Using a Fundamental Vector Class

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

In [1]:
```
from skvectors import create_class_Fundamental_Vector
```

In [2]:
```
# Create a 3-dimensional fundamental vector class

# The first argument is a string with the name of the class
# to be created.

# The number of elements in the iterable given as the second
# argument determines the number of dimensions for the class.

FVC = create_class_Fundamental_Vector('FVC', 'abc')

# Explicit alternative:
# FVC = \
#     create_class_Fundamental_Vector(
#         name = 'FVC',
#         component_names = [ 'a', 'b', 'c' ],
#         brackets = [ '<', '>' ],
#         sep = ', '
#     )
```

In [3]:
```
# Number of dimensions for vectors in the class
FVC.dimensions()
```

Out[3]: 3

In [4]:
```
# Brackets for vectors in the class
# (Used when printing a vector and when applying str to a vector)
FVC.brackets
```

Out[4]:
```
['<', '>']
```

In [5]:
```
# Separator between components for vectors in the class
# (Used when printing a vector and when applying str or repr to a vector)
FVC.sep
```

Out[5]:
```
', '
```

In [6]:
```
# List of component names for vectors in the class
FVC.component_names()
```

Out[6]:
```
['a', 'b', 'c']
```

In [7]:
```
# Initialize vector
FVC(1, -2, +3)
```

Out[7]:
```
FVC(a=1, b=-2, c=3)
```

In [8]:
```
# Initialize vector
FVC(a=1, b=-2, c=+3)
```

Out[8]:
```
FVC(a=1, b=-2, c=3)
```

In [9]:
```
# Initialize vector
l = [ 1, -2, 3 ]
FVC(*l)
```

Out[9]:
```
FVC(a=1, b=-2, c=3)
```

In [10]:
```
# Initialize vector
d = { 'a': 1, 'b': -2, 'c': 3 }
FVC(**d)
```

Out[10]:
```
FVC(a=1, b=-2, c=3)
```

In [11]:
```
# Initialize vector
FVC.repeat_cvalue(8)
```

Out[11]:
```
FVC(a=8, b=8, c=8)
```

In [12]:
```
# Number of dimensions of vector
v = FVC(0, 0, 0)
v.dimensions()
```

Out[12]:
```
3
```

In [13]:
```
# Number of dimensions of vector
v = FVC(0, 0, 0)
len(v)
```

Out[13]:
```
3
```

In [14]:
```
# List of component names for vector
v = FVC(0, 0, 0)
v.cnames
```

Out[14]:
```
['a', 'b', 'c']
```

In [15]:
```
# Check if something is a vector
v = FVC(-3, 4, 5)
FVC.is_vector(v)
```

Out[15]:
```
True
```

In [16]:
```
# Check if something is a vector
d = { 'a': -3, 'b': 4, 'c': 5 }
FVC.is_vector(d)
```

Out[16]:
```
False
```

In [17]:
```
# Print vector
print(FVC(2, 4, 6))
```

```
<2, 4, 6>
```

In [18]:
```
# Apply str to vector
v = FVC(2, 4, 6)
str(v)
```

Out[18]:
```
'<2, 4, 6>'
```

In [19]:
```
# Apply str to vector inside a string
v = FVC(-3.3, 4.6, -5.5)
'str applied to a vector: {!s}'.format(v)
```

Out[19]:
```
'str applied to a vector: <-3.3, 4.6, -5.5>'
```

In [20]:
```
# Apply repr to vector
v = FVC(2, 4, 6)
repr(v)
```

Out[20]:
```
'FVC(a=2, b=4, c=6)'
```

In [21]:
```
# NB: This does only work if the sep parameter in the class
# creation above contains a comma, or a comma and space(s)

# Apply repr to vector
v = FVC(2, 4, 6)
eval(repr(v))
```

Out[21]:
```
FVC(a=2, b=4, c=6)
```

In [22]:
```
# Apply repr to vector inside a string
v = FVC(-3.3, 4.6, -5.5)
'repr applied to a vector: {!r}'.format(v)
```

Out[22]:
```
'repr applied to a vector: FVC(a=-3.3, b=4.6, c=-5.5)'
```

In [23]:
```
# Format vector
v = FVC(2.222222, 4.444444, 6.6666666)
format(v, '.3e')
```

Out[23]:
```
'<2.222e+00, 4.444e+00, 6.667e+00>'
```

In [24]:
```
# Format vectors inside string
u = FVC(2.222222, 4.444444, 6.6666666)
w = FVC(-3.3, 4.6, -5.5)
'format applied to two vectors: {0:.4e} and {1:.2e}'.format(u, w)
```

Out[24]: 'format applied to two vectors: <2.2222e+00, 4.4444e+00, 6.6667e+00> and <-3.30e+00, 4.60e+00, -5.50e+00>'

In [25]:
```
# Check if vector contains a value
v = FVC(2, 3, 4)
3 in v
```

Out[25]: True

In [26]:
```
# Check if vector does not contain a value
v = FVC(2, 3, 4)
3.0 not in v
```

Out[26]: False

In [27]:
```
# The component values
v = FVC(-6, 8, 3)
v.a, v.b, v.c
```

Out[27]: (-6, 8, 3)

In [28]:
```
# Change the component values
v = FVC(0, 0, 0)
v.a, v.b, v.c = 6, 7, 8
v
```

Out[28]: FVC(a=6, b=7, c=8)

In [29]:
```
# Change a component value
v = FVC(0, 0, 0)
v.a += 100
v
```

Out[29]: FVC(a=100, b=0, c=0)

In [30]:
```
# Change a component value
v = FVC(3, -4, 20)
v.c //= 8
v
```

Out[30]: FVC(a=3, b=-4, c=2)

