Help API : Part.Face

>>> help(Part.Face.Area)

Help on getset descriptor Part.Shape.Area:

Area
    Total area of the faces of the shape.

>>> help(Part.Face.BoundBox)

Help on getset descriptor Data.ComplexGeoData.BoundBox:

BoundBox
    Get the BoundBox of the object

>>> help(Part.Face.CenterOfGravity)

Help on getset descriptor Data.ComplexGeoData.CenterOfGravity:

CenterOfGravity
    Get the center of gravity

>>> help(Part.Face.CenterOfMass)

Help on getset descriptor Part.Face.CenterOfMass:

CenterOfMass
    Returns the center of mass of the current system.
    If the gravitational field is uniform, it is the center of gravity.
    The coordinates returned for the center of mass are expressed in the
    absolute Cartesian coordinate system.

>>> help(Part.Face.CompSolids)

Help on getset descriptor Part.Shape.CompSolids:

CompSolids
    List of subsequent shapes in this shape.

>>> help(Part.Face.Compounds)

Help on getset descriptor Part.Shape.Compounds:

Compounds
    List of compounds in this shape.

>>> help(Part.Face.Content)

Help on getset descriptor Base.Persistence.Content:

Content
    Content of the object in XML representation

>>> help(Part.Face.Edges)

Help on getset descriptor Part.Shape.Edges:

Edges
    List of Edges in this shape.

>>> help(Part.Face.Faces)

Help on getset descriptor Part.Shape.Faces:

Faces
    List of faces in this shape.

>>> help(Part.Face.Length)

Help on getset descriptor Part.Shape.Length:

Length
    Total length of the edges of the shape.

>>> help(Part.Face.Mass)

Help on getset descriptor Part.Face.Mass:

Mass
    Returns the mass of the current system.

>>> help(Part.Face.MatrixOfInertia)

Help on getset descriptor Part.Face.MatrixOfInertia:

MatrixOfInertia
    Returns the matrix of inertia. It is a symmetrical matrix.
    The coefficients of the matrix are the quadratic moments of
    inertia.

     | Ixx Ixy Ixz 0 |
     | Ixy Iyy Iyz 0 |
     | Ixz Iyz Izz 0 |
     | 0   0   0   1 |

    The moments of inertia are denoted by Ixx, Iyy, Izz.
    The products of inertia are denoted by Ixy, Ixz, Iyz.
    The matrix of inertia is returned in the central coordinate
    system (G, Gx, Gy, Gz) where G is the centre of mass of the
    system and Gx, Gy, Gz the directions parallel to the X(1,0,0)
    Y(0,1,0) Z(0,0,1) directions of the absolute cartesian
    coordinate system.

>>> help(Part.Face.MemSize)

Help on getset descriptor Base.Persistence.MemSize:

MemSize
    Memory size of the object in byte

>>> help(Part.Face.Module)

Help on getset descriptor Base.BaseClass.Module:

Module
    Module in which this class is defined

>>> help(Part.Face.Orientation)

Help on getset descriptor Part.Shape.Orientation:

Orientation
    Returns the orientation of the shape.

>>> help(Part.Face.OuterWire)

Help on getset descriptor Part.Face.OuterWire:

OuterWire
    The outer wire of this face

>>> help(Part.Face.ParameterRange)

Help on getset descriptor Part.Face.ParameterRange:

ParameterRange
Returns a 4 tuple with the parameter range

>>> help(Part.Face.Placement)

Help on getset descriptor Data.ComplexGeoData.Placement:

Placement
Get the current transformation of the object as placement

>>> help(Part.Face.PrincipalProperties)

Help on getset descriptor Part.Face.PrincipalProperties:

PrincipalProperties
Computes the principal properties of inertia of the current system.
There is always a set of axes for which the products
of inertia of a geometric system are equal to 0; i.e. the
matrix of inertia of the system is diagonal. These axes
are the principal axes of inertia. Their origin is
coincident with the center of mass of the system. The
associated moments are called the principal moments of inertia.
This function computes the eigen values and the
eigen vectors of the matrix of inertia of the system.

>>> help(Part.Face.ShapeType)

Help on getset descriptor Part.Shape.ShapeType:

ShapeType
Returns the type of the shape.

>>> help(Part.Face.Shells)

Help on getset descriptor Part.Shape.Shells:

Shells
List of subsequent shapes in this shape.

>>> help(Part.Face.Solids)

Help on getset descriptor Part.Shape.Solids:

Solids
List of subsequent shapes in this shape.

>>> help(Part.Face.StaticMoments)

Help on getset descriptor Part.Face.StaticMoments:

StaticMoments
Returns Ix, Iy, Iz, the static moments of inertia of the
current system; i.e. the moments of inertia about the
three axes of the Cartesian coordinate system.

