In [1]:

```
from typing import List
```

In [2]:

```
Vector = List[float]
```

In [3]:

```
height_weight_age = [70,  # Inches
                     170, # Pounds
                     40]  # Years
```

In [4]:

```
grades = [95,   # Exam 1
          80,   # Exam 2
          75,   # Exam 3
          62]   # Exam 4
```

In [5]:

```
def add(v: Vector, w: Vector) -> Vector:
    """Adds corresponding elements"""
    assert len(v) == len(w), "Vectors must be the same length"

    return [v_i + w_i for v_i, w_i in zip(v, w)]
```

In [6]:

```
assert add([1, 2, 3], [4, 5, 6]) == [5, 7, 9]
```

In [7]:

```
def subtract(v: Vector, w: Vector) -> Vector:
    """Subtracts corresponding elements"""
    assert len(v) == len(w), "Vectors must be the same length"

    return [v_i - w_i for v_i, w_i in zip(v, w)]
```

In [8]:

```
assert subtract([5, 7, 9], [4, 5, 6]) == [1, 2, 3]
```

In [9]:

```
def vector_sum(vectors: List[Vector]) -> Vector:
    """Sums all corresponding elements"""
    # Check That Vectors Are Not Empty
    assert vectors, "No vectors provided!"

    # Check That the Vectors Are All the Same Size
    num_elements = len(vectors[0])
    assert all(len(v) == num_elements for v in vectors), "Different sizes!"

    # The i-th Element of the Result Is the Sum of Every Vector[i]
    return [sum(vector[i] for vector in vectors)
            for i in range(num_elements)]
```

In [10]:

```
assert vector_sum([[1, 2], [3, 4], [5, 6], [7, 8]]) == [16, 20]
```

In [11]:

```
def scalar_multiply(c: float, v: Vector) -> Vector:
    """Multiplies every element by c"""
    return [c * v_i for v_i in v]
```

In [12]:

```
assert scalar_multiply(2, [1, 2, 3]) == [2, 4, 6]
```

In [13]:

```
def vector_mean(vectors: List[Vector]) -> Vector:
    """Computes the element-wise average"""
    n = len(vectors)
    return scalar_multiply(1 / n, vector_sum(vectors))
```

In [14]:

```
assert vector_mean([[1, 2], [3, 4], [5, 6]]) == [3, 4]
```

In [15]:

```
def dot(v: Vector, w: Vector) -> float:
    """Computes v_1 * w_1 + ... + v_n * w_n"""
    assert len(v) == len(w), "Vectors must be same length"

    return sum(v_i * w_i for v_i, w_i in zip(v, w))
```

In [16]:

```
assert dot([1, 2, 3], [4, 5, 6]) == 32  # 1 * 4 + 2 * 5 + 3 * 6
```

In [17]:

```
def sum_of_squares(v: Vector) -> float:
    """Returns v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)
```

In [18]:

```
assert sum_of_squares([1, 2, 3]) == 14  # 1 * 1 + 2 * 2 + 3 * 3
```

In [19]:

```
import math
```

In [20]:

```
def magnitude(v: Vector) -> float:
    """Returns the magnitude (or length) of v"""
    return math.sqrt(sum_of_squares(v))   # math.sqrt is the square root function
```

In [21]:

```
assert magnitude([3, 4]) == 5
```

In [22]:

```
def squared_distance(v: Vector, w: Vector) -> float:
    """Computes (v_1 - w_1) ** 2 + ... + (v_n - w_n) ** 2"""
    return sum_of_squares(subtract(v, w))
```

In [23]:

```
def distance(v: Vector, w: Vector) -> float:
    """Computes the distance between v and w"""
    return math.sqrt(squared_distance(v, w))
```

In [24]:

```
def distance(v: Vector, w: Vector) -> float:  # Type: ignore
    return magnitude(subtract(v, w))
```

In [25]:

```
# Another Type Alias
Matrix = List[List[float]]

A = [[1, 2, 3],  # A has 2 rows and 3 columns
     [4, 5, 6]]

B = [[1, 2],     # B has 3 rows and 2 columns
     [3, 4],
     [5, 6]]
```

In [26]:

```
from typing import Tuple
```

In [27]:

```
def shape(A: Matrix) -> Tuple[int, int]:
    """Returns (# of rows of A, # of columns of A)"""
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0   # Number of elements in first row
    return num_rows, num_cols
```

In [28]:

```
assert shape([[1, 2, 3], [4, 5, 6]]) == (2, 3)  # 2 rows, 3 columns
```

In [29]:

```
def get_row(A: Matrix, i: int) -> Vector:
    """Returns the i-th row of A (as a Vector)"""
    return A[i]             # A[i] is already the i-th row
```

In [30]:

```
def get_column(A: Matrix, j: int) -> Vector:
    """Returns the j-th column of A (as a Vector)"""
    return [A_i[j]          # j-th element of row A_i
            for A_i in A]   # for each row A_i
```

In [31]:

```
from typing import Callable
```

In [32]:

```
def make_matrix(num_rows: int,
                num_cols: int,
                entry_fn: Callable[[int, int], float]) -> Matrix:
    """
    Returns a num_rows x num_cols matrix whose (i, j)-th entry is entry_fn(i, j)
    """
    return [[entry_fn(i, j)             # Given i, create a list
             for j in range(num_cols)]  # [entry_fn(i, 0), ... ]
            for i in range(num_rows)]   # Create one list for each i
```

In [33]:

```
def identity_matrix(n: int) -> Matrix:
    """Returns the n x n identity matrix"""
    return make_matrix(n, n, lambda i, j: 1 if i == j else 0)
```

In [41]:

```
assert identity_matrix(5) == [[1, 0, 0, 0, 0],
                              [0, 1, 0, 0, 0],
                              [0, 0, 1, 0, 0],
                              [0, 0, 0, 1, 0],
                              [0, 0, 0, 0, 1]]
```

In [42]:

```
data = [[70, 170, 40],
        [65, 120, 26],
        [77, 250, 19]]
```

In [43]:

```
friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
               (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

In [44]:

```
#            User 0  1  2  3  4  5  6  7  8  9

friend_matrix = [[0, 1, 1, 0, 0, 0, 0, 0, 0, 0],  # User 0
                 [1, 0, 1, 1, 0, 0, 0, 0, 0, 0],  # User 1
                 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],  # User 2
                 [0, 1, 1, 0, 1, 0, 0, 0, 0, 0],  # User 3
                 [0, 0, 0, 1, 0, 1, 0, 0, 0, 0],  # User 4
                 [0, 0, 0, 0, 1, 0, 1, 1, 0, 0],  # User 5
                 [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],  # User 6
                 [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],  # User 7
                 [0, 0, 0, 0, 0, 0, 1, 1, 0, 1],  # User 8
                 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]]  # User 9
```

In [45]:

```
assert friend_matrix[0][2] == 1, "0 and 2 are friends"
```

In [46]:

```
assert friend_matrix[0][8] == 0, "0 and 8 are not friends"
```

In [47]:

```
# Only Need to Look at One Row
friends_of_five = [i
                   for i, is_friend in enumerate(friend_matrix[5])
                   if is_friend]
```