A $D$-vector over $\mathbb{F}$ is a function with domain $D$ and co-domain $\mathbb{F}$. $\mathbb{F}$ must be a field.

The set of such vectors is written $\mathbb{F}^D$ (recall from The Function).

An $n$-vector over $\mathbb{F}$ is a function with domain $\{0, 1, 2, \ldots, n - 1\}$ and co-domain $\mathbb{F}$. Can also represent as an $n$-element list.
Vector algebraic properties

Addition

- **Addition is associative**: \((u + v) + w = u + (v + w)\)
- **Addition is commutative**: \(u + v = v + u\)

Scalar-vector multiplication

- **Scalar-vector multiplication is associative**: \((\alpha \beta) v = \alpha (\beta v)\)

Both addition and scalar-vector multiplication

- **Scalar-vector multiplication distributes over addition**: \(\alpha (u + v) = \alpha u + \alpha v\)

Dot-product

- **Dot-product is commutative**: \(u \cdot v = v \cdot u\)
- **Dot-product is homogeneous**: \((\alpha u) \cdot v = \alpha (u \cdot v)\)
- **Dot-product distributes over addition**: \(u \cdot (v + w) = u \cdot v + u \cdot w\)
The vecutil module

The procedures `zero_vec(D)` and `list2vec(L)` are defined in the file `vecutil.py`, which we provide.
Solving a triangular system of linear equations

How to find solution to this linear system?

\[
\begin{align*}
[1, 0.5, -2, 4] \cdot x &= -8 \\
[0, 3, 3, 2] \cdot x &= 3 \\
[0, 0, 1, 5] \cdot x &= -4 \\
[0, 0, 0, 2] \cdot x &= 6
\end{align*}
\]

Write \( x = [x_1, x_2, x_3, x_4] \).

System becomes

\[
\begin{align*}
1x_1 + 0.5x_2 - 2x_3 + 4x_4 &= -8 \\
3x_2 + 3x_3 + 2x_4 &= 3 \\
x_3 + 5x_4 &= -4 \\
2x_4 &= 6
\end{align*}
\]
Solving a triangular system of linear equations: Backward substitution

\[
\begin{align*}
1x_1 &+ 0.5x_2 - 2x_3 + 4x_4 = -8 \\
3x_2 &+ 3x_3 + 2x_4 = 3 \\
1x_3 &+ 5x_4 = -4 \\
2x_4 & = 6
\end{align*}
\]

Solution strategy:

- Solve for \( x_4 \) using fourth equation.
- Plug value for \( x_4 \) into third equations and solve for \( x_3 \).
- Plug values for \( x_4 \) and \( x_3 \) into second equation and solve for \( x_2 \).
- Plug values for \( x_4, x_3, x_2 \) into first equation and solve for \( x_1 \).
Solving a triangular system of linear equations: Backward substitution

\[
\begin{align*}
1x_1 + 0.5x_2 - 2x_3 + 4x_4 &= -8 \\
3x_2 + 3x_3 + 2x_4 &= 3 \\
1x_3 + 5x_4 &= -4 \\
2x_4 &= 6
\end{align*}
\]

2x_4 = 6

so \( x_4 = 6/2 = 3 \)

1x_3 = -4 - 5x_4 = -4 - 5(3) = -19

so \( x_3 = -19/1 = -19 \)

3x_2 = 3 - 3x_3 - 2x_4 = 3 - 2(3) - 3(-19) = 54

so \( x_2 = 54/3 = 18 \)

1x_1 = -8 - 0.5x_2 + 2x_3 - 4x_4 = -8 - 4(3) + 2(-19) - 0.5(18) = -67

so \( x_1 = -67/1 = -67 \)
Use Back Substitution to solve the following triangular system of linear equations.

\[\begin{align*}
2x_1 &+ 2x_2 - 6x_3 = 0 \\
-5x_2 &+ 4x_3 = 7 \\
2x_3 &= 1
\end{align*}\]
Solving a triangular system of linear equations: Backward substitution

Hack to implement backward substitution using vectors:

- Initialize vector $x$ to zero vector.
- Procedure will populate $x$ entry by entry.
- When it is time to populate $x_i$, entries $x_{i+1}, x_{i+2}, \ldots, x_n$ will be populated, and other entries will be zero.
- Therefore can use dot-product:
  - Suppose you are computing $x_2$ using $[0, 3, 3, 2] \cdot [x_1, x_2, x_3, x_4] = 3$
  - So far, vector $x = [x_1, x_2, x_3, x_4] = [0, 0, -19, 3]$.
  - $x_2 := (3 - ([0, 3, 3, 2] \cdot x)) / 3$

```python
def triangular_solve(rowlist, b):
    x = zero_vec(rowlist[0].D)
    for i in reversed(range(len(rowlist))):
        x[i] = (b[i] - rowlist[i] * x) / rowlist[i][i]
    return x
```
Solving a triangular system of linear equations: Backward substitution

```python
def triangular_solve(rowlist, b):
    x = zero_vec(rowlist[0].D)
    for i in reversed(range(len(rowlist))):
        x[i] = (b[i] - rowlist[i] * x)/rowlist[i][i]
    return x
```

**Observations:**

- If `rowlist[i][i]` is zero, procedure will raise `ZeroDivisionError`.
- If this never happens, solution found is the *only* solution to the system.
Solving a triangular system of linear equations: Backward substitution

```python
def triangular_solve(rowlist, b):
    x = zero_vec(rowlist[0].D)
    for i in reversed(range(len(rowlist))):
        x[i] = (b[i] - rowlist[i] * x)/rowlist[i][i]
    return x
```

Our code only works when vectors in rowlist have domain $D = \{0, 1, 2, \ldots, n - 1\}$.

For arbitrary domains, need to specify an ordering for which system is “triangular”:

```python
def triangular_solve(rowlist, label_list, b):
    x = zero_vec(set(label_list))
    for r in reversed(range(len(rowlist))):
        c = label_list[r]
        x[c] = (b[r] - x*rowlist[r])/rowlist[r][c]
    return x
```