# Using a Cartesian Vector Class

In [1]:
```python
from skvectors import create_class_Cartesian_Vector
```

In [2]:
```python
# Create a 4-dimensional cartesian vector class

CVC = create_class_Cartesian_Vector('CVC', 'ABCD')

# Explicit alternative:
# CVC = \
#     create_class_Cartesian_Vector(
#         name = 'CVC',
#         component_names = [ 'A', 'B', 'C', 'D' ],
#         brackets = [ '<', '>' ],
#         sep = ', ',
#         cnull = 0,
#         cunit = 1,
#         functions = None
#     )
```

In [3]:
```python
# Check if a vector is the zero vector (i.e. its length is equal to cnull)
u = CVC(0, 0, 0, 0)
u.is_zero_vector()
```

Out[3]: True

In [4]:
```python
# Check if a vector is the zero vector
u = CVC(0, 0, 1, 0)
u.is_zero_vector()
```

Out[4]: False

```
In [5]:   1  # Check if a vector is a unit vector (i.e. its length is equal to cunit)
          2  u = CVC(0.6, 0.0, -0.8, 0.0)
          3  u.is_unit_vector()

Out[5]:  True
```

```
In [6]:   1  # Check if a vector is a unit vector
          2  u = CVC(-0.1, 0.5, 0.0, 0.8)
          3  u.is_unit_vector()

Out[6]:  False
```

```
In [7]:   1  # Check if a vector is equal to another
          2  u = CVC(-3, -1, 4, 2)
          3  v = CVC(-3, 1, 4, 2)
          4  u == v

Out[7]:  False
```

```
In [8]:   1  # Check if a vector is not equal to another
          2  u = CVC(-3, -1, 4, 2)
          3  v = CVC(-3, 1, 4, 2)
          4  u != v

Out[8]:  True
```

```
In [9]:   1  # Check if a vector is orthogonal to another
          2  u = CVC(3, 2, -1, 4)
          3  v = CVC(0, 3, 6, 0)
          4  u.are_orthogonal(v)

Out[9]:  True
```

```
In [10]:  1  # Check if a vector is orthogonal to another
          2  u = CVC(3, 2, -1, 4)
          3  v = CVC(0, 3, 6, 4)
          4  u.are_orthogonal(v)

Out[10]:  False
```

```
In [11]:    1  # NB: All vectors are orthogonal to the zero vector
            2  u = CVC(3, 2, -1, 4)
            3  v = CVC(0, 0, 0, 0)
            4  u.are_orthogonal(v)
```

Out[11]:  True

```
In [12]:    1  # NB: The zero vector is orthogonal to all vectors
            2  u = CVC(0, 0, 0, 0)
            3  v = CVC(3, 2, -1, 4)
            4  u.are_orthogonal(v)
```

Out[12]:  True

```
In [13]:    1  # Check if the length of a vector is equal to the length of another
            2  u = CVC(-4, -2, 4, 1)
            3  v = CVC(5, 1, 1, -3)
            4  u.equal_lengths(v)
```

Out[13]:  False

```
In [14]:    1  # Check if the length of a vector is equal to the length of another
            2  (2 * CVC(-2, -1, 2, 0)).equal_lengths(CVC(15, 3, 3, -9) / 3)
```

Out[14]:  True

```
In [15]:    1  # Check if a vector is shorter than another
            2  u = CVC(1, 3, -5, 1)
            3  v = CVC(-4, 0, 4, -2)
            4  u.shorter(v)
```

Out[15]:  False

```
In [16]:    1  # Check if a vector is shorter than another
            2  u = CVC(5, -3, -1, -1)
            3  v = CVC(2, -5, 4, 2)
            4  u.shorter(v)
```

Out[16]:  True
```

```python
In [17]:   1  # Check if a vector is longer than another
           2  u = CVC(1, 3, -5, 1)
           3  v = CVC(-4, 0, 4, -2)
           4  u.longer(v)
```

Out[17]: False