In [31]:
```
# The component values / Indexing of vector
v = FVC(7, -8, 9)
v[0], v[1], v[2]
```

Out[31]: (7, -8, 9)

In [32]:
```
# The component values / Indexing of vector
v[-3], v[-2], v[-1]
```

Out[32]: (7, -8, 9)

In [33]:
```
# Indexing of vector
v = FVC(7, -8, 9)
v[0:3], v[:], v[::]
```

Out[33]: ([7, -8, 9], [7, -8, 9], [7, -8, 9])

In [34]:
```
# Change the component values
v = FVC(0, 0, 0)
v[0], v[1], v[2] = 7, -8, 9
v
```

Out[34]: FVC(a=7, b=-8, c=9)

In [35]:
```
# Change the component values
v = FVC(0, 0, 0)
v[0:3] = 7, -8, 9
v
```

Out[35]: FVC(a=7, b=-8, c=9)

In [36]:
```
# Change the component values
u = FVC(0, 0, 0)
w = FVC(7, -8, 9)
u[:] = w
u
```

Out[36]: FVC(a=7, b=-8, c=9)

In [37]:
```
# Change the component values
v = FVC(0, 0, 0)
v[:] = (cv for cv in [ 7, -8, 9 ])
v
```

Out[37]: FVC(a=7, b=-8, c=9)

In [38]:
```
# List of the component values
v = FVC(7, -8, 9)
v.cvalues, v.component_values(), v[:]
```

Out[38]: ([7, -8, 9], [7, -8, 9], [7, -8, 9])

In [39]:
```
# List of the component values
v = FVC(7, -8, 9)
list(v), [ *v ], [ getattr(v, cn) for cn in v.cnames ]
```

Out[39]: ([7, -8, 9], [7, -8, 9], [7, -8, 9])

In [40]:
```
# Iterate over the components
x, y, z = FVC(7, -8, 9)
x, y, z
```

Out[40]: (7, -8, 9)

In [41]:
```
# Iterate over the components
v = FVC(7, -8, 9)
g = (cv for cv in v)
print(*g)
```

```
7 -8 9
```

In [42]:
```
# Iterate over the components
v = FVC(7, -8, 9)
components = iter(v)
next(components), next(components), next(components)
```

Out[42]:
```
(7, -8, 9)
```

In [43]:
```
# Check if vectors are equal
v = FVC(2.0, 4.0, 6.0)
v == FVC(2, 4, 6)
```

Out[43]:
```
True
```

In [44]:
```
# Check if vectors are not equal
v = FVC(2, 4, 6)
v != FVC(2.0, 4.0, 6.0)
```

Out[44]:
```
False
```

In [45]:
```
# Vector as dictionary
v = FVC(2, 4, 6)
v.as_dict()
```

Out[45]:
```
{'a': 2, 'b': 4, 'c': 6}
```

In [46]:
```
# Make shallow copy of vector
u = FVC(2, 4, 6)
w = FVC(*u)
w
```

Out[46]:
```
FVC(a=2, b=4, c=6)
```

In [47]:
```
# Make shallow copy of vector
u = FVC(2, 4, 6)
w = u.copy()
w
```

Out[47]:
```
FVC(a=2, b=4, c=6)
```

In [48]:
```
# Apply a lambda function to each component
v = FVC(-3.3, 4.6, -5.5)
v(lambda s: 10 + s * 1000)
```

Out[48]: FVC(a=-3290.0, b=4610.0, c=-5490.0)

In [49]:
```
# Apply the abs function to each component
v = FVC(-3.3, 4.6, -5.5)
v(abs)
```

Out[49]: FVC(a=3.3, b=4.6, c=5.5)

In [50]:
```
# Apply the abs function to each component
v = FVC(-3, 4, -5)
FVC(*map(abs, v))
```

Out[50]: FVC(a=3, b=4, c=5)

In [51]:
```
# Apply the int class to each component
v = FVC(-3.3, 4.6, -5.5)
v(int)
```

Out[51]: FVC(a=-3, b=4, c=-5)

In [52]:
```
# Change the component values by applying the int class to each component
v = FVC(-3.3, 4.6, -5.5)
v[:] = map(int, v)
v
```

Out[52]: FVC(a=-3, b=4, c=-5)

In [53]:
```
# Create a vector method that takes 1 vector as argument


def square(s):

    return s**2


FVC.create_vector_method_arg1('square', square)
v = FVC(2, 3, -4)
v.vector_square()
```

Out[53]:
```
FVC(a=4, b=9, c=16)
```

In [54]:
```
# Create, from a built in function, a vector method that takes 1 vector as argument
FVC.create_vector_method_arg1('abs', lambda s: abs(s))
v = FVC(2, 3, -4)
v.vector_abs()
```

Out[54]:
```
FVC(a=2, b=3, c=4)
```

In [55]:
```
# Create a vector method that takes 2 vectors as arguments


def add(s, t):

    return s + t


FVC.create_vector_method_arg2('add', add)

v = FVC(2, 3, -4)
v.vector_add(FVC(1, -2, 3)), v.vector_add(1000)
```

Out[55]:
```
(FVC(a=3, b=1, c=-1), FVC(a=1002, b=1003, c=996))
```

In [56]:

```
# Create a vector method that takes 3 vectors as arguments


def select(r, s, t):

    if r < 0:
        result = s
    else:
        result = t

    return result


FVC.create_vector_method_arg3('select', select)

v = FVC(-2, 0, 3)
v.vector_select(FVC(1, 3, 5), FVC(2, 4, 6)), v.vector_select(0, 100)
```

Out[56]:

```
(FVC(a=1, b=4, c=6), FVC(a=0, b=100, c=100))
```

In [ ]:

```

```