Examples

```
>>> from sympy import RegularPolygon, Point, Polygon
>>> t = Polygon(*RegularPolygon(Point(0, 0), 1, 3).vertices)
>>> t
Triangle(Point2D(1, 0), Point2D(-1/2, sqrt(3)/2), Point2D(-1/2, -
↪sqrt(3)/2))
>>> t.scale(2)
Triangle(Point2D(2, 0), Point2D(-1, sqrt(3)/2), Point2D(-1, -sqrt(3)/
↪2))
>>> t.scale(2, 2)
Triangle(Point2D(2, 0), Point2D(-1, sqrt(3)), Point2D(-1, -sqrt(3)))
```

See also:

rotate (page 2196), *translate* (page 2197)

translate(*x=0*, *y=0*)

Shift the object by adding to the x,y-coordinates the values x and y.

Examples

```
>>> from sympy import RegularPolygon, Point, Polygon
>>> t = Polygon(*RegularPolygon(Point(0, 0), 1, 3).vertices)
>>> t
Triangle(Point2D(1, 0), Point2D(-1/2, sqrt(3)/2), Point2D(-1/2, -
↪sqrt(3)/2))
>>> t.translate(2)
Triangle(Point2D(3, 0), Point2D(3/2, sqrt(3)/2), Point2D(3/2, -
↪sqrt(3)/2))
>>> t.translate(2, 2)
Triangle(Point2D(3, 2), Point2D(3/2, sqrt(3)/2 + 2), Point2D(3/2, 2 -␣
↪sqrt(3)/2))
```

See also:

rotate (page 2196), *scale* (page 2196)

Utils

sympy.geometry.util.**intersection**(**entities*, *pairwise=False*, ***kwargs*)

The intersection of a collection of GeometryEntity instances.

Parameters

entities : sequence of GeometryEntity

pairwise (keyword argument) : Can be either True or False

Returns

intersection : list of GeometryEntity

Raises

NotImplementedError

When unable to calculate intersection.

Notes

The intersection of any geometrical entity with itself should return a list with one item: the entity in question. An intersection requires two or more entities. If only a single entity is given then the function will return an empty list. It is possible for $intersection$ to miss intersections that one knows exists because the required quantities were not fully simplified internally. Reals should be converted to Rationals, e.g. Rational(str(real_num)) or else failures due to floating point issues may result.

Case 1: When the keyword argument 'pairwise' is False (default value): In this case, the function returns a list of intersections common to all entities.

Case 2: When the keyword argument 'pairwise' is True: In this case, the functions returns a list intersections that occur between any pair of entities.

Examples

```
>>> from sympy import Ray, Circle, intersection
>>> c = Circle((0, 1), 1)
>>> intersection(c, c.center)
[]
>>> right = Ray((0, 0), (1, 0))
>>> up = Ray((0, 0), (0, 1))
>>> intersection(c, right, up)
[Point2D(0, 0)]
>>> intersection(c, right, up, pairwise=True)
[Point2D(0, 0), Point2D(0, 2)]
>>> left = Ray((1, 0), (0, 0))
>>> intersection(right, left)
[Segment2D(Point2D(0, 0), Point2D(1, 0))]
```

See also:

sympy.geometry.entity.GeometryEntity.intersection (page 2194)

sympy.geometry.util.**convex_hull**(**args*, *polygon=True*)

The convex hull surrounding the Points contained in the list of entities.

Parameters

args : a collection of Points, Segments and/or Polygons

Returns

convex_hull : Polygon if `polygon` is True else as a tuple (U, L) where

L and U are the lower and upper hulls, respectively.

Optional Parameters

polygon
[Boolean. If True, returns a Polygon, if false a tuple, see below.] Default is True.

Notes

This can only be performed on a set of points whose coordinates can be ordered on the number line.

Examples

```
>>> from sympy import convex_hull
>>> points = [(1, 1), (1, 2), (3, 1), (-5, 2), (15, 4)]
>>> convex_hull(*points)
Polygon(Point2D(-5, 2), Point2D(1, 1), Point2D(3, 1), Point2D(15, 4))
>>> convex_hull(*points, **dict(polygon=False))
([Point2D(-5, 2), Point2D(15, 4)],
 [Point2D(-5, 2), Point2D(1, 1), Point2D(3, 1), Point2D(15, 4)])
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.polygon.Polygon* (page 2273)

References

[R521], [R522]

`sympy.geometry.util.`**`are_similar`**(*e1*, *e2*)

Are two geometrical entities similar.

Can one geometrical entity be uniformly scaled to the other?

Parameters
e1 : GeometryEntity

e2 : GeometryEntity

Returns
are_similar : boolean

Raises
GeometryError

When $e1$ and $e2$ cannot be compared.

Notes

If the two objects are equal then they are similar.

Examples

```
>>> from sympy import Point, Circle, Triangle, are_similar
>>> c1, c2 = Circle(Point(0, 0), 4), Circle(Point(1, 4), 3)
>>> t1 = Triangle(Point(0, 0), Point(1, 0), Point(0, 1))
>>> t2 = Triangle(Point(0, 0), Point(2, 0), Point(0, 2))
>>> t3 = Triangle(Point(0, 0), Point(3, 0), Point(0, 1))
>>> are_similar(t1, t2)
True
>>> are_similar(t1, t3)
False
```

See also:

sympy.geometry.entity.GeometryEntity.is_similar (page 2195)

sympy.geometry.util.**centroid**(**args*)

Find the centroid (center of mass) of the collection containing only Points, Segments or Polygons. The centroid is the weighted average of the individual centroid where the weights are the lengths (of segments) or areas (of polygons). Overlapping regions will add to the weight of that region.

If there are no objects (or a mixture of objects) then None is returned.

Examples

```
>>> from sympy import Point, Segment, Polygon
>>> from sympy.geometry.util import centroid
>>> p = Polygon((0, 0), (10, 0), (10, 10))
>>> q = p.translate(0, 20)
>>> p.centroid, q.centroid
(Point2D(20/3, 10/3), Point2D(20/3, 70/3))
>>> centroid(p, q)
Point2D(20/3, 40/3)
>>> p, q = Segment((0, 0), (2, 0)), Segment((0, 0), (2, 2))
>>> centroid(p, q)
Point2D(1, 2 - sqrt(2))
>>> centroid(Point(0, 0), Point(2, 0))
Point2D(1, 0)
```

Stacking 3 polygons on top of each other effectively triples the weight of that polygon:

```
>>> p = Polygon((0, 0), (1, 0), (1, 1), (0, 1))
>>> q = Polygon((1, 0), (3, 0), (3, 1), (1, 1))
>>> centroid(p, q)
Point2D(3/2, 1/2)
>>> centroid(p, p, p, q) # centroid x-coord shifts left
Point2D(11/10, 1/2)
```

Stacking the squares vertically above and below p has the same effect:

```
>>> centroid(p, p.translate(0, 1), p.translate(0, -1), q)
Point2D(11/10, 1/2)
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Segment* (page 2233), *sympy.geometry.polygon.Polygon* (page 2273)

sympy.geometry.util.**idiff**(*eq*, *y*, *x*, *n=1*)

Return dy/dx assuming that `eq == 0`.

