
ppb-vector Documentation

Piper Thunstrom, Jamie Bliss

Jul 07, 2019

Contents

1 Inheriting from Vector	7
Index	9

Note: ppb-vector follows the [semver](#) (semantic versioning) convention. In a nutshell, we commit to forward compatibility within a major version: code that works with version 1.0 ought to work with any 1.x release.

`class ppb_vector.Vector`

The immutable, 2D vector class of the PursuedPyBear project.

`Vector` is an immutable 2D Vector, which can be instantiated as expected:

```
>>> from ppb_vector import Vector
>>> Vector(3, 4)
Vector(3.0, 4.0)
```

`Vector` implements many convenience features, as well as useful mathematical operations for 2D geometry and linear algebra.

`Vector` acts as an iterable and a sequence, allowing usage like converting, indexing, and unpacking:

```
>>> v = Vector(-3, -5)
>>> list(v)
[-3.0, -5.0]
>>> tuple(v)
(-3.0, -5.0)
```

```
>>> v[0]
-3.0
```

```
>>> x, y = Vector(1, 2)
>>> x
1.0
```

```
>>> print(*Vector(1, 2))
1.0 2.0
```

It also acts mostly like a mapping, when it does not conflict with being a sequence. In particular, the coordinates may be accessed by subscripting:

```
>>> v["y"]
-5.0
>>> v["x"]
-3.0
```

x : float

The X coordinate of the vector

y : float

The Y coordinate of the vector

__add__ (*other*: Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]]) → ppb_vector.Vector
Add two vectors.

Parameters *other* – A `Vector` or a vector-like. For a description of vector-likes, see `__new__()`.

```
>>> Vector(1, 0) + (0, 1)
Vector(1.0, 1.0)
>>> (0, 1) + Vector(1, 0)
Vector(1.0, 1.0)
```

`__bool__()` → bool

Check whether the vector is non-zero.

```
>>> assert Vector(1, 1)
>>> assert not Vector(0, 0)
```

`__eq__(other: Any)` → bool

Test whether two vectors are equal.

Return type bool

Parameters **other** – A `Vector` or a vector-like. For a description of vector-likes, see `__new__()`.

```
>>> Vector(1, 0) == (0, 1)
False
```

`__mul__(other)`

Perform a dot product or a scalar product, based on the parameter type.

Parameters

- **other** – If `other` is a scalar (an instance of `typing.SupportsFloat`), return `self.scale_by(other)`.

```
>>> v = Vector(1, 1)
>>> 3 * v
Vector(3.0, 3.0)
```

```
>>> assert 3 * v == v * 3 == v.scale_by(3)
```

A `Vector` can also be divided by a scalar,

using the `/` operator:

```
>>> assert 3 * v / 3 == v
```

- **other** – If `other` is a vector-like, return `self.dot(other)`.

```
>>> v * (-1, -1)
-2.0
```

```
>>> assert v * (-1, -1) == (-1, -1) * v == v.dot((-1, -1))
```

Vector-likes are defined in `convert()`.

`__neg__()` → `ppb_vector.Vector`

Negate a vector.

Negating a `Vector` produces one with identical length and opposite direction. It is equivalent to multiplying it by -1.

```
>>> -Vector(1, 1)
Vector(-1.0, -1.0)
```

static `__new__` (*cls*, **args*, ***kwargs*)

Make a vector from coordinates, or convert a vector-like.

A vector-like can be:

- a length-2 [Sequence](#), whose contents are interpreted as the *x* and *y* coordinates like (4, 2)
- a length-2 [Mapping](#), whose keys are *x* and *y* like {'x': 4, 'y': 2}
- any instance of [Vector](#) or any subclass.

`__reduce__` ()

Helper for pickle.

`__sub__` (*other*: Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]]) → ppb_vector.Vector

Subtract one vector from another.

Parameters *other* – A [Vector](#) or a vector-like. For a description of vector-likes, see

`__new__` ().

```
>>> Vector(3, 3) - (1, 1)
Vector(2.0, 2.0)
```

`__truediv__` (*other*: SupportsFloat) → ppb_vector.Vector

Perform a division between a vector and a scalar.

```
>>> Vector(3, 3) / 3
Vector(1.0, 1.0)
```

angle (*other*: Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]]) → float

Compute the angle between two vectors.

Return type float

Parameters *other* – A [Vector](#) or a vector-like. For a description of vector-likes, see

`__new__` ().

```
>>> Vector(1, 0).angle( (0, 1) )
90.0
```

As with [rotate\(\)](#), angles are expressed in degrees, signed, and refer to a direct coordinate system (i.e. positive rotations are counter-clockwise).

[angle\(\)](#) is guaranteed to produce an angle between -180° and 180°.

asdict () → Mapping[str, float]

Convert a vector to a vector-like dictionary.

```
>>> v = Vector(42, 69)
>>> v.asdict()
{'x': 42.0, 'y': 69.0}
```

The conversion can be reversed by constructing:

```
>>> assert v == Vector(v.asdict())
```

dot (*other*: Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]]) → float
Compute the dot product of two vectors.

Return type float

Parameters **other** – A *Vector* or a vector-like. For a description of vector-likes, see `__new__()`.

```
>>> Vector(1, 1).dot((-1, -1))
-2.0
```

This can also be expressed with `*`:

```
>>> assert Vector(1, 2).dot([2, 1]) == Vector(1, 2) * [2, 1]
```

isclose (*other*: Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]], *, *abs_tol*: SupportsFloat = 1e-09, *rel_tol*: SupportsFloat = 1e-09, *rel_to*: Sequence[Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]]] = ()) → bool
Perform an approximate comparison of two vectors.

