Using a Cartesian 3D Vector Class

In [1]:

```
from math import pi

from skvectors import create_class_Cartesian_3D_Vector
```

In [2]:

```
# Create a 3-dimensional cartesian vector class

CVC3D = create_class_Cartesian_3D_Vector('CVC3D', 'xyz')

# Explicit alternative:
# CVC3D = \
#     create_class_Cartesian_3D_Vector(
#         name = 'CVC3D',
#         component_names = [ 'x', 'y', 'z' ],
#         brackets = [ '<', '>' ],
#         sep = ', ',
#         cnull = 0,
#         cunit = 1,
#         functions = None
#     )
```

In [3]:

```
# Create a vector from the cross product of a vector and another
u = CVC3D(-1, 2, 3)
v = CVC3D(4, 5, 0)
u.cross(v)
```

Out[3]:

```
CVC3D(x=-15, y=12, z=-13)
```

In [4]:
```
# Calculate the sine (from cnull to +cunit) of the smallest angle between two vectors
u = CVC3D(3, 0, 0)
v = CVC3D(1, 0, -1)
u.sin(v)  # 2**-0.5
```
Out[4]:
```
0.7071067811865475
```

In [5]:
```
# Create a vector from a vector rotated around the basis vector x by an angle in radians
u = CVC3D(1, -2, 3)
u.rotate_x(angle=-pi/2), u.rotate_x(pi/2)
```
Out[5]:
```
(CVC3D(x=1, y=3.0, z=2.0), CVC3D(x=1, y=-3.0, z=-1.9999999999999998))
```

In [6]:
```
# Create a vector from a vector rotated around the basis vector y by an angle in radians
u = CVC3D(1, -2, 3)
u.rotate_y(angle=-pi/2), u.rotate_y(pi/2)
```
Out[6]:
```
(CVC3D(x=-3.0, y=-2, z=1.0000000000000002),
 CVC3D(x=3.0, y=-2, z=-0.9999999999999998))
```

In [7]:
```
# Create a vector from a vector rotated around the basis vector z by an angle in radians
u = CVC3D(1, -2, 3)
u.rotate_z(angle=-pi/2), u.rotate_z(pi/2)
```
Out[7]:
```
(CVC3D(x=-2.0, y=-1.0000000000000002, z=3),
 CVC3D(x=2.0, y=0.9999999999999999, z=3))
```

In [8]:
```
# Create a vector from a vector rotated around another by an angle in radians
u = CVC3D(-1, -1, 0)
v = CVC3D(3, 0, -3)
u.axis_rotate(v, angle=pi)
```
Out[8]:
```
CVC3D(x=1.3544899930148195e-16, y=1.0000000000000004, z=1.0000000000000002)
```

In [9]:
```
# Create a vector from a vector rotated around another by an angle in radians
u = CVC3D(-3, -3, 0)
v = CVC3D(5, 5, -5)
u.axis_rotate(v, pi)
```
Out[9]:
```
CVC3D(x=-1.0000000000000007, y=-1.0000000000000002, z=3.999999999999999)
```

In [10]:

```
# Create a vector from a vector reoriented from one direction to another direction.
# NB: The two direction vectors must not point in opposite directions
u = CVC3D(9, 12, 0)
v = CVC3D(1, 0, 1)
w = CVC3D(0, 2, 2)
u.reorient(v, w)
```

Out[10]:

```
CVC3D(x=2.0000000000000018, y=14.0, z=-5.000000000000001)
```

In [11]:

```
# Check if a vector is parallel with another
u = CVC3D(1, 0, -3)
v = CVC3D(-2, 0, 6)
u.are_parallel(v)
```

Out[11]:

```
True
```

In [12]:

```
# Check if a vector is parallel with another
u = CVC3D(1, 0, -3)
v = CVC3D(-2, 0, -6)
u.are_parallel(v)
```

Out[12]:

```
False
```

In [13]:

```
# NB: All vectors are parallel to the zero vector
u = CVC3D(1, 0, -3)
v = CVC3D(0, 0, 0)
u.are_parallel(v)
```

Out[13]:

```
True
```

In [14]:

```
# NB: The zero vector is parallel to all vectors
u = CVC3D(0, 0, 0)
v = CVC3D(1, 0, -3)
u.are_parallel(v)
```

Out[14]:

```
True
```

In [15]:
```
# Calculate the scalar triple product of a vector and two others
u = CVC3D(-1, 2, 3)
v = CVC3D(-2, -2, 2)
w = CVC3D(4, 0, 5)
u.stp(v, w)
```

Out[15]:
```
70
```

In [16]:
```
# Create a vector from the vector triple product of a vector and two others
u = CVC3D(1, 2, 3)
v = CVC3D(2, 3, 1)
w = CVC3D(1, 1, 2)
u.vtp(v, w)
```

Out[16]:
```
CVC3D(x=7, y=16, z=-13)
```

In [17]:
```
# Create a vector from polar coordinates
# The angles are in radians
u = CVC3D.from_polar(radius=10, azimuth=pi/2, inclination=pi/4)
u  # = 10 * CVC3D(x=0, y=2**-0.5, z=2**-0.5)
```

Out[17]:
```
CVC3D(x=4.329780281177467e-16, y=7.0710678118654755, z=7.071067811865475)
```

In [18]:
```
# Create vectors from polar coordinates
CVC3D.from_polar(1, 0, 0), CVC3D.from_polar(1, pi/2, 0), CVC3D.from_polar(1, pi, 0)
```

Out[18]:
```
(CVC3D(x=1.0, y=0.0, z=0.0),
 CVC3D(x=6.123233995736766e-17, y=1.0, z=0.0),
 CVC3D(x=-1.0, y=1.2246467991473532e-16, z=0.0))
```

In [19]:
```
# Create vectors from polar coordinates
CVC3D.from_polar(8, 0, 0), CVC3D.from_polar(8, 0, pi/2), CVC3D.from_polar(8, 0, pi)
```

Out[19]:
```
(CVC3D(x=8.0, y=0.0, z=0.0),
 CVC3D(x=4.898587196589413e-16, y=0.0, z=8.0),
 CVC3D(x=-8.0, y=-0.0, z=9.797174393178826e-16))
```

In [20]:

```python
# Calculate the polar coordinates for a vector and return them in a dictionary
# The angles are in radians
u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
u.polar_as_dict()  # radius = 10.0, azimuth = pi/2, inclination = pi/4
```

Out[20]:

```
{'azimuth': 1.5707963267948966,
 'inclination': 0.7853981633974483,
 'radius': 10.0}
```

In [21]:

```python
# Calculate the radius of a vector converted to polar coordinates
u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
u.radius
```

Out[21]:

```
10.0
```

In [22]:

```python
# Calculate the azimuth of a vector converted to polar coordinates
u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
u.azimuth  # = pi/2 radians
```

Out[22]:

```
1.5707963267948966
```

In [23]:

```python
# Calculate the inclination of a vector converted to polar coordinates
u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
u.inclination  # = pi/4 radians
```

Out[23]:

```
0.7853981633974483
```

In [ ]:

```python

```