Using a Tolerant Cartesian 2D Vector Class

In [1]:
```
from math import acos

from skvectors import create_class_Tolerant_Cartesian_2D_Vector
```

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

TCVC2D = create_class_Tolerant_Cartesian_2D_Vector('TCVC2D', 'uv')

# Explicit alternative:
# TCVC2D = \
#     create_class_Tolerant_Cartesian_2D_Vector(
#         name = 'TCVC2D',
#         component_names = [ 'u', 'v' ],
#         brackets = [ '<', '>' ],
#         sep = ', ',
#         cnull = 0,
#         cunit = 1,
#         functions = None,
#         abs_tol = 1e-12,
#         rel_tol = 1e-9
#     )
```

In [3]:
```
# Absolute tolerance for vector lengths
TCVC2D.abs_tol
```

Out[3]:
```
1e-12
```

In [4]:
```python
# Relative tolerance for vector lengths
TCVC2D.rel_tol
```
Out[4]:
```
1e-09
```

In [5]:
```python
# Calculate the tolerance for a vector based on its length
u = TCVC2D(0.0, 0.0)  # u.length() = 0.0
u.tolerance(), u.tol
```
Out[5]:
```
(1e-12, 1e-12)
```

In [6]:
```python
# Calculate the tolerance for a vector based on its length
u = TCVC2D(-0.6, 0.8)  # u.length() = 1.0
u.tol, (1e6 * u).tol
```
Out[6]:
```
(1e-09, 0.001)
```

In [7]:
```python
# Calculate the tolerance for a vector based on its length
u = TCVC2D(3, -4)  # u.length() = 5.0
u.tol, (u / 1e3).tol, (u / 1e6).tol, (u / 1e9).tol
```
Out[7]:
```
(5e-09, 5.0000000000000005e-12, 1e-12, 1e-12)
```

In [8]:
```python
# Calculate a common tolerance for a vector and another based on their lengths
u = TCVC2D(0, 0)
v = TCVC2D(0, 0)
u.tolerance_with(v)
```
Out[8]:
```
1e-12
```

In [9]:
```python
# Calculate a common tolerance for a vector and another based on their lengths
u = TCVC2D(-0.6, 0.8)  # u.length() = 1.0
v = TCVC2D(3.0, -4.0)  # v.length() = 5.0
u.tolerance_with(v), v.tolerance_with(u)
```
Out[9]:
```
(5e-09, 5e-09)
```

```
In [10]: # Calculate a common tolerance for several vectors based on their lengths
         u = TCVC2D(0, 0)
         v = TCVC2D(0, 0)
         some_vectors = [ u, v ]
         TCVC2D.tolerance_all(some_vectors)
```

```
Out[10]: 1e-12
```

```
In [11]: # Calculate a common tolerance for several vectors based on their lengths
         u = TCVC2D(-0.6, 0.8)  # u.length() = 1.0
         v = TCVC2D(3.0, -4.0)  # v.length() = 5.0
         some_vectors = [ u, v, u - v, u + v ]
         TCVC2D.tolerance_all(some_vectors), TCVC2D.tolerance_all(vector for vector in some_vectors)
```

```
Out[11]: (6.000000000000001e-09, 6.000000000000001e-09)
```

```
In [12]: # NB: This does not work:
         # TCVC2D.tolerance_all([ ])
```

```
In [13]: # NB: This does not work:
         # u = TCVC2D(3.0, -4.0)
         # TCVC2D.tolerance_all([ u ])
```

```
In [14]: # Check if the length of a vector is equal to cnull (within a calculated tolerance)
         nil = TCVC2D.abs_tol / 2
         u = TCVC2D(0, -nil)  # u.length() = 5e-13
         u.is_zero_vector()
```

```
Out[14]: True
```

```
In [15]: # Check if the length of a vector is equal to cnull (within a calculated tolerance)
         not_nil = TCVC2D.abs_tol * 2
         u = TCVC2D(0, -not_nil)  # u.length() = 2e-12
         u.is_zero_vector()
```