>>> help(Part.Face.SubShapes)

Help on getset descriptor Part.Shape.SubShapes:

SubShapes
List of sub-shapes in this shape.

>>> help(Part.Face.Surface)

Help on getset descriptor Part.Face.Surface:

Surface
Returns the geometric surface of the face

>>> help(Part.Face.Tag)

Help on getset descriptor Data.ComplexGeoData.Tag:

Tag
    Geometry Tag

>>> help(Part.Face.Tolerance)

Help on getset descriptor Part.Face.Tolerance:

Tolerance
    Set or get the tolerance of the vertex

>>> help(Part.Face.TypeId)

Help on getset descriptor Base.BaseClass.TypeId:

TypeId
    Is the type of the FreeCAD object with module domain

>>> help(Part.Face.Vertexes)

Help on getset descriptor Part.Shape.Vertexes:

Vertexes
    List of vertexes in this shape.

>>> help(Part.Face.Volume)

Help on getset descriptor Part.Shape.Volume:

Volume
    Total volume of the solids of the shape.

>>> help(Part.Face.Wire)

Help on getset descriptor Part.Face.Wire:

Wire
    The outer wire of this face
    deprecated -- please use OuterWire

>>> help(Part.Face.Wires)

Help on getset descriptor Part.Shape.Wires:

Wires
    List of wires in this shape.

>>> help(Part.Face.__class__)

Help on class type in module builtins:

class type(object)
 | type(object_or_name, bases, dict)
 | type(object) -> the object's type
 | type(name, bases, dict) -> a new type
 |
 | Methods defined here:
 |
 | __call__(self, /, *args, **kwargs)
 | Call self as a function.
 |
 | __delattr__(self, name, /)
 | Implement delattr(self, name).
 |

```
|  __dir__(self, /)
|      Specialized __dir__ implementation for types.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __instancecheck__(self, instance, /)
|      Check if an object is an instance.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setattr__(self, name, value, /)
|      Implement setattr(self, name, value).
|
|  __sizeof__(self, /)
|      Return memory consumption of the type object.
|
|  __subclasscheck__(self, subclass, /)
|      Check if a class is a subclass.
|
|  __subclasses__(self, /)
|      Return a list of immediate subclasses.
|
|  mro(self, /)
|      Return a type's method resolution order.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __prepare__(...)
|      __prepare__() -> dict
|      used to create the namespace for the class statement
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs)
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __abstractmethods__
|
|  __dict__
|
|  __signature__
|
|  __text_signature__
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __base__ = <class 'object'>
|      The base class of the class hierarchy.
|
|      When called, it accepts no arguments and returns a new featureless
|      instance that has no instance attributes and cannot be given any.
|
```

| __bases__ = (<class 'object'>,)
|
| __basicsize__ = 880
|
| __dictoffset__ = 264
|
| __flags__ = 2148291584
|
| __itemsize__ = 40
|
| __mro__ = (<class 'type'>, <class 'object'>)
|
| __weakrefoffset__ = 368

>>> help(Part.Face.__delattr__)

Help on wrapper_descriptor:

__delattr__(self, name, /)
Implement delattr(self, name).

>>> help(Part.Face.__dir__)

Help on method_descriptor:

__dir__(self, /)
Default dir() implementation.

>>> help(Part.Face.__doc__)

No Python documentation found for 'TopoShapeFace is the OpenCasCade topological face wrapper'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.

>>> help(Part.Face.__eq__)

Help on wrapper_descriptor:

__eq__(self, value, /)
Return self==value.

>>> help(Part.Face.__format__)

Help on method_descriptor:

__format__(self, format_spec, /)
Default object formatter.

>>> help(Part.Face.__ge__)

Help on wrapper_descriptor:

__ge__(self, value, /)
Return self>=value.

>>> help(Part.Face.__getattribute__)

Help on wrapper_descriptor:

__getattribute__(self, name, /)
Return getattr(self, name).

>>> help(Part.Face.__getstate__)

Help on method_descriptor:

__getstate__(...)
Serialize the content of this shape to a string in BREP format.

>>> help(Part.Face.__gt__)

Help on wrapper_descriptor:

__gt__(self, value, /)
    Return self>value.

>>> help(Part.Face.__hash__)

Help on wrapper_descriptor:

__hash__(self, /)
    Return hash(self).

>>> help(Part.Face.__init__)

Help on wrapper_descriptor:

__init__(self, /, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

>>> help(Part.Face.__init_subclass__)

Help on built-in function __init_subclass__:

__init_subclass__(...) method of builtins.type instance
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be
    overridden to extend subclasses.