```python
In [18]:   1  # Check if a vector is longer than another
           2  u = CVC(-2, 2, 4, 5)
           3  v = CVC(1, -5, -1, 3)
           4  u.longer(v)
```

Out[18]: True

```python
In [19]:   1  # Calculate the dot product of a vector and another
           2  u = CVC(-2, 1, -1, 3)
           3  v = CVC(2, 3, 0, -4)
           4  u.dot(v)
```

Out[19]: -13

```python
In [20]:   1  # Calculate the dot product of a vector and another
           2  (CVC(-4, -1, -3, 1) + 2).dot(CVC(3, 4, 1, -3) - 1)
```

Out[20]: -13

```python
In [21]:   1  # Calculate the length (magnitude) of a vector
           2  u = CVC(-2, -5, 4, 2)
           3  u.length()
```

Out[21]: 7.0

```python
In [22]:   1  # Calculate the length of a vector
           2  (CVC(3, -2, -3, 7) - CVC(2, -7, -2, 4)).length()
```

Out[22]: 6.0
```

```
In [23]:    1  # Calculate the distance between a position vector and another
            2  u = CVC(3, -2, -3, 7)
            3  v = CVC(2, -7, -2, 4)
            4  u.distance(v)

Out[23]:  6.0
```

```
In [24]:    1  # Create a vector from the normalization of a vector
            2  # (i.e. a vector in the same direction with length equal cunit)
            3  u = CVC(-4, 0, 0, 3)
            4  u.normalize()

Out[24]:  CVC(A=-0.8, B=0.0, C=0.0, D=0.6)
```

```
In [25]:    1  # Create a vector by projecting a vector onto another
            2  u = CVC(9, -9, 0, 6)
            3  v = CVC(0, 2, 1, -4)
            4  w = u.project(v)
            5  w

Out[25]:  CVC(A=-0.0, B=-4.0, C=-2.0, D=8.0)
```

```
In [26]:    1  # Create a vector from the inverse of the projection of a vector onto another
            2  w = CVC(-0, -4, -2, 8)
            3  v = CVC(-3, 3, 0, -2)
            4  u = w.inv_project(v)
            5  u

Out[26]:  CVC(A=9.0, B=-9.0, C=-0.0, D=6.0)
```

```
In [27]:    1  # Create a vector by rejecting a vector from another
            2  u = CVC(9, -9, 0, 6)
            3  v = CVC(0, 2, 1, -4)
            4  u.reject(v)

Out[27]:  CVC(A=9.0, B=-5.0, C=2.0, D=-2.0)
```

```
In [28]:   1  # Calculate the scalar projection of a vector onto another
           2  u = CVC(3, 0, 4, -1)
           3  v = CVC(-6, 0, -8, 0)
           4  u.scalar_project(v)
```

Out[28]:  -5.0

```
In [29]:   1  # Calculate the smallest angle in radians between a vector and another
           2  u = CVC(4.5, -3.0, 0.0, -1.5)
           3  w = CVC(-3.0, -3.0, 0.0, -3.0)
           4  u.angle(w)   # = pi/2
```

Out[29]:  1.5707963267948963

```
In [30]:   1  # Calculate the cosine of the smallest angle between a vector and another
           2  u = CVC(-1, 0, 0, -1)
           3  v = CVC(2, 0, -2, 2)
           4  u.cos(v)   # = -(2/3)**0.5
```

Out[30]:  -0.8164965809277259

```
In [31]:   1  # Limit the resulting value to be greater than or equal to s*cunit and smaller than or equal to t*cunit
           2  s = -1
           3  t = 2
           4  [
           5      CVC.clip(i, s, t)
           6      for i in [ -3, -2, -1, 0, 1, 2, 3, 4 ]
           7  ]
```

Out[31]:  [-1, -1, -1, 0, 1, 2, 2, 2]

```
In [ ]:    1
```