Return type bool

Parameters **other** – A *Vector* or a vector-like. For a description of vector-likes, see `__new__()`.

```
>>> assert Vector(1, 0).isclose((1, 1e-10))
```

`isclose()` takes optional, keyword arguments, akin to those of `math.isclose()`:

Parameters

- **abs_tol** – the absolute tolerance is the minimum magnitude (of the difference vector) under which two inputs are considered close, without consideration for (the magnitude of) the input values.
- **rel_tol** – the relative tolerance: if the length of the difference vector is less than `rel_tol * input.length` for any input, the two vectors are considered close.
- **rel_to** – an iterable of additional vector-likes which are considered to be inputs, for the purpose of the relative tolerance.

length

Compute the length of a vector.

```
>>> Vector(45, 60).length
75.0
```

normalize () → ppb_vector.Vector

Return a vector with the same direction and unit length.

```
>>> Vector(3, 4).normalize()
Vector(0.6, 0.8)
```

Note that `normalize()` is equivalent to `scale(1)`:

```
>>> assert Vector(7, 24).normalize() == Vector(7, 24).scale_to(1)
```


reflect (*surface_normal*: Union[Vector, Tuple[SupportsFloat, SupportsFloat], Sequence[SupportsFloat], Mapping[str, SupportsFloat]]) → ppb_vector.Vector
 Reflect a vector against a surface.

Parameters **other** – A *Vector* or a vector-like. For a description of vector-likes, see `__new__()`.

Compute the reflection of a *Vector* on a surface going through the origin, described by its normal vector.

```
>>> Vector(5, 3).reflect((-1, 0))
Vector(-5.0, 3.0)
```

```
>>> Vector(5, 3).reflect(Vector(-1, -2).normalize())
Vector(0.5999999999999996, -5.800000000000001)
```

rotate (*angle*: SupportsFloat) → ppb_vector.Vector

Rotate a vector.

Rotate a vector in relation to the origin and return a new *Vector*.

```
>>> Vector(1, 0).rotate(90)
Vector(0.0, 1.0)
```

Positive rotation is counter/anti-clockwise.

scale_by (*scalar*: SupportsFloat) → ppb_vector.Vector

Compute a vector-scalar multiplication.

```
>>> Vector(1, 2).scale_by(3)
Vector(3.0, 6.0)
```

Can also be expressed with `*`:

```
>>> assert Vector(1, 2).scale_by(3) == 3 * Vector(1, 2)
```

scale_to (*length*: SupportsFloat) → ppb_vector.Vector

Scale a given *Vector* to a certain length.

```
>>> Vector(7, 24).scale_to(2)
Vector(0.56, 1.92)
```

truncate (*max_length*: SupportsFloat) → ppb_vector.Vector

Scale a given *Vector* down to a given length, if it is larger.

```
>>> Vector(7, 24).truncate(3)
Vector(0.84, 2.88)
```

It produces a vector with the same direction, but possibly a different length.

Note that `vector.scale(max_length)` is equivalent to `vector.truncate(max_length)` when `max_length < vector.length`.

```
>>> Vector(3, 4).scale(4)
Vector(2.4, 3.2)
>>> Vector(3, 4).truncate(4)
Vector(2.4, 3.2)
```

```
>>> Vector(3, 4).scale(6)
Vector(3.6, 4.8)
>>> Vector(3, 4).truncate(6)
Vector(3.0, 4.0)
```

Note: `x.truncate(max_length)` may sometimes be slightly-larger than `max_length`, due to floating-point rounding effects.

update (*x: Optional[SupportsFloat] = None, y: Optional[SupportsFloat] = None*)

Return a new `Vector` replacing specified fields with new values.

CHAPTER 1

Inheriting from `Vector`

`Vector` does not support inheritance.

Symbols

`__add__()` (*ppb_vector.Vector method*), 1
`__bool__()` (*ppb_vector.Vector method*), 2
`__eq__()` (*ppb_vector.Vector method*), 2
`__mul__()` (*ppb_vector.Vector method*), 2
`__neg__()` (*ppb_vector.Vector method*), 2
`__new__()` (*ppb_vector.Vector static method*), 3
`__reduce__()` (*ppb_vector.Vector method*), 3
`__sub__()` (*ppb_vector.Vector method*), 3
`__truediv__()` (*ppb_vector.Vector method*), 3

A

`angle()` (*ppb_vector.Vector method*), 3
`asdict()` (*ppb_vector.Vector method*), 3

D

`dot()` (*ppb_vector.Vector method*), 3

I

`isclose()` (*ppb_vector.Vector method*), 4

L

`length` (*ppb_vector.Vector attribute*), 4

N

`normalize()` (*ppb_vector.Vector method*), 4

R

`reflect()` (*ppb_vector.Vector method*), 4
`rotate()` (*ppb_vector.Vector method*), 5

S

`scale_by()` (*ppb_vector.Vector method*), 5
`scale_to()` (*ppb_vector.Vector method*), 5

T

`truncate()` (*ppb_vector.Vector method*), 5

U

`update()` (*ppb_vector.Vector method*), 6

V

`Vector` (*class in ppb_vector*), 1

X

`x` (*ppb_vector.Vector attribute*), 1

Y

`y` (*ppb_vector.Vector attribute*), 1