hypothesis_geometry

Release 1.0.0

Azat Ibrakov

Jan 13, 2021

CONTENTS

Python Module Index 27

Index 29

Note: If object is not listed in documentation it should be considered as implementation detail that can change and should not be relied upon.

hypothesis_geometry.planar.**points**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Point]

Returns a strategy for points.

Parameters

- **x_coordinates** – strategy for points' x-coordinates.
- **y_coordinates** – strategy for points' y-coordinates, `None` for reusing x-coordinates strategy.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Point = context.point_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> points = planar.points(coordinates)
>>> point = points.example()
>>> isinstance(point, Point)
True
>>> (isinstance(point.x, coordinates_type)
...  and isinstance(point.y, coordinates_type))
True
>>> (min_coordinate <= point.x <= max_coordinate
...  and min_coordinate <= point.y <= max_coordinate)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> points = planar.points(x_coordinates, y_coordinates)
>>> point = points.example()
>>> isinstance(point, Point)
True
>>> (isinstance(point.x, coordinates_type)
...  and isinstance(point.y, coordinates_type))
```

(continues on next page)

(continued from previous page)

```
True
>>> (min_x_coordinate <= point.x <= max_x_coordinate
...  and min_y_coordinate <= point.y <= max_y_coordinate)
True
```

hypothesis_geometry.planar.**multipoints**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*, ***, *min_size: int = 0*, *max_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Multipoint]

Returns a strategy for multipoints.

Parameters

- **x_coordinates** – strategy for points' x-coordinates.
- **y_coordinates** – strategy for points' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for multipoint size.
- **max_size** – upper bound for multipoint size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Multipoint = context.multipoint_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> multipoints = planar.multipoints(coordinates,
...                                  min_size=min_size,
...                                  max_size=max_size)
>>> multipoint = multipoints.example()
>>> isinstance(multipoint, Multipoint)
True
>>> min_size <= len(multipoint.points) <= max_size
True
>>> all(isinstance(point.x, coordinates_type)
...     and isinstance(point.y, coordinates_type)
...     for point in multipoint.points)
True
>>> all(min_coordinate <= point.x <= max_coordinate
...     and min_coordinate <= point.y <= max_coordinate
...     for point in multipoint.points)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 5, 10
>>> multipoints = planar.multipoints(x_coordinates, y_coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size)
>>> multipoint = multipoints.example()
>>> isinstance(multipoint, Multipoint)
True
>>> min_size <= len(multipoint.points) <= max_size
True
>>> all(isinstance(point.x, coordinates_type)
...     and isinstance(point.y, coordinates_type)
...     for point in multipoint.points)
True
>>> all(min_x_coordinate <= point.x <= max_x_coordinate
...     and min_y_coordinate <= point.y <= max_y_coordinate
...     for point in multipoint.points)
True
```

hypothesis_geometry.planar.**segments**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Segment]

Returns a strategy for segments.

Parameters

- **x_coordinates** – strategy for endpoints' x-coordinates.
- **y_coordinates** – strategy for endpoints' y-coordinates, `None` for reusing x-coordinates strategy.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Segment = context.segment_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> segments = planar.segments(coordinates)
>>> segment = segments.example()
>>> isinstance(segment, Segment)
True
```

(continues on next page)

(continued from previous page)