Parameters

y : the dependent variable or a list of dependent variables (with y first)

x : the variable that the derivative is being taken with respect to

n : the order of the derivative (default is 1)

Examples

```
>>> from sympy.abc import x, y, a
>>> from sympy.geometry.util import idiff
```

```
>>> circ = x**2 + y**2 - 4
>>> idiff(circ, y, x)
-x/y
>>> idiff(circ, y, x, 2).simplify()
(-x**2 - y**2)/y**3
```

Here, a is assumed to be independent of x:

```
>>> idiff(x + a + y, y, x)
-1
```

Now the x-dependence of a is made explicit by listing a after y in a list.

```
>>> idiff(x + a + y, [y, a], x)
-Derivative(a, x) - 1
```

See also:

***sympy.core.function.Derivative* (page 1042)**
represents unevaluated derivatives

***sympy.core.function.diff* (page 1048)**
explicitly differentiates wrt symbols

Points

class sympy.geometry.point.**Point**(**args*, ***kwargs*)

A point in a n-dimensional Euclidean space.

Parameters

coords : sequence of n-coordinate values. In the special case where n=2 or 3, a Point2D or Point3D will be created as appropriate.

evaluate : if $True$ (default), all floats are turn into exact types.

dim : number of coordinates the point should have. If coordinates are unspecified, they are padded with zeros.

on_morph : indicates what should happen when the number of coordinates of a point need to be changed by adding or removing zeros. Possible values are $'warn'$, $'error'$, or $ignore$ (default). No warning or error is given when $*args$ is empty and dim is given. An error is always raised when trying to remove nonzero coordinates.

Raises

TypeError : When instantiating with anything but a Point or sequence

ValueError : when instantiating with a sequence with length < 2 or when trying to reduce dimensions if keyword $on_morph =' error'$ is set.

Examples

```
>>> from sympy import Point
>>> from sympy.abc import x
>>> Point(1, 2, 3)
Point3D(1, 2, 3)
>>> Point([1, 2])
Point2D(1, 2)
>>> Point(0, x)
Point2D(0, x)
>>> Point(dim=4)
Point(0, 0, 0, 0)
```

Floats are automatically converted to Rational unless the evaluate flag is False:

```
>>> Point(0.5, 0.25)
Point2D(1/2, 1/4)
>>> Point(0.5, 0.25, evaluate=False)
Point2D(0.5, 0.25)
```

See also:

sympy.geometry.line.Segment **(page 2233)**
Connects two Points

Attributes

length	
origin: A *Point* representing the origin of the	appropriately-dimensioned space.

static affine_rank(**args*)

The affine rank of a set of points is the dimension of the smallest affine space containing all the points. For example, if the points lie on a line (and are not all the same) their affine rank is 1. If the points lie on a plane but not a line, their affine rank is 2. By convention, the empty set has affine rank -1.

property ambient_dimension

Number of components this point has.

classmethod are_coplanar(**points*)

Return True if there exists a plane in which all the points lie. A trivial True value is returned if $len(points) < 3$ or all Points are 2-dimensional.

Parameters
A set of points

Returns
boolean

Raises
ValueError : if less than 3 unique points are given

Examples

```
>>> from sympy import Point3D
>>> p1 = Point3D(1, 2, 2)
>>> p2 = Point3D(2, 7, 2)
>>> p3 = Point3D(0, 0, 2)
>>> p4 = Point3D(1, 1, 2)
>>> Point3D.are_coplanar(p1, p2, p3, p4)
True
>>> p5 = Point3D(0, 1, 3)
>>> Point3D.are_coplanar(p1, p2, p3, p5)
False
```

canberra_distance(*p*)

The Canberra Distance from self to point p.

Returns the weighted sum of horizontal and vertical distances to point p.

Parameters
p : Point

Returns
canberra_distance : The weighted sum of horizontal and vertical distances to point p. The weight used is the sum of absolute values of the coordinates.

Raises
ValueError when both vectors are zero.

Examples

```
>>> from sympy import Point
>>> p1, p2 = Point(1, 1), Point(3, 3)
>>> p1.canberra_distance(p2)
1
>>> p1, p2 = Point(0, 0), Point(3, 3)
>>> p1.canberra_distance(p2)
2
```

See also:

sympy.geometry.point.Point.distance (page 2204)

distance(*other*)

The Euclidean distance between self and another GeometricEntity.

Returns
distance : number or symbolic expression.

Raises
TypeError : if other is not recognized as a GeometricEntity or is a GeometricEntity for which distance is not defined.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(1, 1), Point(4, 5)
>>> l = Line((3, 1), (2, 2))
>>> p1.distance(p2)
5
>>> p1.distance(l)
sqrt(2)
```

The computed distance may be symbolic, too:

```
>>> from sympy.abc import x, y
>>> p3 = Point(x, y)
>>> p3.distance((0, 0))
sqrt(x**2 + y**2)
```

See also:

sympy.geometry.line.Segment.length (page 2235), *sympy.geometry.point.Point.taxicab_distance* (page 2208)

dot(*p*)

Return dot product of self with another Point.

equals(*other*)

Returns whether the coordinates of self and other agree.

intersection(*other*)

The intersection between this point and another GeometryEntity.

Parameters
other : GeometryEntity or sequence of coordinates

Returns
intersection : list of Points

Notes

The return value will either be an empty list if there is no intersection, otherwise it will contain this point.

Examples

```
>>> from sympy import Point
>>> p1, p2, p3 = Point(0, 0), Point(1, 1), Point(0, 0)
>>> p1.intersection(p2)
[]
>>> p1.intersection(p3)
[Point2D(0, 0)]
```

is_collinear(**args*)

Returns $True$ if there exists a line that contains $self$ and $points$. Returns $False$ otherwise. A trivially True value is returned if no points are given.

Parameters
args : sequence of Points

Returns
is_collinear : boolean

Examples

```
>>> from sympy import Point
>>> from sympy.abc import x
>>> p1, p2 = Point(0, 0), Point(1, 1)
>>> p3, p4, p5 = Point(2, 2), Point(x, x), Point(1, 2)
>>> Point.is_collinear(p1, p2, p3, p4)
True
>>> Point.is_collinear(p1, p2, p3, p5)
False
```

See also:

sympy.geometry.line.Line (page 2227)

is_concyclic(**args*)

Do $self$ and the given sequence of points lie in a circle?

Returns True if the set of points are concyclic and False otherwise. A trivial value of True is returned if there are fewer than 2 other points.

Parameters

args : sequence of Points

Returns

is_concyclic : boolean

Examples

