Using a Cartesian 2D Vector Class

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

In [1]:

```
from math import pi

from skvectors import create_class_Cartesian_2D_Vector
```

In [2]:

```
# Create a 2-dimensional cartesian vector class

CVC2D = create_class_Cartesian_2D_Vector('CVC2D', 'uv')

# Explicit alternative:
# CVC2D = \
#     create_class_Cartesian_2D_Vector(
#         name = 'CVC2D',
#         component_names = [ 'u', 'v' ],
#         brackets = [ '<', '>' ],
#         sep = ', ',
#         cnull = 0,
#         cunit = 1,
#         functions = None
#     )
```

In [3]:

```
# Create a vector that is perpendicular to a vector
u = CVC2D(4, -3)
u.perp()
```

Out[3]:

```
CVC2D(u=3, v=4)
```

In [4]:

```
# NB: The zero vector is perpendicular to all vectors, including itself
u = CVC2D(0, 0)
u.perp()
```

Out[4]:

```
CVC2D(u=0, v=0)
```

In [5]:

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

Out[5]:

```
10
```

In [6]:

```
# Calculate the sine (from -cunit to +cunit) of the counterclockwise angle between a vector and another
u = CVC2D(3, 0)
v = CVC2D(1, -1)
u.sin(v)  # = -2**-0.5
```

Out[6]:

```
-0.7071067811865475
```

In [7]:

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

Out[7]:

```
-2.356194490192345
```

In [8]:

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

Out[8]:

```
2.356194490192345
```

In [9]:

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

Out[9]:

```
CVC2D(u=0.9999999999999998, v=-1.0000000000000002)
```

In [10]:

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

Out[10]:

```
CVC2D(u=12.0, v=-9.0)
```

In [11]:

```
# Check if a vector is parallel to another
u = CVC2D(1, 0)
v = CVC2D(-2, 0)
u.are_parallel(v)
```

Out[11]:

```
True
```

In [12]:

```
# Check if a vector is parallel to another
u = CVC2D(1, 1)
v = CVC2D(-2, 0)
u.are_parallel(v)
```

Out[12]:

```
False
```

In [13]:

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

Out[13]:

```
True
```

In [14]:

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

Out[14]:

```
True
```

In [15]:

```
# NB: The zero vector is parallel to itself
u = CVC2D(0, 0)
u.are_parallel(u)
```

Out[15]:

```
True
```

In [16]:

```
# Create a vector from polar coordinates
# The azimuth angle is in radians
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]:

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

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]:

```
# Calculate the polar coordinates for a vector and return them in a dictionary
# The azimuth angle is in radians from -pi*cunit to +pi*cunit
u = CVC2D(1, -3**0.5)
u.polar_as_dict()  # radius = 2.0, azimuth = -pi/3 radians
```

Out[18]:

```
{'azimuth': -1.0471975511965976, 'radius': 1.9999999999999998}
```

In [19]:

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

Out[19]:

```
1.9999999999999998
```

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

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