```
>>> (isinstance(segment.start.x, coordinates_type)
...  and isinstance(segment.start.y, coordinates_type)
...  and isinstance(segment.end.x, coordinates_type)
...  and isinstance(segment.end.y, coordinates_type))
True
>>> (min_coordinate <= segment.start.x <= max_coordinate
...  and min_coordinate <= segment.start.y <= max_coordinate
...  and min_coordinate <= segment.end.x <= max_coordinate
...  and min_coordinate <= segment.end.y <= max_coordinate)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> segments = planar.segments(x_coordinates, y_coordinates)
>>> segment = segments.example()
>>> isinstance(segment, Segment)
True
>>> (isinstance(segment.start.x, coordinates_type)
...  and isinstance(segment.start.y, coordinates_type)
...  and isinstance(segment.end.x, coordinates_type)
...  and isinstance(segment.end.y, coordinates_type))
True
>>> (min_x_coordinate <= segment.start.x <= max_x_coordinate
...  and min_y_coordinate <= segment.start.y <= max_y_coordinate
...  and min_x_coordinate <= segment.end.x <= max_x_coordinate
...  and min_y_coordinate <= segment.end.y <= max_y_coordinate)
True
```

hypothesis_geometry.planar.**multisegments**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*, ***, *min_size: int = 0*, *max_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Multisegment]

Returns a strategy for multisegments.

Parameters

- **x_coordinates** – strategy for segments' x-coordinates.
- **y_coordinates** – strategy for segments' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for multisegment size.
- **max_size** – upper bound for multisegment size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Multisegment = context.multisegment_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> multisegments = planar.multisegments(coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size)
>>> multisegment = multisegments.example()
>>> isinstance(multisegment, Multisegment)
True
>>> min_size <= len(multisegment.segments) <= max_size
True
>>> all(isinstance(segment.start.x, coordinates_type)
...     and isinstance(segment.start.y, coordinates_type)
...     and isinstance(segment.start.x, coordinates_type)
...     and isinstance(segment.start.y, coordinates_type)
...     for segment in multisegment.segments)
True
>>> all(min_coordinate <= segment.start.x <= max_coordinate
...     and min_coordinate <= segment.start.y <= max_coordinate
...     and min_coordinate <= segment.end.x <= max_coordinate
...     and min_coordinate <= segment.end.y <= max_coordinate
...     for segment in multisegment.segments)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 5, 10
>>> multisegments = planar.multisegments(x_coordinates, y_coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size)
>>> multisegment = multisegments.example()
>>> isinstance(multisegment, Multisegment)
True
>>> min_size <= len(multisegment.segments) <= max_size
True
>>> all(isinstance(segment.start.x, coordinates_type)
...     and isinstance(segment.start.y, coordinates_type)
```

(continues on next page)

(continued from previous page)

```
...     and isinstance(segment.start.x, coordinates_type)
...     and isinstance(segment.start.y, coordinates_type)
...     for segment in multisegment.segments)
True
>>> all(min_x_coordinate <= segment.start.x <= max_x_coordinate
...     and min_y_coordinate <= segment.start.y <= max_y_coordinate
...     and min_x_coordinate <= segment.end.x <= max_x_coordinate
...     and min_y_coordinate <= segment.end.y <= max_y_coordinate
...     for segment in multisegment.segments)
True
```

hypothesis_geometry.planar.**contours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate], y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None, *, min_size: int = <MinContourSize.CONVEX: 3>, max_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Contour]

Returns a strategy for contours.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for contour size.
- **max_size** – upper bound for contour size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.contours(coordinates,
...                            min_size=min_size,
...                            max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
```

(continues on next page)

(continued from previous page)

```
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.contours(x_coordinates, y_coordinates,
...                            min_size=min_size,
...                            max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**convex_contours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*, ***, *min_size: int = <MinContourSize.CONVEX: 3>*, *max_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Contour]

Returns a strategy for convex contours. Convex contour is a contour such that the line segment formed by any two points from contour's line segments stays inside the region bounded by the contour.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for contour size.
- **max_size** – upper bound for contour size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
```

(continues on next page)

(continued from previous page)

```
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.convex_contours(coordinates,
...                                   min_size=min_size,
...                                   max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.convex_contours(x_coordinates, y_coordinates,
...                                   min_size=min_size,
...                                   max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**concave_contours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate], y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None, *, min_size: int = <MinContourSize.CONCAVE: 4>, max_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Contour]

Returns a strategy for concave contours. Concave contour is a contour that is not convex.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for contour size.
- **max_size** – upper bound for contour size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.concave_contours(coordinates,
...                                    min_size=min_size,
...                                    max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
```

(continues on next page)

(continued from previous page)

```
...                                     allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                     allow_infinity=False,
...                                     allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.concave_contours(x_coordinates, y_coordinates,
...                                     min_size=min_size,
...                                     max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**triangular_contours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Contour]

Returns a strategy for triangular contours. Triangular contour is a contour formed by 3 points.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> contours = planar.triangular_contours(coordinates)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> len(contour.vertices) == 3
True
>>> all(isinstance(vertex.x, coordinates_type)
```

(continues on next page)

(continued from previous page)

