In [ ]:
```
# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'
```

Prerequisite Material

Following the notation of *Spacecraft Dynamics and Control: An Introduction*, First Edition. A.H.J. de Ruiter, C.J. Damaren and J.R. Forbes. 2013 John Wiley & Sons, Ltd.

Vectors

A vector is a three-dimensional quantity, denoted $\vec{r}$, with *magnitude* ($|\vec{r}|$) and *direction* that satisfies the following rules:

Addition

$$\begin{aligned}
(\vec{a}+\vec{b})+\vec{c} &= \vec{a}+(\vec{b}+\vec{c}) && \text{associative} && (1)\\
\vec{a}+\vec{b} &= \vec{b}+\vec{a} && \text{commutative} && (2)\\
\vec{a}+\vec{0} &= \vec{a} && \text{identity} && (3)\\
\vec{a}+(\vec{-a}) &= \vec{0} && \text{inverse} && (4)
\end{aligned}$$

Scalar Multiplication

$$\begin{aligned}
a(b\vec{c}) &= (ab)\vec{c} && (5)\\
(a+b)\vec{c} &= a\vec{c}+b\vec{c} && (6)\\
a(\vec{b}+\vec{c}) &= a\vec{b}+a\vec{c} && (7)\\
1\vec{a} &= \vec{a} && (8)\\
0\vec{a} &= \vec{0} && (9)
\end{aligned}$$

Scalar (Dot) Product

Defined as

$$\vec{a}\cdot\vec{b} \triangleq |\vec{a}||\vec{b}|\cos\theta \tag{10}$$

θ is the angle between $\vec{a}$ and $\vec{b}$.

The dot product is the the projection of $\vec{a}$ onto $\vec{b}$ multiplied by $|\vec{b}|$.

The dot product is commutative $\vec{a}\cdot\vec{b}=\vec{b}\cdot\vec{a}$.

The dot product is also distributive $(\vec{a}+\vec{b})\cdot\vec{c}=\vec{a}\cdot\vec{c}+\vec{b}\cdot\vec{c}$

Additional properties:

$$\vec{a} \cdot \vec{a} = |\vec{a}|^2 \geq 0 \tag{11}$$
$$\vec{a} \cdot \vec{a} = 0 \Leftrightarrow \vec{a} = \vec{0} \tag{12}$$
$$\vec{a} \cdot (c\vec{b}) = c\vec{a} \cdot \vec{b} \tag{13}$$
$$\vec{a} \cdot \vec{b} = 0 \Leftrightarrow \vec{a} \perp \vec{b} \text{ or } \vec{a} = \vec{0} \text{ or } \vec{b} = \vec{0} \tag{14}$$

Vector Cross Product

Defined as

$$\vec{c} = \vec{a} \times \vec{b} \tag{15}$$

where

$$|\vec{c}| = |\vec{a}||\vec{b}| \sin\theta \tag{16}$$

θ is the angle between $\vec{a}$ and $\vec{b}$. The direction of $\vec{c}$ is perpendicular to the plane containing $\vec{a}$ and $\vec{b}$ according to the right-hand rule.

Changing the order reverses the direction of the cross-product

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a} \tag{17}$$

Additional properties:

$$(\vec{a} + \vec{b}) \times \vec{c} = \vec{a} \times \vec{c} + \vec{b} \times \vec{c} \quad \text{distributive} \tag{18}$$
$$\vec{a} \times \vec{a} = \vec{0} \tag{19}$$
$$(a\vec{b}) \times \vec{c} = a(\vec{b} \times \vec{c}) \tag{20}$$

Modeling

We will use computer simulation and modeling as the paradigm for learning the material in this course. We will develop a simulation of orbit trajectories using the Anaconda distribution of the Python programming language. Our work will be structured like Allen Downey's work in his book Modeling and Simulation in Python. Professor Downey's code and notebooks can be found in his ModSimPy GitHub repository. Many of the functions used here are from Allen Downey's ModSimPy library.

We will start by creating some functions for vector and matrix operations.

First, create a vector using the pandas Series data structure. pandas is the Python Data Analysis Library. We will use many pandas data objects in this course.

In [ ]:
```
import pandas as pd
```

Vectors

Vectors are defined using the pandas Series data structure.

In [ ]:
```
# Define a Vector using the pandas data structure called Series
def Vector(x, y, z=None):
    """
    create a Vector, 2D or 3D
    """
    if z is None:
        return pd.Series(dict(x=x, y=y))
    else:
        return pd.Series(dict(x=x, y=y, z=z))
```

In [ ]:
```
a = Vector(1,2,3)
```

We can access the individual components of the vector using pandas Series dot notation.

