Part 1: Vectors and Graphics

1.1 Data Analysis with Python

numpy: https://numpy.org/doc/stable/
matplotlib: https://matplotlib.org/stable/index.html
pandas: https://pandas.pydata.org/docs/

1.2 几个基本概念

vector（向量）

linear transformation（线性变换）：线性变换是一种函数，将一个向量作为输入并返回一个向量作为输出, 同时保持所操作向量（在特殊意义上）的几何形状。例如, 如果一个向量（点）的集合位于二维平面的一条直线上，在应用线性变换后，它们仍然会位于一条直线上。

matrix（矩阵）: 矩阵是一种可以表示线性变换的矩形数组。

vector space（向量空间）

dimension（维度）

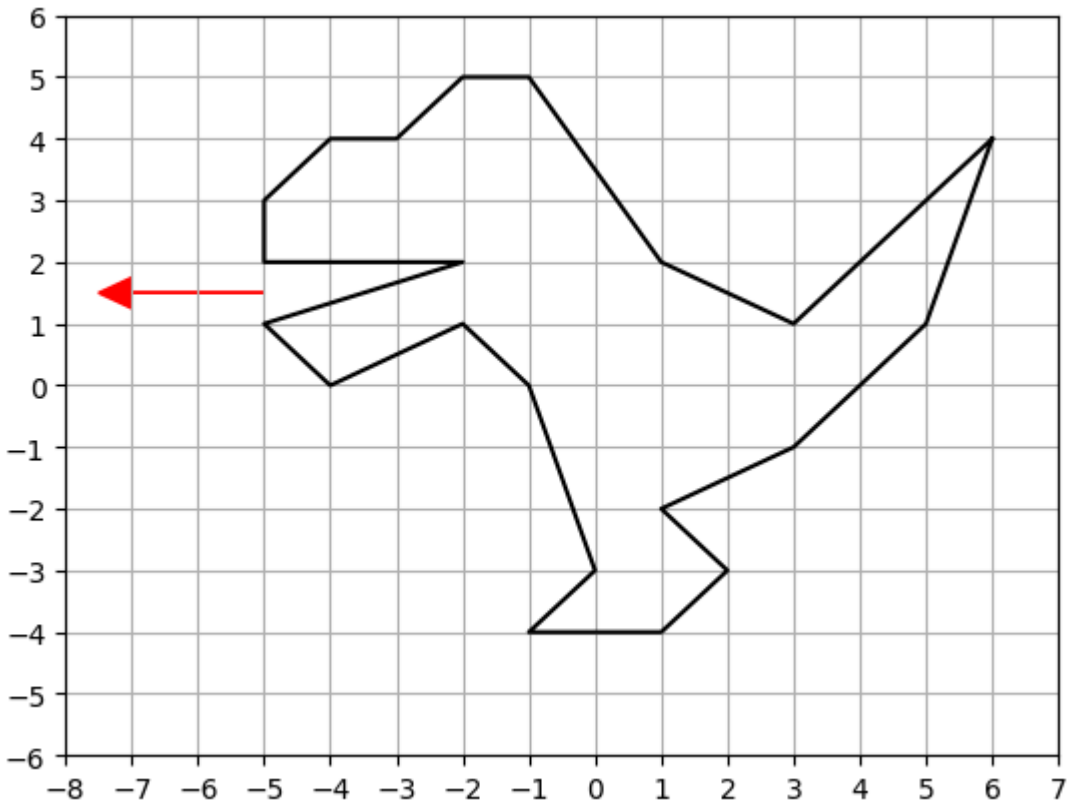
system of linear equations（线性方程组）

1.3 二维向量绘图

二维向量一个点相较于原点的位置，可以表示两点之间的位置关系。编程中常常使用 [x，y] 表示一个二维向量。

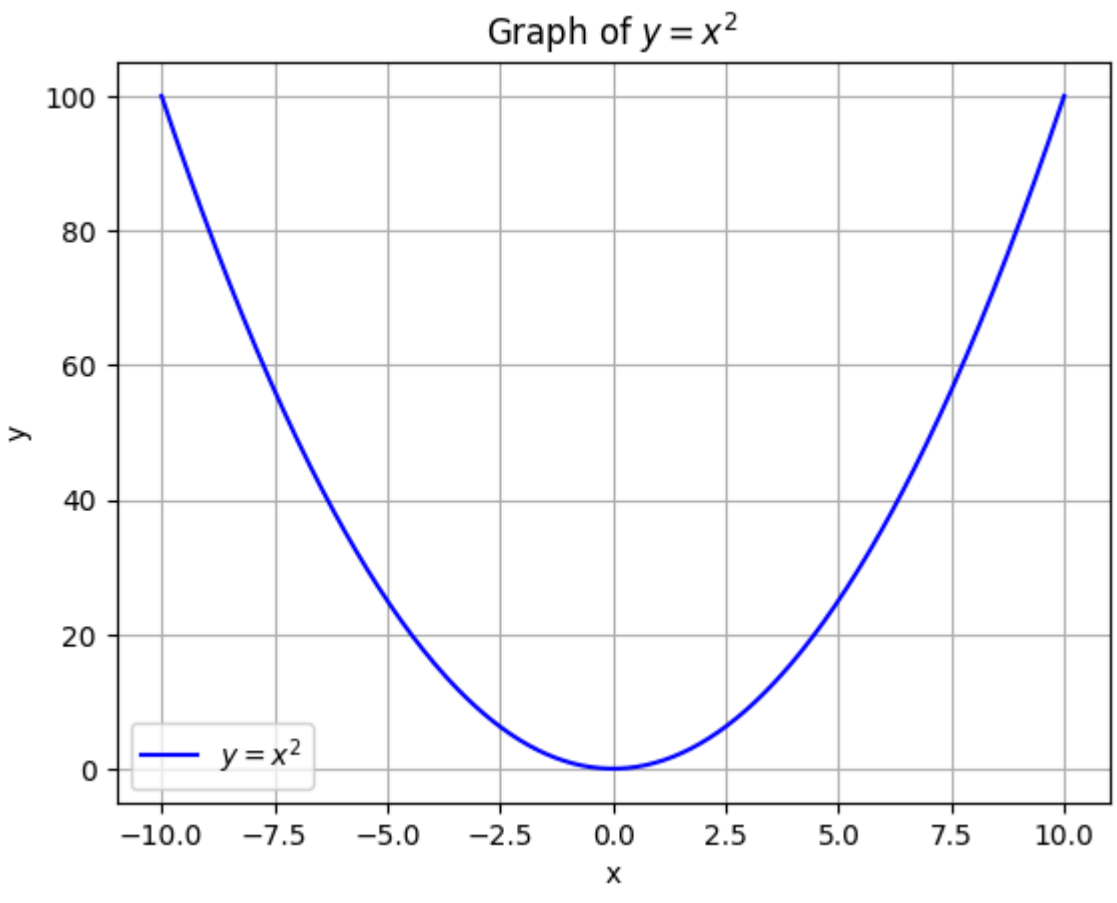
练习: 使用matplotlib画箭头和线

In [ ]:
```python
import matplotlib.pyplot as plt
import numpy as np

# 创建一个画布和坐标轴
fig, ax = plt.subplots()

# 画线
ax.plot([6, 3, 1, -1, -2, -3, -4, -5, -5, -2, -5, -4, -2, -1,  0, -1,  1,  2,  1,  3, 5, 6],
        [4, 1, 2,  5,  5,  4,  4,  3,  2,  2,  1,  0,  1,  0, -3, -4, -4, -3, -2, -1, 1, 4],
        color='black')

# 画箭头
# x, y 是箭头的起点坐标
# dx, dy 是箭头的长度和方向(可以理解为起点坐标当作原点，箭头的终点坐标是原点加上 dx, dy)
# head_width, head_length 是箭头的宽度和长度
# fc, ec 是箭头的填充颜色和边框颜色
ax.arrow(x=-5, y=1.5, dx=-2, dy=0, head_width=0.5, head_length=0.5, fc='red', ec='red')