```
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> contours = planar.triangular_contours(x_coordinates, y_coordinates)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> len(contour.vertices) == 3
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**rectangular_contours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Contour]

Returns a strategy for axis-aligned rectangular contours. Rectangular contour is a contour formed by 4 points.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> contours = planar.rectangular_contours(coordinates)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> len(contour.vertices) == 4
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> contours = planar.rectangular_contours(x_coordinates, y_coordinates)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> len(contour.vertices) == 4
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**boxes**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Box]

Returns a strategy for boxes.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Box = context.box_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> boxes = planar.boxes(coordinates)
>>> box = boxes.example()
>>> isinstance(box, Box)
True
>>> (isinstance(box.min_x, coordinates_type)
...  and isinstance(box.max_x, coordinates_type)
...  and isinstance(box.min_y, coordinates_type)
...  and isinstance(box.max_y, coordinates_type))
True
>>> (min_coordinate <= box.min_x <= max_coordinate
...  and min_coordinate <= box.max_x <= max_coordinate
...  and min_coordinate <= box.min_y <= max_coordinate
...  and min_coordinate <= box.max_y <= max_coordinate)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> boxes = planar.boxes(x_coordinates, y_coordinates)
>>> box = boxes.example()
>>> isinstance(box, Box)
True
>>> (isinstance(box.min_x, coordinates_type)
...  and isinstance(box.max_x, coordinates_type)
...  and isinstance(box.min_y, coordinates_type)
...  and isinstance(box.max_y, coordinates_type))
True
>>> (min_x_coordinate <= box.min_x <= max_x_coordinate
...  and min_x_coordinate <= box.max_x <= max_x_coordinate)
True
>>> (min_y_coordinate <= box.min_y <= max_y_coordinate
...  and min_y_coordinate <= box.max_y <= max_y_coordinate)
True
```

hypothesis_geometry.planar.**star_contours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate], y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None, *, min_size: int = <MinContourSize.CONVEX: 3>, max_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Contour]

Returns a strategy for star contours. Star contour is a contour such that every vertex is visible from centroid, i.e. segments from centroid to vertices do not cross or overlap contour.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for contour size.
- **max_size** – upper bound for contour size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.star_contours(coordinates,
...                                 min_size=min_size,
...                                 max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
```

(continues on next page)

(continued from previous page)

```
...                                     allow_infinity=False,
...                                     allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                     allow_infinity=False,
...                                     allow_nan=False)
>>> min_size, max_size = 5, 10
>>> contours = planar.star_contours(x_coordinates, y_coordinates,
...                                 min_size=min_size,
...                                 max_size=max_size)
>>> contour = contours.example()
>>> isinstance(contour, Contour)
True
>>> min_size <= len(contour.vertices) <= max_size
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**multicontours**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate]*, *y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None*, ***, *min_size: int = 0*, *max_size: Optional[int] = None*, *min_contour_size: int = <MinContourSize.CONVEX: 3>*, *max_contour_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[Sequence[ground.hints.Contour]]

Returns a strategy for multicontours. Multicontour is a possibly empty sequence of non-crossing and non-overlapping contours.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for size.
- **max_size** – upper bound for size, `None` for unbound.
- **min_contour_size** – lower bound for contour size.
- **max_contour_size** – upper bound for contour size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Contour = context.contour_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
>>> min_contour_size, max_contour_size = 3, 5
>>> multicontours = planar.multicontours(coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size,
...                                      min_contour_size=min_contour_size,
...                                      max_contour_size=max_contour_size)
>>> multicontour = multicontours.example()
>>> isinstance(multicontour, list)
True
>>> all(isinstance(contour, Contour) for contour in multicontour)
True
>>> min_size <= len(multicontour) <= max_size
True
>>> all(min_contour_size <= len(contour.vertices) <= max_contour_size
...     for contour in multicontour)
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for contour in multicontour
...     for vertex in contour.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for contour in multicontour
...     for vertex in contour.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 5, 10
>>> min_contour_size, max_contour_size = 3, 5
>>> multicontours = planar.multicontours(x_coordinates, y_coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size,
...                                      min_contour_size=min_contour_size,
...                                      max_contour_size=max_contour_size)
>>> multicontour = multicontours.example()
>>> isinstance(multicontour, list)
True
>>> all(isinstance(contour, Contour) for contour in multicontour)
True
>>> min_size <= len(multicontour) <= max_size
```