>>> help(Part.Face.__le__)

Help on wrapper_descriptor:

__le__(self, value, /)
    Return self<=value.

>>> help(Part.Face.__lt__)

Help on wrapper_descriptor:

__lt__(self, value, /)
    Return self<value.

>>> help(Part.Face.__ne__)

Help on wrapper_descriptor:

__ne__(self, value, /)
    Return self!=value.

>>> help(Part.Face.__new__)

Help on built-in function __new__:

__new__(*args, **kwargs) method of builtins.type instance
    Create and return a new object. See help(type) for accurate signature.

>>> help(Part.Face.__reduce__)

Help on method_descriptor:

__reduce__(self, /)
    Helper for pickle.

>>> help(Part.Face.__reduce_ex__)

Help on method_descriptor:

__reduce_ex__(self, protocol, /)
    Helper for pickle.

>>> help(Part.Face.__repr__)

Help on wrapper_descriptor:

__repr__(self, /)
    Return repr(self).

>>> help(Part.Face.__setattr__)

Help on wrapper_descriptor:

__setattr__(self, name, value, /)
    Implement setattr(self, name, value).

>>> help(Part.Face.__setstate__)

Help on method_descriptor:

__setstate__(...)
    Deserialize the content of this shape from a string in BREP format.

>>> help(Part.Face.__sizeof__)

Help on method_descriptor:

__sizeof__(self, /)
    Size of object in memory, in bytes.

>>> help(Part.Face.__str__)

Help on wrapper_descriptor:

__str__(self, /)
    Return str(self).

>>> help(Part.Face.__subclasshook__)

Help on built-in function __subclasshook__:

__subclasshook__(...) method of builtins.type instance
    Abstract classes can override this to customize issubclass().

    This is invoked early on by abc.ABCMeta.__subclasscheck__().
    It should return True, False or NotImplemented. If it returns
    NotImplemented, the normal algorithm is used. Otherwise, it
    overrides the normal algorithm (and the outcome is cached).

>>> help(Part.Face.addWire)

Help on method_descriptor:

addWire(...)
    Adds a wire to the face.
    addWire(wire)

>>> help(Part.Face.ancestorsOfType)

Help on method_descriptor:

ancestorsOfType(...)
    For a sub-shape of this shape get its ancestors of a type.
    ancestorsOfType(shape, shape type) -> list

>>> help(Part.Face.applyRotation)

Help on method_descriptor:

applyRotation(...)
Apply an additional rotation to the placement

>>> help(Part.Face.applyTranslation)

Help on method_descriptor:

applyTranslation(...)
Apply an additional translation to the placement

>>> help(Part.Face.check)

Help on method_descriptor:

check(...)
Checks the shape and report errors in the shape structure.
check([runBopCheck = False])
--
This is a more detailed check as done in isValid().
if runBopCheck is True, a BOPCheck analysis is also performed.

>>> help(Part.Face.childShapes)

Help on method_descriptor:

childShapes(...)
Return a list of sub-shapes that are direct children of this shape.
childShapes([cumOri=True, cumLoc=True]) -> list
--
* If cumOri is true, the function composes all
sub-shapes with the orientation of this shape.
* If cumLoc is true, the function multiplies all
sub-shapes by the location of this shape, i.e. it applies to
each sub-shape the transformation that is associated with this shape.

>>> help(Part.Face.cleaned)

Help on method_descriptor:

cleaned(...)
This creates a cleaned copy of the shape with the triangulation removed.
clean()
--
This can be useful to reduce file size when exporting as a BREP file.
Warning: Use the cleaned shape with care because certain algorithms may work incorrectly
if the shape has no internal triangulation any more.

>>> help(Part.Face.common)

Help on method_descriptor:

common(...)
Intersection of this and a given (list of) topo shape.
common(tool) -> Shape
or
common((tool1,tool2,...),[tolerance=0.0]) -> Shape
--
Supports:
- Fuzzy Boolean operations (global tolerance for a Boolean operation)
- Support of multiple arguments for a single Boolean operation (s1 AND (s2 OR s3))
- Parallelization of Boolean Operations algorithm

OCC 6.9.0 or later is required.

>>> help(Part.Face.complement)

Help on method_descriptor:

complement(...)
Computes the complement of the orientation of this shape,
i.e. reverses the interior/exterior status of boundaries of this shape.
complement()

>>> help(Part.Face.copy)

Help on method_descriptor:

copy(...)
Create a copy of this shape
copy(copyGeom=True, copyMesh=False) -> Shape
--
If copyMesh is True, triangulation contained in original shape will be
copied along with geometry.
If copyGeom is False, only topological objects will be copied, while
geometry and triangulation will be shared with original shape.