```
>>> from sympy import Point
```

Define 4 points that are on the unit circle:

```
>>> p1, p2, p3, p4 = Point(1, 0), (0, 1), (-1, 0), (0, -1)
```

```
>>> p1.is_concyclic() == p1.is_concyclic(p2, p3, p4) == True
True
```

Define a point not on that circle:

```
>>> p = Point(1, 1)
```

```
>>> p.is_concyclic(p1, p2, p3)
False
```

property is_nonzero

True if any coordinate is nonzero, False if every coordinate is zero, and None if it cannot be determined.

is_scalar_multiple(*p*)

Returns whether each coordinate of $self$ is a scalar multiple of the corresponding coordinate in point p.

property is_zero

True if every coordinate is zero, False if any coordinate is not zero, and None if it cannot be determined.

property length

Treating a Point as a Line, this returns 0 for the length of a Point.

Examples

```
>>> from sympy import Point
>>> p = Point(0, 1)
>>> p.length
0
```

midpoint(*p*)

The midpoint between self and point p.

Parameters

p : Point

Returns

midpoint : Point

Examples

```
>>> from sympy import Point
>>> p1, p2 = Point(1, 1), Point(13, 5)
>>> p1.midpoint(p2)
Point2D(7, 3)
```

See also:

sympy.geometry.line.Segment.midpoint (page 2235)

property origin

A point of all zeros of the same ambient dimension as the current point

property orthogonal_direction

Returns a non-zero point that is orthogonal to the line containing $self$ and the origin.

Examples

```
>>> from sympy import Line, Point
>>> a = Point(1, 2, 3)
>>> a.orthogonal_direction
Point3D(-2, 1, 0)
>>> b = _
>>> Line(b, b.origin).is_perpendicular(Line(a, a.origin))
True
```

static project(*a*, *b*)

Project the point a onto the line between the origin and point b along the normal direction.

Parameters

a : Point

b : Point

Returns

p : Point

Examples

```
>>> from sympy import Line, Point
>>> a = Point(1, 2)
>>> b = Point(2, 5)
>>> z = a.origin
>>> p = Point.project(a, b)
>>> Line(p, a).is_perpendicular(Line(p, b))
True
>>> Point.is_collinear(z, p, b)
True
```

See also:

sympy.geometry.line.LinearEntity.projection (page 2225)

taxicab_distance(*p*)

The Taxicab Distance from self to point p.

Returns the sum of the horizontal and vertical distances to point p.

Parameters

p : Point

Returns

taxicab_distance : The sum of the horizontal

and vertical distances to point p.

Examples

```
>>> from sympy import Point
>>> p1, p2 = Point(1, 1), Point(4, 5)
>>> p1.taxicab_distance(p2)
7
```

See also:

sympy.geometry.point.Point.distance (page 2204)

property unit

Return the Point that is in the same direction as $self$ and a distance of 1 from the origin

class sympy.geometry.point.**Point2D**(**args*, *_nocheck=False*, ***kwargs*)

A point in a 2-dimensional Euclidean space.

Parameters

coords : sequence of 2 coordinate values.

Raises

TypeError

When trying to add or subtract points with different dimensions. When trying to create a point with more than two dimensions. When $intersection$ is called with object other than a Point.

Examples

```
>>> from sympy import Point2D
>>> from sympy.abc import x
>>> Point2D(1, 2)
Point2D(1, 2)
>>> Point2D([1, 2])
Point2D(1, 2)
>>> Point2D(0, x)
Point2D(0, x)
```

Floats are automatically converted to Rational unless the evaluate flag is False:

```
>>> Point2D(0.5, 0.25)
Point2D(1/2, 1/4)
>>> Point2D(0.5, 0.25, evaluate=False)
Point2D(0.5, 0.25)
```

See also:

sympy.geometry.line.Segment **(page 2233)**
Connects two Points

Attributes

x	
y	
length	

property bounds

Return a tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the geometric figure.

property coordinates

Returns the two coordinates of the Point.

Examples

```
>>> from sympy import Point2D
>>> p = Point2D(0, 1)
>>> p.coordinates
(0, 1)
```

rotate(*angle*, *pt=None*)

Rotate `angle` radians counterclockwise about Point `pt`.

Examples

```
>>> from sympy import Point2D, pi
>>> t = Point2D(1, 0)
>>> t.rotate(pi/2)
Point2D(0, 1)
>>> t.rotate(pi/2, (2, 0))
Point2D(2, -1)
```

See also:

translate (page 2210), *scale* (page 2210)

scale(*x=1*, *y=1*, *pt=None*)

Scale the coordinates of the Point by multiplying by `x` and `y` after subtracting `pt` – default is (0, 0) – and then adding `pt` back again (i.e. `pt` is the point of reference for the scaling).

Examples

```
>>> from sympy import Point2D
>>> t = Point2D(1, 1)
>>> t.scale(2)
Point2D(2, 1)
>>> t.scale(2, 2)
Point2D(2, 2)
```

See also:

rotate (page 2209), *translate* (page 2210)

transform(*matrix*)

Return the point after applying the transformation described by the 3x3 Matrix, `matrix`.

See also:

sympy.geometry.point.Point2D.rotate (page 2209), *sympy.geometry.point.Point2D.scale* (page 2210), *sympy.geometry.point.Point2D.translate* (page 2210)

translate(*x=0*, *y=0*)

Shift the Point by adding x and y to the coordinates of the Point.

Examples

```
>>> from sympy import Point2D
>>> t = Point2D(0, 1)
>>> t.translate(2)
Point2D(2, 1)
>>> t.translate(2, 2)
Point2D(2, 3)
```

(continues on next page)

(continued from previous page)

```
>>> t + Point2D(2, 2)
Point2D(2, 3)
```

See also:

sympy.geometry.point.Point2D.rotate (page 2209), *scale* (page 2210)

property x

Returns the X coordinate of the Point.

Examples

```
>>> from sympy import Point2D
>>> p = Point2D(0, 1)
>>> p.x
0
```

property y

Returns the Y coordinate of the Point.

Examples

```
>>> from sympy import Point2D
>>> p = Point2D(0, 1)
>>> p.y
1
```

class sympy.geometry.point.**Point3D**(**args*, *_nocheck=False*, ***kwargs*)

A point in a 3-dimensional Euclidean space.

Parameters

coords : sequence of 3 coordinate values.

Raises

TypeError

When trying to add or subtract points with different dimensions. When *intersection* is called with object other than a Point.

Examples