(continues on next page)

(continued from previous page)

```
True
>>> all(min_contour_size <= len(contour.vertices) <= max_contour_size
...     for contour in multicontour)
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for contour in multicontour
...     for vertex in contour.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for contour in multicontour
...     for vertex in contour.vertices)
True
```

hypothesis_geometry.planar.**polygons**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate], y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None, *, min_size: int = <MinContourSize.CONVEX: 3>, max_size: Optional[int] = None, min_holes_size: int = 0, max_holes_size: Optional[int] = None, min_hole_size: int = <MinContourSize.CONVEX: 3>, max_hole_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Polygon]

Returns a strategy for polygons.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for border size.
- **max_size** – upper bound for border size, `None` for unbound.
- **min_holes_size** – lower bound for holes count.
- **max_holes_size** – upper bound for holes count, `None` for countless.
- **min_hole_size** – lower bound for hole size.
- **max_hole_size** – upper bound for hole size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Polygon = context.polygon_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 5, 10
```

(continues on next page)

(continued from previous page)

```
>>> min_holes_size, max_holes_size = 1, 4
>>> min_hole_size, max_hole_size = 3, 5
>>> polygons = planar.polygons(coordinates,
...                            min_size=min_size,
...                            max_size=max_size,
...                            min_holes_size=min_holes_size,
...                            max_holes_size=max_holes_size,
...                            min_hole_size=min_hole_size,
...                            max_hole_size=max_hole_size)
>>> polygon = polygons.example()
>>> isinstance(polygon, Polygon)
True
>>> min_size <= len(polygon.border.vertices) <= max_size
True
>>> min_holes_size <= len(polygon.holes) <= max_holes_size
True
>>> all(min_hole_size <= len(hole.vertices) <= max_hole_size
...     for hole in polygon.holes)
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in polygon.border.vertices)
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for hole in polygon.holes
...     for vertex in hole.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for vertex in polygon.border.vertices)
True
>>> all(min_coordinate <= vertex.x <= max_coordinate
...     and min_coordinate <= vertex.y <= max_coordinate
...     for hole in polygon.holes
...     for vertex in hole.vertices)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 5, 10
>>> min_holes_size, max_holes_size = 1, 4
>>> min_hole_size, max_hole_size = 3, 5
>>> polygons = planar.polygons(x_coordinates, y_coordinates,
...                            min_size=min_size,
...                            max_size=max_size,
...                            min_holes_size=min_holes_size,
...                            max_holes_size=max_holes_size,
```

(continues on next page)

(continued from previous page)