# 设置坐标轴的范围
ax.set_xlim(-8, 7)
ax.set_ylim(-6, 6)

# 使用 numpy 生成间隔相等的刻度
ax.set_xticks(np.arange(-8, 8, 1))
ax.set_yticks(np.arange(-6, 7, 1))

# 显示网格线
ax.grid(True)

# 显示图形
plt.show()
```



练习：绘制函数y=x^2的图像

In [ ]:
```python
import matplotlib.pyplot as plt
import numpy as np

# 生成 x 值的数组
x = np.linspace(-10, 10, 400)  # 在区间 [-10, 10] 生成400个点

# 计算 y 值
y = x**2

# 创建一个图形和坐标轴
fig, ax = plt.subplots()

# 画 y = x^2 的图像, 本质就是取足够多的点画两点之间的线
ax.plot(x, y, label='$y = x^2$', color='blue')  # 这里使用蓝色的线条并添加标签

# 设置标题和标签
ax.set_title('Graph of $y = x^2$')
ax.set_xlabel('x')
ax.set_ylabel('y')

# 显示网格线
ax.grid(True)

# 显示图例
ax.legend()

# 显示图形
plt.show()
```



1.4 二维向量运算

向量加法和减法（平移变换）：用python很容易实现，不过实际运用时可以使用numpy库，如下。向量和表示一个向量相较于原向量的位置变化。

In [ ]:
```python
def add(v1, v2):
    return (v1[0] + v2[0], v1[1] + v2[1])
```

In [ ]:
```python
import numpy as np

# 创建两个向量
v1 = np.array([1, 2])
v2 = np.array([2, 1])

# 向量加法运算
v3 = v1 + v2

print(v3)
```

```
[3 3]
```

向量的数乘：即为向量的重复相加。如下，numpy库实现。

In [ ]:
```python
import numpy as np

# 创建一个向量
v1 = np.array([1, 2])

# 向量的数量乘法
v2 = 2 * v1

print(v2)
```

```
[2 4]
```