```
>>> from sympy import Point3D
>>> from sympy.abc import x
>>> Point3D(1, 2, 3)
Point3D(1, 2, 3)
>>> Point3D([1, 2, 3])
Point3D(1, 2, 3)
>>> Point3D(0, x, 3)
Point3D(0, x, 3)
```

Floats are automatically converted to Rational unless the evaluate flag is False:

```
>>> Point3D(0.5, 0.25, 2)
Point3D(1/2, 1/4, 2)
>>> Point3D(0.5, 0.25, 3, evaluate=False)
Point3D(0.5, 0.25, 3)
```

Attributes

x	
y	
z	
length	

static are_collinear(**points*)

Is a sequence of points collinear?

Test whether or not a set of points are collinear. Returns True if the set of points are collinear, or False otherwise.

Parameters
points : sequence of Point

Returns
are_collinear : boolean

Examples

```
>>> from sympy import Point3D
>>> from sympy.abc import x
>>> p1, p2 = Point3D(0, 0, 0), Point3D(1, 1, 1)
>>> p3, p4, p5 = Point3D(2, 2, 2), Point3D(x, x, x), Point3D(1, 2, 6)
>>> Point3D.are_collinear(p1, p2, p3, p4)
True
>>> Point3D.are_collinear(p1, p2, p3, p5)
False
```

See also:

sympy.geometry.line.Line3D (page 2243)

property coordinates

Returns the three coordinates of the Point.

Examples

```
>>> from sympy import Point3D
>>> p = Point3D(0, 1, 2)
>>> p.coordinates
(0, 1, 2)
```

direction_cosine(*point*)

Gives the direction cosine between 2 points

Parameters
p : Point3D

Returns
list

Examples

```
>>> from sympy import Point3D
>>> p1 = Point3D(1, 2, 3)
>>> p1.direction_cosine(Point3D(2, 3, 5))
[sqrt(6)/6, sqrt(6)/6, sqrt(6)/3]
```

direction_ratio(*point*)

Gives the direction ratio between 2 points

Parameters
p : Point3D

Returns
list

Examples

```
>>> from sympy import Point3D
>>> p1 = Point3D(1, 2, 3)
>>> p1.direction_ratio(Point3D(2, 3, 5))
[1, 1, 2]
```

intersection(*other*)

The intersection between this point and another GeometryEntity.

Parameters
other : GeometryEntity or sequence of coordinates

Returns
intersection : list of Points

Notes

The return value will either be an empty list if there is no intersection, otherwise it will contain this point.

Examples

```
>>> from sympy import Point3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(1, 1, 1), Point3D(0, 0, 0)
>>> p1.intersection(p2)
[]
>>> p1.intersection(p3)
[Point3D(0, 0, 0)]
```

scale(*x=1*, *y=1*, *z=1*, *pt=None*)

Scale the coordinates of the Point by multiplying by `x` and `y` after subtracting `pt` – default is (0, 0) – and then adding `pt` back again (i.e. `pt` is the point of reference for the scaling).

Examples

```
>>> from sympy import Point3D
>>> t = Point3D(1, 1, 1)
>>> t.scale(2)
Point3D(2, 1, 1)
>>> t.scale(2, 2)
Point3D(2, 2, 1)
```

See also:

translate (page 2214)

transform(*matrix*)

Return the point after applying the transformation described by the 4x4 Matrix, `matrix`.

See also:

sympy.geometry.point.Point3D.scale (page 2214), *sympy.geometry.point.Point3D.translate* (page 2214)

translate(*x=0*, *y=0*, *z=0*)

Shift the Point by adding x and y to the coordinates of the Point.

Examples

```
>>> from sympy import Point3D
>>> t = Point3D(0, 1, 1)
>>> t.translate(2)
Point3D(2, 1, 1)
>>> t.translate(2, 2)
Point3D(2, 3, 1)
>>> t + Point3D(2, 2, 2)
Point3D(2, 3, 3)
```

See also:

scale (page 2214)

property x

Returns the X coordinate of the Point.

Examples

```
>>> from sympy import Point3D
>>> p = Point3D(0, 1, 3)
>>> p.x
0
```

property y

Returns the Y coordinate of the Point.

Examples

```
>>> from sympy import Point3D
>>> p = Point3D(0, 1, 2)
>>> p.y
1
```

property z

Returns the Z coordinate of the Point.

Examples

```
>>> from sympy import Point3D
>>> p = Point3D(0, 1, 1)
>>> p.z
1
```

Lines

class sympy.geometry.line.**LinearEntity**(*p1*, *p2=None*, ***kwargs*)

A base class for all linear entities (Line, Ray and Segment) in n-dimensional Euclidean space.

Notes

This is an abstract class and is not meant to be instantiated.

See also:

sympy.geometry.entity.GeometryEntity (page 2194)

Attributes

ambient_dimension	
direction	
length	
p1	
p2	
points	

property ambient_dimension

A property method that returns the dimension of LinearEntity object.

Parameters
p1 : LinearEntity

Returns
dimension : integer

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(1, 1)
>>> l1 = Line(p1, p2)
>>> l1.ambient_dimension
2
```

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0, 0), Point(1, 1, 1)
>>> l1 = Line(p1, p2)
>>> l1.ambient_dimension
3
```

angle_between(*l2*)

Return the non-reflex angle formed by rays emanating from the origin with directions the same as the direction vectors of the linear entities.

Parameters
l1 : LinearEntity

l2 : LinearEntity

Returns
angle : angle in radians

Notes

From the dot product of vectors v1 and v2 it is known that:

```
dot(v1, v2) = |v1|*|v2|*cos(A)
```

where A is the angle formed between the two vectors. We can get the directional vectors of the two lines and readily find the angle between the two using the above formula.

Examples

```
>>> from sympy import Line
>>> e = Line((0, 0), (1, 0))
>>> ne = Line((0, 0), (1, 1))
>>> sw = Line((1, 1), (0, 0))
>>> ne.angle_between(e)
pi/4
>>> sw.angle_between(e)
3*pi/4
```

To obtain the non-obtuse angle at the intersection of lines, use the `smallest_angle_between` method:

```
>>> sw.smallest_angle_between(e)
pi/4
```

```
>>> from sympy import Point3D, Line3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(1, 1, 1), Point3D(-1, 2, 0)
>>> l1, l2 = Line3D(p1, p2), Line3D(p2, p3)
>>> l1.angle_between(l2)
acos(-sqrt(2)/3)
>>> l1.smallest_angle_between(l2)
acos(sqrt(2)/3)
```

See also:

is_perpendicular (page 2221), *Ray2D.closing_angle* (page 2240)

arbitrary_point(*parameter='t'*)

A parameterized point on the Line.

Parameters
parameter : str, optional

The name of the parameter which will be used for the parametric point. The default value is 't'. When this parameter is 0, the first

point used to define the line will be returned, and when it is 1 the second point will be returned.

Returns

point : Point

Raises

ValueError

When `parameter` already appears in the Line's definition.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(1, 0), Point(5, 3)
>>> l1 = Line(p1, p2)
>>> l1.arbitrary_point()
Point2D(4*t + 1, 3*t)
>>> from sympy import Point3D, Line3D
>>> p1, p2 = Point3D(1, 0, 0), Point3D(5, 3, 1)
>>> l1 = Line3D(p1, p2)
>>> l1.arbitrary_point()
Point3D(4*t + 1, 3*t, t)
```

See also:

sympy.geometry.point.Point (page 2202)

static are_concurrent(**lines*)

Is a sequence of linear entities concurrent?