```
...                             min_hole_size=min_hole_size,
...                             max_hole_size=max_hole_size)
>>> polygon = polygons.example()
>>> isinstance(polygon, Polygon)
True
>>> min_size <= len(polygon.border.vertices) <= max_size
True
>>> min_holes_size <= len(polygon.holes) <= max_holes_size
True
>>> all(min_hole_size <= len(hole.vertices) <= max_hole_size
...     for hole in polygon.holes)
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for vertex in polygon.border.vertices)
True
>>> all(isinstance(vertex.x, coordinates_type)
...     and isinstance(vertex.y, coordinates_type)
...     for hole in polygon.holes
...     for vertex in hole.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for vertex in polygon.border.vertices)
True
>>> all(min_x_coordinate <= vertex.x <= max_x_coordinate
...     and min_y_coordinate <= vertex.y <= max_y_coordinate
...     for hole in polygon.holes
...     for vertex in hole.vertices)
True
```

hypothesis_geometry.planar.**multipolygons**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate], y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None, *, min_size: int = 0, max_size: Optional[int] = None, min_border_size: int = <MinContourSize.CONVEX: 3>, max_border_size: Optional[int] = None, min_holes_size: int = 0, max_holes_size: Optional[int] = None, min_hole_size: int = <MinContourSize.CONVEX: 3>, max_hole_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[ground.hints.Multipolygon]

Returns a strategy for multipolygons.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_size** – lower bound for size.
- **max_size** – upper bound for size, `None` for unbound.
- **min_border_size** – lower bound for polygons' border size.

- **max_border_size** – upper bound for polygons' border size, `None` for unbound.
- **min_holes_size** – lower bound for polygons' holes count.
- **max_holes_size** – upper bound for polygons' holes count, `None` for countless.
- **min_hole_size** – lower bound for hole size.
- **max_hole_size** – upper bound for polygons' hole size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
>>> Multipolygon = context.multipolygon_cls
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_size, max_size = 0, 5
>>> min_border_size, max_border_size = 5, 10
>>> min_holes_size, max_holes_size = 1, 4
>>> min_hole_size, max_hole_size = 3, 5
>>> multipolygons = planar.multipolygons(coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size,
...                                      min_border_size=min_border_size,
...                                      max_border_size=max_border_size,
...                                      min_holes_size=min_holes_size,
...                                      max_holes_size=max_holes_size,
...                                      min_hole_size=min_hole_size,
...                                      max_hole_size=max_hole_size)
>>> multipolygon = multipolygons.example()
>>> isinstance(multipolygon, Multipolygon)
True
>>> min_size <= len(multipolygon.polygons) <= max_size
True
>>> all(min_border_size <= len(polygon.border.vertices) <= max_border_size
...     and min_holes_size <= len(polygon.holes) <= max_holes_size
...     and all(min_hole_size <= len(hole.vertices) <= max_hole_size
...             for hole in polygon.holes)
...     for polygon in multipolygon.polygons)
True
>>> all(all(isinstance(vertex.x, coordinates_type)
...         and isinstance(vertex.y, coordinates_type)
...         for vertex in polygon.border.vertices)
...     and all(isinstance(vertex.x, coordinates_type)
...             and isinstance(vertex.y, coordinates_type)
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
>>> all(all(min_coordinate <= vertex.x <= max_coordinate
...         and min_coordinate <= vertex.y <= max_coordinate
...         for vertex in polygon.border.vertices)
```

(continues on next page)

(continued from previous page)

```
...     and all(min_coordinate <= vertex.x <= max_coordinate
...             and min_coordinate <= vertex.y <= max_coordinate
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_size, max_size = 0, 5
>>> min_border_size, max_border_size = 5, 10
>>> min_holes_size, max_holes_size = 1, 4
>>> min_hole_size, max_hole_size = 3, 5
>>> multipolygons = planar.multipolygons(x_coordinates, y_coordinates,
...                                      min_size=min_size,
...                                      max_size=max_size,
...                                      min_border_size=min_border_size,
...                                      max_border_size=max_border_size,
...                                      min_holes_size=min_holes_size,
...                                      max_holes_size=max_holes_size,
...                                      min_hole_size=min_hole_size,
...                                      max_hole_size=max_hole_size)
>>> multipolygon = multipolygons.example()
>>> isinstance(multipolygon, Multipolygon)
True
>>> min_size <= len(multipolygon.polygons) <= max_size
True
>>> all(min_border_size <= len(polygon.border.vertices) <= max_border_size
...     and min_holes_size <= len(polygon.holes) <= max_holes_size
...     and all(min_hole_size <= len(hole.vertices) <= max_hole_size
...             for hole in polygon.holes)
...     for polygon in multipolygon.polygons)
True
>>> all(all(isinstance(vertex.x, coordinates_type)
...         and isinstance(vertex.y, coordinates_type)
...         for vertex in polygon.border.vertices)
...     and all(isinstance(vertex.x, coordinates_type)
...             and isinstance(vertex.y, coordinates_type)
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
>>> all(all(min_x_coordinate <= vertex.x <= max_x_coordinate
...         and min_y_coordinate <= vertex.y <= max_y_coordinate
...         for vertex in polygon.border.vertices)
...     and all(min_x_coordinate <= vertex.x <= max_x_coordinate
...             and min_y_coordinate <= vertex.y <= max_y_coordinate
...             for hole in polygon.holes
```

(continues on next page)