```
Out[15]: False
```

In [16]:
```
# Check if the length of a vector is not equal to cnull (within a calculated tolerance)
nil = TCVC2D.abs_tol / 2
u = TCVC2D(0, -nil)  # u.length() = 5e-13
bool(u)
```
Out[16]:
```
False
```

In [17]:
```
# Check if the length of a vector is not equal to cnull (within a calculated tolerance)
not_nil = TCVC2D.abs_tol * 2
u = TCVC2D(0, -not_nil)  # u.length() = 2e-12
bool(u)
```
Out[17]:
```
True
```

In [18]:
```
# Check if the length of a vector is equal to cunit (within a calculated tolerance)
u = TCVC2D(-0.6, 0.8)  # u.length() = 1.0
nil = TCVC2D.rel_tol / 2
v = (1 + nil) * u  # Make the length of v slightly longer than 1.0; v.length() = 1.0 + 5e-10
v.is_unit_vector()
```
Out[18]:
```
True
```

In [19]:
```
# Check if the length of a vector is equal to cunit (within a calculated tolerance)
u = TCVC2D(-0.6, 0.8)  # u.length() = 1.0
not_nil = TCVC2D.rel_tol * 2
v = (1 + not_nil) * u  # Make the length of v longer than 1.0; v.length() = 1.0 + 2e-9
v.is_unit_vector()
```
Out[19]:
```
False
```

In [20]:
```
# Check if a vector is equal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
nil = u.tolerance() / 2
v = (1 + nil / u.length()) * u  # Make v slightly different from u
u == v
```
Out[20]:
```
True
```

In [21]:
```python
# Check if a vector is equal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
not_nil = u.tolerance() * 2
v = (1 + not_nil / u.length()) * u  # Make v different from u
u == v
```

Out[21]:
```
False
```

In [22]:
```python
# Check if a vector is equal to any of some other vectors (within a calculated tolerance)
u = TCVC2D(3, -4)
nil = u.tolerance() / 2
v = (1 + nil / u.length()) * u  # Make v slightly different from u
w = TCVC2D(-4, 3)
some_vectors = [ v, w ]
u in some_vectors
```

Out[22]:
```
True
```

In [23]:
```python
# Check if a vector is equal to any of some other vectors (within a calculated tolerance)
u = TCVC2D(3, -4)
not_nil = u.tolerance() * 2
v = (1 + not_nil / u.length()) * u  # Make v different from u
w = TCVC2D(-4, 3)
some_vectors = [ v, w ]
u in some_vectors
```

Out[23]:
```
False
```

In [24]:
```python
# Check if a vector is not equal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
nil = u.tolerance() / 2
v = (1 + nil / u.length()) * u  # Make v slightly different from u
u != v
```

Out[24]:
```
False
```

In [25]:
```
# Check if a vector is not equal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
not_nil = u.tolerance() * 2
v = (1 + not_nil / u.length()) * u  # Make v different from u
u != v
```
Out[25]:
```
True
```

In [26]:
```
# Check if a vector is not equal to any of some other vectors (within a calculated tolerance)
u = TCVC2D(3, -4)
nil = u.tolerance() / 2
v = (1 + nil / u.length()) * u  # Make v slightly different from u
w = TCVC2D(-4, 3)
some_vectors = [ v, w ]
u not in some_vectors
```
Out[26]:
```
False
```

In [27]:
```
# Check if a vector is not equal to any of some other vectors (within a calculated tolerance)
u = TCVC2D(3, -4)
not_nil = u.tolerance() * 2
v = (1 + not_nil / u.length()) * u  # Make v different from u
w = TCVC2D(-4, 3)
some_vectors = [ v, w ]
u not in some_vectors
```
Out[27]:
```
True
```

In [28]:
```
# Check if a vector has equal length to another (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(-4, 3)
nil = u.tolerance_with(v) / 2
v *= (1 + nil / u.length())  # Make v slightly longer
u.equal_lengths(v)
```
Out[28]:
```
True
```

In [29]:
```
# Check if a vector has equal length to another (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(-4, 3)
not_nil = u.tolerance_with(v) * 2
v *= (1 + not_nil / u.length())  # Make v longer
u.equal_lengths(v)
```
Out[29]:
```
False
```

In [30]:
```
# Check if a vector is shorter than another vector (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(-4, 3)
nil = u.tolerance_with(v) / 2
u *= (1 - nil / u.length())  # Make u slightly shorter
u.shorter(v)
```
Out[30]:
```
False
```

In [31]:
```
# Check if a vector is shorter than another vector (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(-4, 3)
not_nil = u.tolerance_with(v) * 2
u *= (1 - not_nil / u.length())  # Make u shorter
u.shorter(v)
```
Out[31]:
```
True
```

In [32]:
```
# Check if a vector is longer than another vector (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(-4, 3)
nil = u.tolerance_with(v) / 2
u *= (1 + nil / u.length())  # Make u slightly longer
u.longer(v)
```
Out[32]:
```
False
```

```
In [33]:
# Check if a vector is longer than another vector (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(-4, 3)
not_nil = u.tolerance_with(v) * 2
u *= (1 + not_nil / u.length())  # Make u longer
u.longer(v)

Out[33]: True
```

```
In [34]:
# Check if a vector is orthogonal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(0, 0)
nil = TCVC2D.abs_tol / 2
v.u = nil
u.are_orthogonal(v)

Out[34]: True
```

```
In [35]:
# Check if a vector is orthogonal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
v = TCVC2D(0, 0)
not_nil = TCVC2D.abs_tol * 2
v.u = not_nil
u.are_orthogonal(v)

Out[35]: False
```

```
In [36]:
# Check if a vector is orthogonal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
nil = TCVC2D.abs_tol / 2  # = 5e-13
v = u.rotate(acos(nil))  # u.cos(v) = 5e-13
u.are_orthogonal(v), (u * 1e9).are_orthogonal(v / 1e9), (u / 1e9).are_orthogonal(v * 1e9)

Out[36]: (True, True, True)
```

```
In [37]:
# Check if a vector is orthogonal to another (within a calculated tolerance)
u = TCVC2D(3, -4)
not_nil = TCVC2D.abs_tol * 2  # = 2e-12
v = u.rotate(acos(not_nil))  # u.cos(v) = 2e-12
u.are_orthogonal(v), (u * 1e9).are_orthogonal(v / 1e9), (u / 1e9).are_orthogonal(v * 1e9)

Out[37]: (False, False, False)
```

In [38]:

```
# Create a vector by rounding the component values in a vector
u = TCVC2D(-1.000000004, 2.123456789)  #  u.tolerance() = circa 2.3e-9
u.round_components(), u.cround
```

Out[38]:

```
(TCVC2D(u=-1.0, v=2.12345679), TCVC2D(u=-1.0, v=2.12345679))
```

In [ ]:

```

```