练习：通过将所有向量各自的x坐标和y坐标相加，可以实现任意数量的向量相加。例如，向量和(1, 2) + (2, 4) + (3, 6) + (4, 8)有分量1 + 2 + 3 + 4 = 10与分量2 + 4 + 6 + 8 = 20，结果为(10, 20)。实现新的add函数，接收任意多个向量作为参数。

In [ ]:
```python
import numpy as np

# 使用numpy的广播机制和numpy数组实现（见下附录）
# numpy广播（计算）：广播是两数组形同、对应位置上的元素做某种运算。
#
def vector_addition(*vectors):
    # 将所有向量转换为 NumPy 数组
    np_vectors = [np.array(vector) for vector in vectors]

    # 使用 np.sum 进行向量加法，指定 axis=0 表示按列进行求和
    result = np.sum(np_vectors, axis=0)

    return result

# 示例
vector1 = [1, 2, 3]
vector2 = [4, 5, 6]
vector3 = [7, 8, 9]

result = vector_addition(vector1, vector2, vector3)
print("向量加法结果:", result)
```

```
向量加法结果: [12 15 18]
```

附录：numpy broadcast（numpy的广播机制）

1. numpy广播（计算）

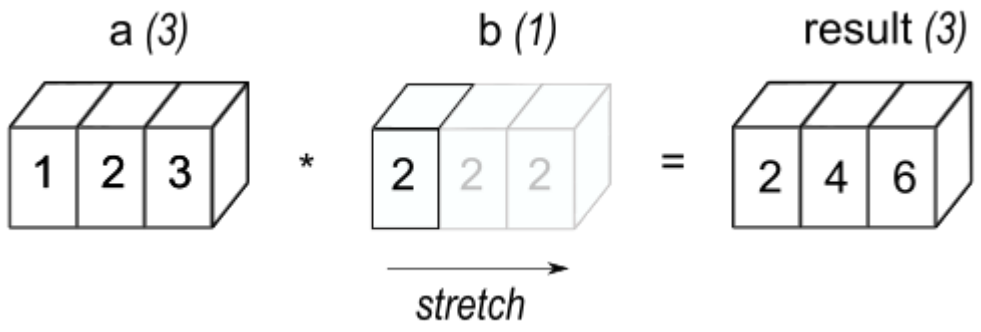
   广播是两数组形同、对应位置上的元素做某种运算。

2. numpy广播机制

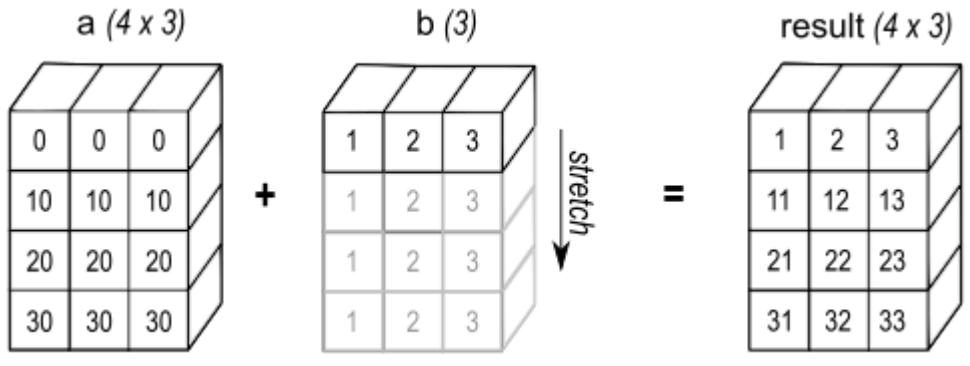
   广播机制是Numpy让两个不同shape的数组能够做一些运算，需要对参与运算的两个数组做一些处理或者说扩展，最终是参与运算的两个数组的shape一样，然后广播计算(对应位置数据进行某运算)得到结果。

   广播时需要对两个数组做广播机制处理，不是所有情况下两个数组都能进行广播机制的处理。要求两数组需满足广播兼容，具体规则是，比较两个数组的shape，从shape的尾部开始一一比对。

   A. 如果两个数组的维度相同，对应位置上轴的长度相同或其中一个的轴长度为1，广播兼容，可在轴长度为1的轴上进行广播机制处理。如下图所示：



B. 如果两个数组的维度不同，那么给低维度的数组前扩展提升一维，扩展维的轴长度为1,然后在扩展出的维上进行广播机制处理。如下图所示：



练习：实现函数translate(translation, vectors)，接收一个平移向量和一个向量列表，返回一个根据平移向量平移后的向量列表。例如，对于translate((1,1), [(0,0), (0,1,), (-3,-3)])，它应该返回[(1,1), (1,2), (-2, -2)]。

In [ ]:
```python
import numpy as np

def translate(translation, v):
    # 将向量转换为 NumPy 数组
    np_translation = np.array(translation)
    np_v = np.array(v)

    # 使用 NumPy 数组的广播机制进行向量加法
    result = np_translation + np_v

    return result

# 示例
translation = [1, 1]
v = [(0, 0), (0, 1), (-3, -3)]

result = translate(translation, v)
print("平移结果:\n", result)
```

```
平移结果:
 [[ 1  1]
 [ 1  2]
 [-2 -2]]
```