(continued from previous page)

```
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
```

hypothesis_geometry.planar.**mixes**(*x_coordinates: hypothesis.strategies.SearchStrategy[Coordinate], y_coordinates: Optional[hypothesis.strategies.SearchStrategy[Coordinate]] = None, *, min_multipoint_size: int = 0, max_multipoint_size: Optional[int] = None, min_multisegment_size: int = 0, max_multisegment_size: Optional[int] = None, min_multipolygon_size: int = 0, max_multipolygon_size: Optional[int] = None, min_multipolygon_border_size: int = <MinContourSize.CONVEX: 3>, max_multipolygon_border_size: Optional[int] = None, min_multipolygon_holes_size: int = 0, max_multipolygon_holes_size: Optional[int] = None, min_multipolygon_hole_size: int = <MinContourSize.CONVEX: 3>, max_multipolygon_hole_size: Optional[int] = None*) → hypothesis.strategies.SearchStrategy[Tuple[ground.hints.Multipoint, ground.hints.Multisegment, ground.hints.Multipolygon]]

Returns a strategy for mixes. Mix is a triplet of disjoint multipoint, multisegment and multipolygon.

Parameters

- **x_coordinates** – strategy for vertices' x-coordinates.
- **y_coordinates** – strategy for vertices' y-coordinates, `None` for reusing x-coordinates strategy.
- **min_multipoint_size** – lower bound for multipoint size.
- **max_multipoint_size** – upper bound for multipoint size, `None` for unbound.
- **min_multisegment_size** – lower bound for multisegment size.
- **max_multisegment_size** – upper bound for multisegment size, `None` for unbound.
- **min_multipolygon_size** – lower bound for multipolygon size.
- **max_multipolygon_size** – upper bound for multipolygon size, `None` for unbound.
- **min_multipolygon_border_size** – lower bound for polygons' border size.
- **max_multipolygon_border_size** – upper bound for polygons' border size, `None` for unbound.
- **min_multipolygon_holes_size** – lower bound for polygons' holes count.
- **max_multipolygon_holes_size** – upper bound for polygons' holes count, `None` for countless.
- **min_multipolygon_hole_size** – lower bound for hole size.
- **max_multipolygon_hole_size** – upper bound for polygons' hole size, `None` for unbound.

```
>>> from ground.base import get_context
>>> from hypothesis import strategies
>>> from hypothesis_geometry import planar
>>> context = get_context()
```

(continues on next page)

(continued from previous page)

```
>>> Multipoint, Multipolygon, Multisegment = (context.multipoint_cls,
...                                           context.multipolygon_cls,
...                                           context.multisegment_cls)
```

For same coordinates' domain:

```
>>> min_coordinate, max_coordinate = -1., 1.
>>> coordinates_type = float
>>> coordinates = strategies.floats(min_coordinate, max_coordinate,
...                                 allow_infinity=False,
...                                 allow_nan=False)
>>> min_multipoint_size, max_multipoint_size = 2, 3
>>> min_multisegment_size, max_multisegment_size = 1, 4
>>> min_multipolygon_size, max_multipolygon_size = 0, 5
>>> min_multipolygon_border_size, max_multipolygon_border_size = 5, 10
>>> min_multipolygon_holes_size, max_multipolygon_holes_size = 1, 4
>>> min_multipolygon_hole_size, max_multipolygon_hole_size = 3, 5
>>> mixes = planar.mixes(
...         coordinates,
...         min_multipoint_size=min_multipoint_size,
...         max_multipoint_size=max_multipoint_size,
...         min_multisegment_size=min_multisegment_size,
...         max_multisegment_size=max_multisegment_size,
...         min_multipolygon_size=min_multipolygon_size,
...         max_multipolygon_size=max_multipolygon_size,
...         min_multipolygon_border_size=min_multipolygon_border_size,
...         max_multipolygon_border_size=max_multipolygon_border_size,
...         min_multipolygon_holes_size=min_multipolygon_holes_size,
...         max_multipolygon_holes_size=max_multipolygon_holes_size,
...         min_multipolygon_hole_size=min_multipolygon_hole_size,
...         max_multipolygon_hole_size=max_multipolygon_hole_size)
>>> mix = mixes.example()
>>> isinstance(mix, tuple)
True
>>> len(mix) == 3
True
>>> multipoint, multisegment, multipolygon = mix
>>> isinstance(multipoint, Multipoint)
True
>>> min_multipoint_size <= len(multipoint.points) <= max_multipoint_size
True
>>> all(isinstance(point.x, coordinates_type)
...     and isinstance(point.y, coordinates_type)
...     for point in multipoint.points)
True
>>> all(min_coordinate <= point.x <= max_coordinate
...     and min_coordinate <= point.y <= max_coordinate
...     for point in multipoint.points)
True
>>> isinstance(multisegment, Multisegment)
True
>>> (min_multisegment_size <= len(multisegment.segments)
...  <= max_multisegment_size)
True
>>> all(isinstance(segment.start.x, coordinates_type)
...     and isinstance(segment.start.y, coordinates_type)
...     and isinstance(segment.end.x, coordinates_type)
```