Two or more linear entities are concurrent if they all intersect at a single point.

Parameters

lines : a sequence of linear entities.

Returns

True : if the set of linear entities intersect in one point

False : otherwise.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(3, 5)
>>> p3, p4 = Point(-2, -2), Point(0, 2)
>>> l1, l2, l3 = Line(p1, p2), Line(p1, p3), Line(p1, p4)
>>> Line.are_concurrent(l1, l2, l3)
True
>>> l4 = Line(p2, p3)
>>> Line.are_concurrent(l2, l3, l4)
False
>>> from sympy import Point3D, Line3D
>>> p1, p2 = Point3D(0, 0, 0), Point3D(3, 5, 2)
>>> p3, p4 = Point3D(-2, -2, -2), Point3D(0, 2, 1)
```

(continues on next page)

(continued from previous page)

```
>>> l1, l2, l3 = Line3D(p1, p2), Line3D(p1, p3), Line3D(p1, p4)
>>> Line3D.are_concurrent(l1, l2, l3)
True
>>> l4 = Line3D(p2, p3)
>>> Line3D.are_concurrent(l2, l3, l4)
False
```

See also:

sympy.geometry.util.intersection (page 2197)

bisectors(*other*)

Returns the perpendicular lines which pass through the intersections of self and other that are in the same plane.

Parameters
line : Line3D

Returns
list: two Line instances

Examples

```
>>> from sympy import Point3D, Line3D
>>> r1 = Line3D(Point3D(0, 0, 0), Point3D(1, 0, 0))
>>> r2 = Line3D(Point3D(0, 0, 0), Point3D(0, 1, 0))
>>> r1.bisectors(r2)
[Line3D(Point3D(0, 0, 0), Point3D(1, 1, 0)), Line3D(Point3D(0, 0, 0),␣
↪Point3D(1, -1, 0))]
```

contains(*other*)

Subclasses should implement this method and should return True if other is on the boundaries of self; False if not on the boundaries of self; None if a determination cannot be made.

property direction

The direction vector of the LinearEntity.

Returns
p : a Point; the ray from the origin to this point is the

direction of $self$

Examples

```
>>> from sympy import Line
>>> a, b = (1, 1), (1, 3)
>>> Line(a, b).direction
Point2D(0, 2)
>>> Line(b, a).direction
Point2D(0, -2)
```

This can be reported so the distance from the origin is 1:

```
>>> Line(b, a).direction.unit
Point2D(0, -1)
```

See also:

sympy.geometry.point.Point.unit (page 2208)

intersection(*other*)

The intersection with another geometrical entity.

Parameters

o : Point or LinearEntity

Returns

intersection : list of geometrical entities

Examples

```
>>> from sympy import Point, Line, Segment
>>> p1, p2, p3 = Point(0, 0), Point(1, 1), Point(7, 7)
>>> l1 = Line(p1, p2)
>>> l1.intersection(p3)
[Point2D(7, 7)]
>>> p4, p5 = Point(5, 0), Point(0, 3)
>>> l2 = Line(p4, p5)
>>> l1.intersection(l2)
[Point2D(15/8, 15/8)]
>>> p6, p7 = Point(0, 5), Point(2, 6)
>>> s1 = Segment(p6, p7)
>>> l1.intersection(s1)
[]
>>> from sympy import Point3D, Line3D, Segment3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(1, 1, 1), Point3D(7, 7, 7)
>>> l1 = Line3D(p1, p2)
>>> l1.intersection(p3)
[Point3D(7, 7, 7)]
>>> l1 = Line3D(Point3D(4,19,12), Point3D(5,25,17))
>>> l2 = Line3D(Point3D(-3, -15, -19), direction_ratio=[2,8,8])
>>> l1.intersection(l2)
[Point3D(1, 1, -3)]
>>> p6, p7 = Point3D(0, 5, 2), Point3D(2, 6, 3)
>>> s1 = Segment3D(p6, p7)
>>> l1.intersection(s1)
[]
```

See also:

sympy.geometry.point.Point (page 2202)

is_parallel(*l2*)

Are two linear entities parallel?

Parameters

l1 : LinearEntity

l2 : LinearEntity

Returns

True : if l1 and l2 are parallel,

False : otherwise.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(1, 1)
>>> p3, p4 = Point(3, 4), Point(6, 7)
>>> l1, l2 = Line(p1, p2), Line(p3, p4)
>>> Line.is_parallel(l1, l2)
True
>>> p5 = Point(6, 6)
>>> l3 = Line(p3, p5)
>>> Line.is_parallel(l1, l3)
False
>>> from sympy import Point3D, Line3D
>>> p1, p2 = Point3D(0, 0, 0), Point3D(3, 4, 5)
>>> p3, p4 = Point3D(2, 1, 1), Point3D(8, 9, 11)
>>> l1, l2 = Line3D(p1, p2), Line3D(p3, p4)
>>> Line3D.is_parallel(l1, l2)
True
>>> p5 = Point3D(6, 6, 6)
>>> l3 = Line3D(p3, p5)
>>> Line3D.is_parallel(l1, l3)
False
```

See also:

coefficients (page 2239)

is_perpendicular(*l2*)

Are two linear entities perpendicular?