In [ ]:
```
a.y
```

NumPy is the Python package for operating on multi-dimensional numerical arrays. It contains many functions that will allow us to work with vectors and matricies.

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

Vector Magnitude

The vector magnitude function, `vector_mag()`, uses the NumPy dot and sqrt functions.

In [ ]:
```
# Define a function to compute the magnitude of a vector
def vector_mag(v):
    """
    magnitude of a vector
    """
    return np.sqrt(np.dot(v,v))
```

In [ ]:
```
vector_mag(a)
```

Dot Product

We define our own `vector_dot()` function that uses the NumPy dot function.

In [ ]:
```
# Define a function to compute the dot product of two vectors
def vector_dot(v, w):
    """
    dot product of v and w
    """
    return np.dot(v, w)
```

In [ ]:
```
b = Vector(4,5,6)

vector_dot(a,b)
```

Unit Vector

A unit vector of $\vec{a}$ is a vector pointing in the same direction as $\vec{a}$ with a magnitude of 1. A unit vector is often denoted as $\hat{a}$. The function `vector_hat()` computes the unit vector.

In [ ]:
```
# Define a function to compute a unit vector in the direction of v
def vector_hat(v):
    """
    unit vector in the direction of v
    """
    # check if the magnitude of the Quantity is 0
    mag = vector_mag(v)
    if mag == 0:
        return v
    else:
        return v / mag
```

In [ ]:
```
a_hat = vector_hat(a)
```

Check the magnitude of $\vec{a}$, i.e. $|\vec{a}|$

In [ ]:
```
vector_mag(a_hat)
```

Vector Cross Product

The `vector_cross()` funtion puts the result into the Vector data type if it is 3-D. `vector_cross()` uses the Numpy cross function.

In [ ]:
```
# Define a function to compute the cross product of two vectors
def vector_cross(v, w):
    """
    cross product of v and w

    returns: number or Quantity for 2-D, Vector for 3-D
    """
    result = np.cross(v, w)

    if len(v) == 3:
        return Vector(*result)
    else:
        return result
```

In [ ]:
```
c = vector_cross(a, b)
```

Matricies

Matricies are represented using NumPy arrays.

```
In [ ]: A = np.array([[1, 2], [3, 4]])
```

```
In [ ]: B = np.array([[4, 1],
                      [2, 2]])
```

Matrix Addition and Subtraction

Matrix addtion and subtraction are performed element-wise.

```
In [ ]: C = A + B
```

Individual elements can be referenced using Python bracket notation. Array indexes are zero based.

```
In [ ]: A[0,0]
```

Matrix Multiplication

We will use the NumPy function `np.matmul()` or the `@` operator for matrix multiplication. The inner dimentions of the two matricies must be the same. The resulting matrix will have the outer dimensions such as (n,k),(k,m)->(n,m).

```
In [ ]: I = np.array([[1, 0, 0],
                      [0, 1, 0],
                      [0, 0, 1]])
        D = np.array([[1, 2, 3],
                      [4, 5, 6],
                      [7, 8, 9]])
        np.matmul(I, D)
```

```
In [ ]: E = D @ I
```

Exercises

Exercise: Numerically test the funtions we have defined above using the well-known vector identites listed below.

$$\vec{a} \cdot \vec{a} = |\vec{a}|^2$$

$$\vec{a} \cdot (\vec{b} \times \vec{c}) = (\vec{a} \times \vec{b}) \cdot \vec{c} \qquad \text{interchange of dot and cross, tri}$$

$$\vec{a} \times (\vec{b} \times \vec{c}) = \vec{b}(\vec{a} \cdot \vec{c}) - \vec{c}(\vec{a} \cdot \vec{b}) \qquad \text{double cross product or ba}$$

$$\vec{a} \times (\vec{b} \times \vec{c}) + \vec{c} \times (\vec{a} \times \vec{b}) + \vec{b} \times (\vec{c} \times \vec{a}) = \vec{0} \qquad \text{Jacobi Identity}$$

$$(\vec{a} \times \vec{b}) \cdot (\vec{c} \times \vec{d}) = (\vec{a} \cdot \vec{c})(\vec{b} \cdot \vec{d}) - (\vec{a} \cdot \vec{d})(\vec{b} \cdot \vec{c}) \qquad \text{scalar product of two cros}$$

Specifically, does the right-hand side equal the left-hand side?

In [ ]:
```
# Solution goes here
a = Vector(2,3,4)
lhs = vector_dot(a,a)
rhs = vector_mag(a)**2
print(lhs==rhs,lhs,rhs)
```

In [ ]:
```
# Solution goes here
```

In [ ]:
```
# Solution goes here
```

In [ ]:
```
# Solution goes here
```

In [ ]:
```
# Solution goes here
```