(continues on next page)

(continued from previous page)

```
...     and isinstance(segment.end.y, coordinates_type)
...     for segment in multisegment.segments)
True
>>> all(min_coordinate <= segment.start.x <= max_coordinate
...     and min_coordinate <= segment.start.y <= max_coordinate
...     and min_coordinate <= segment.end.x <= max_coordinate
...     and min_coordinate <= segment.end.y <= max_coordinate
...     for segment in multisegment.segments)
True
>>> isinstance(multipolygon, Multipolygon)
True
>>> (min_multipolygon_size <= len(multipolygon.polygons)
...  <= max_multipolygon_size)
True
>>> all(min_multipolygon_border_size
...     <= len(polygon.border.vertices)
...     <= max_multipolygon_border_size
...     and (min_multipolygon_holes_size
...          <= len(polygon.holes)
...          <= max_multipolygon_holes_size)
...     and all(min_multipolygon_hole_size
...             <= len(hole.vertices)
...             <= max_multipolygon_hole_size
...             for hole in polygon.holes)
...     for polygon in multipolygon.polygons)
True
>>> all(all(isinstance(vertex.x, coordinates_type)
...         and isinstance(vertex.y, coordinates_type)
...         for vertex in polygon.border.vertices)
...     and all(isinstance(vertex.x, coordinates_type)
...             and isinstance(vertex.y, coordinates_type)
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
>>> all(all(min_coordinate <= vertex.x <= max_coordinate
...         and min_coordinate <= vertex.y <= max_coordinate
...         for vertex in polygon.border.vertices)
...     and all(min_coordinate <= vertex.x <= max_coordinate
...             and min_coordinate <= vertex.y <= max_coordinate
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
```

For different coordinates' domains:

```
>>> min_x_coordinate, max_x_coordinate = -1., 1.
>>> min_y_coordinate, max_y_coordinate = 10., 100.
>>> coordinates_type = float
>>> x_coordinates = strategies.floats(min_x_coordinate, max_x_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> y_coordinates = strategies.floats(min_y_coordinate, max_y_coordinate,
...                                   allow_infinity=False,
...                                   allow_nan=False)
>>> min_multipoint_size, max_multipoint_size = 2, 3
```

(continues on next page)