Parameters

l1 : LinearEntity

l2 : LinearEntity

Returns

True : if l1 and l2 are perpendicular,

False : otherwise.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2, p3 = Point(0, 0), Point(1, 1), Point(-1, 1)
>>> l1, l2 = Line(p1, p2), Line(p1, p3)
>>> l1.is_perpendicular(l2)
True
>>> p4 = Point(5, 3)
>>> l3 = Line(p1, p4)
```

(continues on next page)

(continued from previous page)

```
>>> l1.is_perpendicular(l3)
False
>>> from sympy import Point3D, Line3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(1, 1, 1), Point3D(-1, 2, 0)
>>> l1, l2 = Line3D(p1, p2), Line3D(p2, p3)
>>> l1.is_perpendicular(l2)
False
>>> p4 = Point3D(5, 3, 7)
>>> l3 = Line3D(p1, p4)
>>> l1.is_perpendicular(l3)
False
```

See also:

coefficients (page 2239)

is_similar(*other*)

Return True if self and other are contained in the same line.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2, p3 = Point(0, 1), Point(3, 4), Point(2, 3)
>>> l1 = Line(p1, p2)
>>> l2 = Line(p1, p3)
>>> l1.is_similar(l2)
True
```

property length

The length of the line.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(3, 5)
>>> l1 = Line(p1, p2)
>>> l1.length
oo
```

property p1

The first defining point of a linear entity.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(5, 3)
>>> l = Line(p1, p2)
>>> l.p1
Point2D(0, 0)
```

See also:

sympy.geometry.point.Point (page 2202)

property p2

The second defining point of a linear entity.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(5, 3)
>>> l = Line(p1, p2)
>>> l.p2
Point2D(5, 3)
```

See also:

sympy.geometry.point.Point (page 2202)

parallel_line(p)

Create a new Line parallel to this linear entity which passes through the point p.

Parameters
p : Point

Returns
line : Line

Examples

```
>>> from sympy import Point, Line
>>> p1, p2, p3 = Point(0, 0), Point(2, 3), Point(-2, 2)
>>> l1 = Line(p1, p2)
>>> l2 = l1.parallel_line(p3)
>>> p3 in l2
True
>>> l1.is_parallel(l2)
True
>>> from sympy import Point3D, Line3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(2, 3, 4), Point3D(-2, 2, 0)
>>> l1 = Line3D(p1, p2)
>>> l2 = l1.parallel_line(p3)
>>> p3 in l2
True
```

(continues on next page)

(continued from previous page)

```
>>> l1.is_parallel(l2)
True
```

See also:

is_parallel (page 2220)

perpendicular_line(*p*)

Create a new Line perpendicular to this linear entity which passes through the point *p*.

Parameters
p : Point

Returns
line : Line

Examples

```
>>> from sympy import Point3D, Line3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(2, 3, 4), Point3D(-2, 2, 0)
>>> L = Line3D(p1, p2)
>>> P = L.perpendicular_line(p3); P
Line3D(Point3D(-2, 2, 0), Point3D(4/29, 6/29, 8/29))
>>> L.is_perpendicular(P)
True
```

In 3D the, the first point used to define the line is the point through which the perpendicular was required to pass; the second point is (arbitrarily) contained in the given line:

```
>>> P.p2 in L
True
```

See also:

sympy.geometry.line.LinearEntity.is_perpendicular (page 2221), *perpendicular_segment* (page 2224)

perpendicular_segment(*p*)

Create a perpendicular line segment from *p* to this line.

The enpoints of the segment are `p` and the closest point in the line containing self. (If self is not a line, the point might not be in self.)

Parameters
p : Point

Returns
segment : Segment

Notes

Returns p itself if p is on this linear entity.

Examples

```
>>> from sympy import Point, Line
>>> p1, p2, p3 = Point(0, 0), Point(1, 1), Point(0, 2)
>>> l1 = Line(p1, p2)
>>> s1 = l1.perpendicular_segment(p3)
>>> l1.is_perpendicular(s1)
True
>>> p3 in s1
True
>>> l1.perpendicular_segment(Point(4, 0))
Segment2D(Point2D(4, 0), Point2D(2, 2))
>>> from sympy import Point3D, Line3D
>>> p1, p2, p3 = Point3D(0, 0, 0), Point3D(1, 1, 1), Point3D(0, 2, 0)
>>> l1 = Line3D(p1, p2)
>>> s1 = l1.perpendicular_segment(p3)
>>> l1.is_perpendicular(s1)
True
>>> p3 in s1
True
>>> l1.perpendicular_segment(Point3D(4, 0, 0))
Segment3D(Point3D(4, 0, 0), Point3D(4/3, 4/3, 4/3))
```

See also:

perpendicular_line (page 2224)

property points

The two points used to define this linear entity.

Returns

points : tuple of Points

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(5, 11)
>>> l1 = Line(p1, p2)
>>> l1.points
(Point2D(0, 0), Point2D(5, 11))
```

See also:

sympy.geometry.point.Point (page 2202)

projection(*other*)

Project a point, line, ray, or segment onto this linear entity.

Parameters

other : Point or LinearEntity (Line, Ray, Segment)

Returns

projection : Point or LinearEntity (Line, Ray, Segment)

The return type matches the type of the parameter `other`.

Raises

GeometryError

When method is unable to perform projection.

Notes

A projection involves taking the two points that define the linear entity and projecting those points onto a Line and then reforming the linear entity using these projections. A point P is projected onto a line L by finding the point on L that is closest to P. This point is the intersection of L and the line perpendicular to L that passes through P.

Examples

```
>>> from sympy import Point, Line, Segment, Rational
>>> p1, p2, p3 = Point(0, 0), Point(1, 1), Point(Rational(1, 2), 0)
>>> l1 = Line(p1, p2)
>>> l1.projection(p3)
Point2D(1/4, 1/4)
>>> p4, p5 = Point(10, 0), Point(12, 1)
>>> s1 = Segment(p4, p5)
>>> l1.projection(s1)
Segment2D(Point2D(5, 5), Point2D(13/2, 13/2))
>>> p1, p2, p3 = Point(0, 0, 1), Point(1, 1, 2), Point(2, 0, 1)
>>> l1 = Line(p1, p2)
>>> l1.projection(p3)
Point3D(2/3, 2/3, 5/3)
>>> p4, p5 = Point(10, 0, 1), Point(12, 1, 3)
>>> s1 = Segment(p4, p5)
>>> l1.projection(s1)
Segment3D(Point3D(10/3, 10/3, 13/3), Point3D(5, 5, 6))
```

See also:

sympy.geometry.point.Point (page 2202), *perpendicular_line* (page 2224)

random_point(*seed=None*)

A random point on a LinearEntity.

Returns

point : Point

Examples

```
>>> from sympy import Point, Line, Ray, Segment
>>> p1, p2 = Point(0, 0), Point(5, 3)
>>> line = Line(p1, p2)
>>> r = line.random_point(seed=42)  # seed value is optional
>>> r.n(3)
Point2D(-0.72, -0.432)
>>> r in line
True
>>> Ray(p1, p2).random_point(seed=42).n(3)
Point2D(0.72, 0.432)
>>> Segment(p1, p2).random_point(seed=42).n(3)
Point2D(3.2, 1.92)
```

See also:

sympy.geometry.point.Point (page 2202)

smallest_angle_between(*l2*)

Return the smallest angle formed at the intersection of the lines containing the linear entities.

