

# Using a Cartesian 2D 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_2D_Vector
```

```
In [2]: 1 # Create a 2-dimensional cartesian vector class
        2
        3 CVC2D = create_class_Cartesian_2D_Vector('CVC2D', 'uv')
        4
        5 # Explicit alternative:
        6 # CVC2D = \
        7 #     create_class_Cartesian_2D_Vector(
        8 #         name = 'CVC2D',
        9 #         component_names = [ 'u', 'v' ],
       10 #         brackets = [ '<', '>' ],
       11 #         sep = ', ',
       12 #         cnull = 0,
       13 #         cunit = 1,
       14 #         functions = None
       15 #     )
```

```
In [3]: 1 # Create a vector that is perpendicular to a vector
        2 u = CVC2D(4, -3)
        3 u.perp()
```

```
Out[3]: CVC2D(u=3, v=4)
```

```
In [4]: 1 # NB: The zero vector is perpendicular to all vectors, including itself
        2 u = CVC2D(0, 0)
        3 u.perp()
```

Out[4]: CVC2D(u=0, v=0)

```
In [5]: 1 # Calculate the perp-dot product of a vector and another
        2 u = CVC2D(1, -2)
        3 v = CVC2D(3, 4)
        4 u.perp_dot(v)
```

Out[5]: 10

```
In [6]: 1 # Calculate the sine (from -cunit to +cunit) of the counterclockwise angle between a vector and another
        2 u = CVC2D(3, 0)
        3 v = CVC2D(1, -1)
        4 u.sin(v) # = -2**-.5
```

Out[6]: -0.7071067811865475

```
In [7]: 1 # Calculate the counterclockwise angle in radians (from -cunit*pi to +cunit*pi) between a vector and another
        2 u = CVC2D(1, 1)
        3 v = CVC2D(0, -1)
        4 u.angle(v) # = -3/4*pi radians
```

Out[7]: -2.356194490192345

```
In [8]: 1 # Calculate the counterclockwise angle in radians between a vector and another
        2 u = CVC2D(1, 1)
        3 v = CVC2D(-1, 0)
        4 u.angle(v) # = 3/4*pi radians
```

Out[8]: 2.356194490192345

```
In [9]: 1 # Create a vector by rotating a vector counterclockwise by an angle in radians
        2 u = CVC2D(1, 1)
        3 u.rotate(angle=3/2*pi)
```

Out[9]: CVC2D(u=0.9999999999999998, v=-1.0000000000000002)

```
In [10]: 1 # Create a vector by reorienting a vector from one direction to another direction
         2 # NB: The two direction vectors must not have opposite directions
         3 u = CVC2D(9, 12)
         4 v = CVC2D(1, 0)
         5 w = CVC2D(0, -2)
         6 u.reorient(v, w)
```

Out[10]: CVC2D(u=12.0, v=-9.0)

```
In [11]: 1 # Check if a vector is parallel to another
         2 u = CVC2D(1, 0)
         3 v = CVC2D(-2, 0)
         4 u.are_parallel(v)
```

Out[11]: True

```
In [12]: 1 # Check if a vector is parallel to another
         2 u = CVC2D(1, 1)
         3 v = CVC2D(-2, 0)
         4 u.are_parallel(v)
```

Out[12]: False

```
In [13]: 1 # NB: All vectors are parallel to the zero vector
         2 u = CVC2D(3, -4)
         3 v = CVC2D(0, 0)
         4 u.are_parallel(v)
```

Out[13]: True

```
In [14]: 1 # NB: The zero vector is parallel to all vectors
         2 u = CVC2D(0, 0)
         3 v = CVC2D(3, -4)
         4 u.are_parallel(v)
```

Out[14]: True

```
In [15]: 1 # NB: The zero vector is parallel to itself
         2 u = CVC2D(0, 0)
         3 u.are_parallel(u)
```

Out[15]: True

```
In [16]: 1 # Create a vector from polar coordinates
2 # The azimuth angle is in radians
3 CVC2D.from_polar(radius=2, azimuth=-pi/3) # u = 1.0, v = -3**0.5
```

```
Out[16]: CVC2D(u=1.0000000000000002, v=-1.7320508075688772)
```

```
In [17]: 1 # Create vectors from polar coordinates
2 [
3     CVC2D.from_polar(radius=1, azimuth=angle)
4     for angle in [ 0/2*pi, 1/2*pi, 2/2*pi, 3/2*pi ]
5 ]
```

```
Out[17]: [CVC2D(u=1.0, v=0.0),
CVC2D(u=6.123233995736766e-17, v=1.0),
CVC2D(u=-1.0, v=1.2246467991473532e-16),
CVC2D(u=-1.8369701987210297e-16, v=-1.0)]
```

```
In [18]: 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 u = CVC2D(1, -3**0.5)
4 u.polar_as_dict() # radius = 2.0, azimuth = -pi/3 radians
```

```
Out[18]: {'azimuth': -1.0471975511965976, 'radius': 1.9999999999999998}
```

```
In [19]: 1 # Calculate the radius of a vector converted to polar coordinates
2 u = CVC2D(1, -3**0.5)
3 u.radius
```

```
Out[19]: 1.9999999999999998
```

```
In [20]: 1 # Calculate the azimuth angle in radians of a vector converted to polar coordinates
2 u = CVC2D(1, -3**0.5)
3 u.azimuth
```

```
Out[20]: -1.0471975511965976
```

```
In [ ]: 1
```

