
Out[25]: q1b2 results: All test cases passed!

q1c2 results: All test cases passed!

q2a2 results: All test cases passed!

q3b results: All test cases passed!

q4b results: All test cases passed!

q4c results: All test cases passed!
```

Submission

1. Save file to confirm all changes are on disk
2. Run *Kernel > Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File > Save and export Notebook as > HTML*.
5. Open in Google Chrome and print to PDF.
6. Submit to Gradescope