Parameters

l1 : LinearEntity

l2 : LinearEntity

Returns

angle : angle in radians

Examples

```
>>> from sympy import Point, Line
>>> p1, p2, p3 = Point(0, 0), Point(0, 4), Point(2, -2)
>>> l1, l2 = Line(p1, p2), Line(p1, p3)
>>> l1.smallest_angle_between(l2)
pi/4
```

See also:

angle_between (page 2216), *Ray2D.closing_angle* (page 2240)

class sympy.geometry.line.**Line**(**args*, ***kwargs*)

An infinite line in space.

A 2D line is declared with two distinct points, point and slope, or an equation. A 3D line may be defined with a point and a direction ratio.

Parameters

p1 : Point

p2 : Point

slope : SymPy expression

direction_ratio : list

equation : equation of a line

Notes

$Line$ will automatically subclass to $Line2D$ or $Line3D$ based on the dimension of $p1$. The $slope$ argument is only relevant for $Line2D$ and the $direction_ratio$ argument is only relevant for $Line3D$.

The order of the points will define the direction of the line which is used when calculating the angle between lines.

Examples

```
>>> from sympy import Line, Segment, Point, Eq
>>> from sympy.abc import x, y, a, b
```

```
>>> L = Line(Point(2,3), Point(3,5))
>>> L
Line2D(Point2D(2, 3), Point2D(3, 5))
>>> L.points
(Point2D(2, 3), Point2D(3, 5))
>>> L.equation()
-2*x + y + 1
>>> L.coefficients
(-2, 1, 1)
```

Instantiate with keyword `slope`:

```
>>> Line(Point(0, 0), slope=0)
Line2D(Point2D(0, 0), Point2D(1, 0))
```

Instantiate with another linear object

```
>>> s = Segment((0, 0), (0, 1))
>>> Line(s).equation()
x
```

The line corresponding to an equation in the for $ax + by + c = 0$, can be entered:

```
>>> Line(3*x + y + 18)
Line2D(Point2D(0, -18), Point2D(1, -21))
```

If x or y has a different name, then they can be specified, too, as a string (to match the name) or symbol:

```
>>> Line(Eq(3*a + b, -18), x='a', y=b)
Line2D(Point2D(0, -18), Point2D(1, -21))
```

See also:

sympy.geometry.point.Point (page 2202), *sympy.geometry.line.Line2D* (page 2238), *sympy.geometry.line.Line3D* (page 2243)

contains(*other*)

Return True if *other* is on this Line, or False otherwise.

Examples

```
>>> from sympy import Line,Point
>>> p1, p2 = Point(0, 1), Point(3, 4)
>>> l = Line(p1, p2)
>>> l.contains(p1)
True
>>> l.contains((0, 1))
True
>>> l.contains((0, 0))
False
>>> a = (0, 0, 0)
>>> b = (1, 1, 1)
>>> c = (2, 2, 2)
>>> l1 = Line(a, b)
>>> l2 = Line(b, a)
>>> l1 == l2
False
>>> l1 in l2
True
```

distance(*other*)

Finds the shortest distance between a line and a point.

Raises

NotImplementedError is raised if `other` is not a Point

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(1, 1)
>>> s = Line(p1, p2)
>>> s.distance(Point(-1, 1))
sqrt(2)
>>> s.distance((-1, 2))
3*sqrt(2)/2
>>> p1, p2 = Point(0, 0, 0), Point(1, 1, 1)
>>> s = Line(p1, p2)
>>> s.distance(Point(-1, 1, 1))
2*sqrt(6)/3
>>> s.distance((-1, 1, 1))
2*sqrt(6)/3
```

equals(*other*)

Returns True if self and other are the same mathematical entities

plot_interval(*parameter='t'*)

The plot interval for the default geometric plot of line. Gives values that will produce

a line that is +/- 5 units long (where a unit is the distance between the two points that define the line).

Parameters

parameter : str, optional

Default value is 't'.

Returns

plot_interval : list (plot interval)

[parameter, lower_bound, upper_bound]

Examples

```
>>> from sympy import Point, Line
>>> p1, p2 = Point(0, 0), Point(5, 3)
>>> l1 = Line(p1, p2)
>>> l1.plot_interval()
[t, -5, 5]
```

class sympy.geometry.line.**Ray**(*p1*, *p2=None*, ***kwargs*)

A Ray is a semi-line in the space with a source point and a direction.

Parameters

p1 : Point

The source of the Ray

p2 : Point or radian value

This point determines the direction in which the Ray propagates. If given as an angle it is interpreted in radians with the positive direction being ccw.

Notes

Ray will automatically subclass to $Ray2D$ or $Ray3D$ based on the dimension of $p1$.

Examples

```
>>> from sympy import Ray, Point, pi
>>> r = Ray(Point(2, 3), Point(3, 5))
>>> r
Ray2D(Point2D(2, 3), Point2D(3, 5))
>>> r.points
(Point2D(2, 3), Point2D(3, 5))
>>> r.source
Point2D(2, 3)
>>> r.xdirection
oo
>>> r.ydirection
oo
```

(continues on next page)

(continued from previous page)

```
>>> r.slope
2
>>> Ray(Point(0, 0), angle=pi/4).slope
1
```

See also:

sympy.geometry.line.Ray2D (page 2240), *sympy.geometry.line.Ray3D* (page 2244), *sympy.geometry.point.Point* (page 2202), *sympy.geometry.line.Line* (page 2227)

Attributes

source

contains(*other*)

Is other GeometryEntity contained in this Ray?

Examples

```
>>> from sympy import Ray,Point,Segment
>>> p1, p2 = Point(0, 0), Point(4, 4)
>>> r = Ray(p1, p2)
>>> r.contains(p1)
True
>>> r.contains((1, 1))
True
>>> r.contains((1, 3))
False
>>> s = Segment((1, 1), (2, 2))
>>> r.contains(s)
True
>>> s = Segment((1, 2), (2, 5))
>>> r.contains(s)
False
>>> r1 = Ray((2, 2), (3, 3))
>>> r.contains(r1)
True
>>> r1 = Ray((2, 2), (3, 5))
>>> r.contains(r1)
False
```

distance(*other*)

Finds the shortest distance between the ray and a point.

Raises

NotImplementedError is raised if `other` is not a Point

Examples

```
>>> from sympy import Point, Ray
>>> p1, p2 = Point(0, 0), Point(1, 1)
>>> s = Ray(p1, p2)
>>> s.distance(Point(-1, -1))
sqrt(2)
>>> s.distance((-1, 2))
3*sqrt(2)/2
>>> p1, p2 = Point(0, 0, 0), Point(1, 1, 2)
>>> s = Ray(p1, p2)
>>> s
Ray3D(Point3D(0, 0, 0), Point3D(1, 1, 2))
>>> s.distance(Point(-1, -1, 2))
4*sqrt(3)/3
>>> s.distance((-1, -1, 2))
4*sqrt(3)/3
```

equals(*other*)

Returns True if self and other are the same mathematical entities

plot_interval(*parameter='t'*)

The plot interval for the default geometric plot of the Ray. Gives values that will produce a ray that is 10 units long (where a unit is the distance between the two points that define the ray).

Parameters

parameter : str, optional

Default value is 't'.

Returns

plot_interval : list

[parameter, lower_bound, upper_bound]

Examples