(continued from previous page)

```
>>> min_multisegment_size, max_multisegment_size = 1, 4
>>> min_multipolygon_size, max_multipolygon_size = 0, 5
>>> min_multipolygon_border_size, max_multipolygon_border_size = 5, 10
>>> min_multipolygon_holes_size, max_multipolygon_holes_size = 1, 4
>>> min_multipolygon_hole_size, max_multipolygon_hole_size = 3, 5
>>> mixes = planar.mixes(
...         x_coordinates, y_coordinates,
...         min_multipoint_size=min_multipoint_size,
...         max_multipoint_size=max_multipoint_size,
...         min_multisegment_size=min_multisegment_size,
...         max_multisegment_size=max_multisegment_size,
...         min_multipolygon_size=min_multipolygon_size,
...         max_multipolygon_size=max_multipolygon_size,
...         min_multipolygon_border_size=min_multipolygon_border_size,
...         max_multipolygon_border_size=max_multipolygon_border_size,
...         min_multipolygon_holes_size=min_multipolygon_holes_size,
...         max_multipolygon_holes_size=max_multipolygon_holes_size,
...         min_multipolygon_hole_size=min_multipolygon_hole_size,
...         max_multipolygon_hole_size=max_multipolygon_hole_size)
>>> mix = mixes.example()
>>> isinstance(mix, tuple)
True
>>> len(mix) == 3
True
>>> multipoint, multisegment, multipolygon = mix
>>> isinstance(multipoint, Multipoint)
True
>>> min_multipoint_size <= len(multipoint.points) <= max_multipoint_size
True
>>> all(isinstance(point.x, coordinates_type)
...     and isinstance(point.y, coordinates_type)
...     for point in multipoint.points)
True
>>> all(min_x_coordinate <= point.x <= max_x_coordinate
...     and min_y_coordinate <= point.y <= max_y_coordinate
...     for point in multipoint.points)
True
>>> isinstance(multisegment, Multisegment)
True
>>> (min_multisegment_size <= len(multisegment.segments)
...  <= max_multisegment_size)
True
>>> all(isinstance(segment.start.x, coordinates_type)
...     and isinstance(segment.start.y, coordinates_type)
...     and isinstance(segment.end.x, coordinates_type)
...     and isinstance(segment.end.y, coordinates_type)
...     for segment in multisegment.segments)
True
>>> all(min_x_coordinate <= segment.start.x <= max_x_coordinate
...     and min_y_coordinate <= segment.start.y <= max_y_coordinate
...     and min_x_coordinate <= segment.end.x <= max_x_coordinate
...     and min_y_coordinate <= segment.end.y <= max_y_coordinate
...     for segment in multisegment.segments)
True
>>> isinstance(multipolygon, Multipolygon)
True
>>> (min_multipolygon_size <= len(multipolygon.polygons)
```

(continues on next page)

(continued from previous page)

```
...  <= max_multipolygon_size)
True
>>> all(min_multipolygon_border_size
...     <= len(polygon.border.vertices)
...     <= max_multipolygon_border_size
...     and (min_multipolygon_holes_size
...          <= len(polygon.holes)
...          <= max_multipolygon_holes_size)
...     and all(min_multipolygon_hole_size
...             <= len(hole.vertices)
...             <= max_multipolygon_hole_size
...             for hole in polygon.holes)
...     for polygon in multipolygon.polygons)
True
>>> all(all(isinstance(vertex.x, coordinates_type)
...         and isinstance(vertex.y, coordinates_type)
...         for vertex in polygon.border.vertices)
...     and all(isinstance(vertex.x, coordinates_type)
...             and isinstance(vertex.y, coordinates_type)
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
>>> all(all(min_x_coordinate <= vertex.x <= max_x_coordinate
...         and min_y_coordinate <= vertex.y <= max_y_coordinate
...         for vertex in polygon.border.vertices)
...     and all(min_x_coordinate <= vertex.x <= max_x_coordinate
...             and min_y_coordinate <= vertex.y <= max_y_coordinate
...             for hole in polygon.holes
...             for vertex in hole.vertices)
...     for polygon in multipolygon.polygons)
True
```

PYTHON MODULE INDEX

h

hypothesis_geometry.planar, 1

INDEX

B

boxes() (*in module hypothesis_geometry.planar*), 12

C

concave_contours() (*in module hypothesis_geometry.planar*), 8
contours() (*in module hypothesis_geometry.planar*), 6
convex_contours() (*in module hypothesis_geometry.planar*), 7

H

hypothesis_geometry.planar
    module, 1

M

mixes() (*in module hypothesis_geometry.planar*), 22
module
    hypothesis_geometry.planar, 1
multicontours() (*in module hypothesis_geometry.planar*), 15
multipoints() (*in module hypothesis_geometry.planar*), 2
multipolygons() (*in module hypothesis_geometry.planar*), 19
multisegments() (*in module hypothesis_geometry.planar*), 4

P

points() (*in module hypothesis_geometry.planar*), 1
polygons() (*in module hypothesis_geometry.planar*), 17

R

rectangular_contours() (*in module hypothesis_geometry.planar*), 11

S

segments() (*in module hypothesis_geometry.planar*), 3
star_contours() (*in module hypothesis_geometry.planar*), 13

T

triangular_contours() (*in module hypothesis_geometry.planar*), 10