

# Using a Cartesian 3D Vector Class

Copyright (c) 2019 Tor Olav Kristensen, <http://subcube.com> (<http://subcube.com>)

<https://github.com/t-o-k/scikit-vectors> (<https://github.com/t-o-k/scikit-vectors>)

Use of this source code is governed by a BSD-license that can be found in the LICENSE file.

```
In [1]: 1 from math import pi
        2
        3 from skvectors import create_class_Cartesian_3D_Vector
```

```
In [2]: 1 # Create a 3-dimensional cartesian vector class
        2
        3 CVC3D = create_class_Cartesian_3D_Vector('CVC3D', 'xyz')
        4
        5 # Explicit alternative:
        6 # CVC3D = \
        7 #     create_class_Cartesian_3D_Vector(
        8 #         name = 'CVC3D',
        9 #         component_names = [ 'x', 'y', 'z' ],
       10 #         brackets = [ '<', '>' ],
       11 #         sep = ', ',
       12 #         cnull = 0,
       13 #         cunit = 1,
       14 #         functions = None
       15 #     )
```

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

```
Out[3]: CVC3D(x=-15, y=12, z=-13)
```

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

Out[4]: 0.7071067811865475

```
In [5]: 1 # Create a vector from a vector rotated around the basis vector x by an angle in radians
        2 u = CVC3D(1, -2, 3)
        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]: 1 # Create a vector from a vector rotated around the basis vector y by an angle in radians
        2 u = CVC3D(1, -2, 3)
        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]: 1 # Create a vector from a vector rotated around the basis vector z by an angle in radians
        2 u = CVC3D(1, -2, 3)
        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]: 1 # Create a vector from a vector rotated around another by an angle in radians
        2 u = CVC3D(-1, -1, 0)
        3 v = CVC3D(3, 0, -3)
        4 u.axis_rotate(v, angle=pi)
```

Out[8]: CVC3D(x=1.3544899930148195e-16, y=1.0000000000000004, z=1.0000000000000002)

```
In [9]: 1 # Create a vector from a vector rotated around another by an angle in radians
        2 u = CVC3D(-3, -3, 0)
        3 v = CVC3D(5, 5, -5)
        4 u.axis_rotate(v, pi)
```

Out[9]: CVC3D(x=-1.0000000000000007, y=-1.0000000000000002, z=3.9999999999999999)

```
In [10]: 1 # Create a vector from a vector reoriented from one direction to another direction
2 # NB: The two direction vectors must not have opposite directions
3 u = CVC3D(9, 12, 0)
4 v = CVC3D(1, 0, 1)
5 w = CVC3D(0, 2, 2)
6 u.reorient(v, w)
```

Out[10]: CVC3D(x=2.0000000000000018, y=14.0, z=-5.000000000000001)

```
In [11]: 1 # Check if a vector is parallel to another
2 u = CVC3D(1, 0, -3)
3 v = CVC3D(-2, 0, 6)
4 u.are_parallel(v)
```

Out[11]: True

```
In [12]: 1 # Check if a vector is parallel to another
2 u = CVC3D(1, 0, -3)
3 v = CVC3D(-2, 0, -6)
4 u.are_parallel(v)
```

Out[12]: False

```
In [13]: 1 # NB: All vectors are parallel to the zero vector
2 u = CVC3D(1, 0, -3)
3 v = CVC3D(0, 0, 0)
4 u.are_parallel(v)
```

Out[13]: True

```
In [14]: 1 # NB: The zero vector is parallel to all vectors
2 u = CVC3D(0, 0, 0)
3 v = CVC3D(1, 0, -3)
4 u.are_parallel(v)
```

Out[14]: True

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

Out[15]: 70

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

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

```
In [17]: 1 # Create a vector from polar coordinates
2 # The angles are in radians
3 u = CVC3D.from_polar(radius=10, azimuth=pi/2, inclination=pi/4)
4 u # x = 0, y = 10/2**0.5, z = 10/2**0.5
```

Out[17]: CVC3D(x=4.3297802811774667e-16, y=7.0710678118654755, z=7.071067811865475)

```
In [18]: 1 # Create vectors from polar coordinates
2 [
3     CVC3D.from_polar(radius=8, azimuth=0, inclination=angle)
4     for angle in [ -2/2*pi, -1/2*pi, 0/2*pi, 1/2*pi, 2/2*pi ]
5 ]
```

Out[18]: [CVC3D(x=-8.0, y=-0.0, z=-9.797174393178826e-16),  
CVC3D(x=4.898587196589413e-16, y=0.0, z=-8.0),  
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 [19]: 1 # Create vectors from polar coordinates
2 [
3     CVC3D.from_polar(radius=8, azimuth=angle, inclination=0)
4     for angle in [ -2/2*pi, -1/2*pi, 0/2*pi, 1/2*pi, 2/2*pi ]
5 ]
```

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

```
In [20]: 1 # Calculate the polar coordinates for a vector and return them in a dictionary
2 # The azimuth angle is in radians from -pi*cunit to +pi*cunit
3 # The inclination angle is in radians from -pi/2*cunit to +pi/2*cunit
4 u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
5 u.polar_as_dict() # radius = 10.0, azimuth = pi/2 radians, inclination = pi/4 radians
```

```
Out[20]: {'azimuth': 1.5707963267948966,
'inclination': 0.7853981633974483,
'radius': 10.0}
```

```
In [21]: 1 # Calculate the radius of a vector converted to polar coordinates
2 u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
3 u.radius
```

```
Out[21]: 10.0
```

```
In [22]: 1 # Calculate the azimuth of a vector converted to polar coordinates
2 u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
3 u.azimuth # = pi/2 radians
```

```
Out[22]: 1.5707963267948966
```

```
In [23]: 1 # Calculate the inclination of a vector converted to polar coordinates
2 u = 10 * CVC3D(0, 2**-0.5, 2**-0.5)
3 u.inclination # = pi/4 radians
```

```
Out[23]: 0.7853981633974483
```

```
In [ ]: 1
```