```
>>> from sympy import Ray, pi
>>> r = Ray((0, 0), angle=pi/4)
>>> r.plot_interval()
[t, 0, 10]
```

property source

The point from which the ray emanates.

Examples

```
>>> from sympy import Point, Ray
>>> p1, p2 = Point(0, 0), Point(4, 1)
>>> r1 = Ray(p1, p2)
>>> r1.source
Point2D(0, 0)
>>> p1, p2 = Point(0, 0, 0), Point(4, 1, 5)
>>> r1 = Ray(p2, p1)
>>> r1.source
Point3D(4, 1, 5)
```

See also:

sympy.geometry.point.Point (page 2202)

class sympy.geometry.line.**Segment**(*p1*, *p2*, ***kwargs*)

A line segment in space.

Parameters

p1 : Point

p2 : Point

Notes

If 2D or 3D points are used to define $Segment$, it will be automatically subclassed to $Segment2D$ or $Segment3D$.

Examples

```
>>> from sympy import Point, Segment
>>> Segment((1, 0), (1, 1)) # tuples are interpreted as pts
Segment2D(Point2D(1, 0), Point2D(1, 1))
>>> s = Segment(Point(4, 3), Point(1, 1))
>>> s.points
(Point2D(4, 3), Point2D(1, 1))
>>> s.slope
2/3
>>> s.length
sqrt(13)
>>> s.midpoint
Point2D(5/2, 2)
>>> Segment((1, 0, 0), (1, 1, 1)) # tuples are interpreted as pts
Segment3D(Point3D(1, 0, 0), Point3D(1, 1, 1))
>>> s = Segment(Point(4, 3, 9), Point(1, 1, 7)); s
Segment3D(Point3D(4, 3, 9), Point3D(1, 1, 7))
>>> s.points
(Point3D(4, 3, 9), Point3D(1, 1, 7))
>>> s.length
sqrt(17)
>>> s.midpoint
Point3D(5/2, 2, 8)
```

See also:

sympy.geometry.line.Segment2D (page 2242), *sympy.geometry.line.Segment3D* (page 2247), *sympy.geometry.point.Point* (page 2202), *sympy.geometry.line.Line* (page 2227)

Attributes

length	(number or SymPy expression)
midpoint	(Point)

contains(*other*)

Is the other GeometryEntity contained within this Segment?

Examples

```
>>> from sympy import Point, Segment
>>> p1, p2 = Point(0, 1), Point(3, 4)
>>> s = Segment(p1, p2)
>>> s2 = Segment(p2, p1)
>>> s.contains(s2)
True
>>> from sympy import Point3D, Segment3D
>>> p1, p2 = Point3D(0, 1, 1), Point3D(3, 4, 5)
>>> s = Segment3D(p1, p2)
>>> s2 = Segment3D(p2, p1)
>>> s.contains(s2)
True
>>> s.contains((p1 + p2)/2)
True
```

distance(*other*)

Finds the shortest distance between a line segment and a point.

Raises

NotImplementedError is raised if `other` is not a Point

Examples

```
>>> from sympy import Point, Segment
>>> p1, p2 = Point(0, 1), Point(3, 4)
>>> s = Segment(p1, p2)
>>> s.distance(Point(10, 15))
sqrt(170)
>>> s.distance((0, 12))
sqrt(73)
>>> from sympy import Point3D, Segment3D
>>> p1, p2 = Point3D(0, 0, 3), Point3D(1, 1, 4)
>>> s = Segment3D(p1, p2)
```

(continues on next page)

(continued from previous page)

```
>>> s.distance(Point3D(10, 15, 12))
sqrt(341)
>>> s.distance((10, 15, 12))
sqrt(341)
```

equals(*other*)

Returns True if self and other are the same mathematical entities

property length

The length of the line segment.

Examples

```
>>> from sympy import Point, Segment
>>> p1, p2 = Point(0, 0), Point(4, 3)
>>> s1 = Segment(p1, p2)
>>> s1.length
5
>>> from sympy import Point3D, Segment3D
>>> p1, p2 = Point3D(0, 0, 0), Point3D(4, 3, 3)
>>> s1 = Segment3D(p1, p2)
>>> s1.length
sqrt(34)
```

See also:

sympy.geometry.point.Point.distance (page 2204)

property midpoint

The midpoint of the line segment.

Examples

```
>>> from sympy import Point, Segment
>>> p1, p2 = Point(0, 0), Point(4, 3)
>>> s1 = Segment(p1, p2)
>>> s1.midpoint
Point2D(2, 3/2)
>>> from sympy import Point3D, Segment3D
>>> p1, p2 = Point3D(0, 0, 0), Point3D(4, 3, 3)
>>> s1 = Segment3D(p1, p2)
>>> s1.midpoint
Point3D(2, 3/2, 3/2)
```

See also:

sympy.geometry.point.Point.midpoint (page 2207)

perpendicular_bisector(*p=None*)

The perpendicular bisector of this segment.

If no point is specified or the point specified is not on the bisector then the bisector is returned as a Line. Otherwise a Segment is returned that joins the point specified and the intersection of the bisector and the segment.

Parameters
p : Point

Returns
bisector : Line or Segment

Examples

```
>>> from sympy import Point, Segment
>>> p1, p2, p3 = Point(0, 0), Point(6, 6), Point(5, 1)
>>> s1 = Segment(p1, p2)
>>> s1.perpendicular_bisector()
Line2D(Point2D(3, 3), Point2D(-3, 9))
```

```
>>> s1.perpendicular_bisector(p3)
Segment2D(Point2D(5, 1), Point2D(3, 3))
```

See also:

LinearEntity.perpendicular_segment (page 2224)

plot_interval(*parameter='t'*)

The plot interval for the default geometric plot of the Segment gives values that will produce the full segment in a plot.

Parameters
parameter : str, optional

Default value is 't'.

Returns
plot_interval : list

[parameter, lower_bound, upper_bound]

Examples

```
>>> from sympy import Point, Segment
>>> p1, p2 = Point(0, 0), Point(5, 3)
>>> s1 = Segment(p1, p2)
>>> s1.plot_interval()
[t, 0, 1]
```

class sympy.geometry.line.**LinearEntity2D**(*p1*, *p2=None*, ***kwargs*)

A base class for all linear entities (line, ray and segment) in a 2-dimensional Euclidean space.