>>> help(Part.Face.countElement)

Help on method_descriptor:

countElement(...)
Returns the count of a type of element
countElement(type) -> int

>>> help(Part.Face.countSubElements)

Help on method_descriptor:

countSubElements(...)
Return the number of elements of a type

>>> help(Part.Face.curvatureAt)

Help on method_descriptor:

curvatureAt(...)
Get the curvature at the given parameter [0|Length] if defined
curvatureAt(u,v) -> Float

>>> help(Part.Face.curveOnSurface)

Help on method_descriptor:

curveOnSurface(...)
Returns the curve associated to the edge in the parametric space of the face.
curveOnSurface(Edge) -> (curve, min, max) or None
--
If this curve exists then a tuple of curve and parameter range is returned.
Returns None if this curve does not exist.

>>> help(Part.Face.cut)

Help on method_descriptor:

cut(...)
Difference of this and a given (list of) topo shape
cut(tool) -> Shape
or
cut((tool1,tool2,...),[tolerance=0.0]) -> Shape
--

Supports:
- Fuzzy Boolean operations (global tolerance for a Boolean operation)
- Support of multiple arguments for a single Boolean operation
- Parallelization of Boolean Operations algorithm

OCC 6.9.0 or later is required.

>>> help(Part.Face.cutHoles)

Help on method_descriptor:

cutHoles(...)
Cut holes in the face.
cutHoles(list_of_wires)

>>> help(Part.Face.defeaturing)

Help on method_descriptor:

defeaturing(...)
Remove a feature defined by supplied faces and return a new shape.
defeaturing(shapeList) -> Shape
--
The parameter is a list of faces.

>>> help(Part.Face.derivative1At)

Help on method_descriptor:

derivative1At(...)
Get the first derivative at the given parameter [0|Length] if defined
derivative1At(u,v) -> (vectorU,vectorV)

>>> help(Part.Face.derivative2At)

Help on method_descriptor:

derivative2At(...)
Vector = d2At(pos) - Get the second derivative at the given parameter [0|Length] if defined
derivative2At(u,v) -> (vectorU,vectorV)

>>> help(Part.Face.distToShape)

Help on method_descriptor:

distToShape(...)
Find the minimum distance to another shape.
distToShape(shape) -> (dist, vectors, infos)
--
dist is the minimum distance, in mm (float value).

vectors is a list of pairs of App.Vector. Each pair corresponds to solution.
Example: [(Vector (2.0, -1.0, 2.0), Vector (2.0, 0.0, 2.0)), (Vector (2.0,
-1.0, 2.0), Vector (2.0, -1.0, 3.0))] First vector is a point on self, second
vector is a point on s.

infos contains additional info on the solutions. It is a list of tuples:
(topo1, index1, params1, topo2, index2, params2)

topo1, topo2 are strings identifying type of BREP element: 'Vertex',
'Edge', or 'Face'.

index1, index2 are indexes of the elements (zero-based).

params1, params2 are parameters of internal space of the elements. For
vertices, params is None. For edges, params is one float, u. For faces,

params is a tuple (u,v).

>>> help(Part.Face.dumpContent)

Help on method_descriptor:

dumpContent(...)
Dumps the content of the object, both the XML representation as well as the additional datafiles
required, into a byte representation. It will be returned as byte array.
dumpContent() -- returns a byte array with full content
dumpContent(Compression=1-9) -- Sets the data compression from 0 (no) to 9 (max)

>>> help(Part.Face.dumpToString)

Help on method_descriptor:

dumpToString(...)
Dump information about the shape to a string.
dumpToString() -> string

>>> help(Part.Face.exportBinary)

Help on method_descriptor:

exportBinary(...)
Export the content of this shape in binary format to a file.
exportBinary(filename)

>>> help(Part.Face.exportBrep)

Help on method_descriptor:

exportBrep(...)
Export the content of this shape to an BREP file.
exportBrep(filename)
--
BREP is an OpenCasCade native format.

>>> help(Part.Face.exportBrepToString)

Help on method_descriptor:

exportBrepToString(...)
Export the content of this shape to a string in BREP format.
exportBrepToString() -> string
--
BREP is an OpenCasCade native format.

>>> help(Part.Face.exportIges)

Help on method_descriptor:

exportIges(...)
Export the content of this shape to an IGES file.
exportIges(filename)

>>> help(Part.Face.exportStep)

Help on method_descriptor:

exportStep(...)
Export the content of this shape to an STEP file.
exportStep(filename)

>>> help(Part.Face.exportStl)

Help on method_descriptor:

exportStl(...)
Export the content of this shape to an STL mesh file.
exportStl(filename)

>>> help(Part.Face.extrude)

Help on method_descriptor:

extrude(...)
Extrude the shape along a direction.
extrude(direction, length)

>>> help(Part.Face.findPlane)

Help on method_descriptor:

findPlane(...)
return a plane if the shape is planar
findPlane(tol=None) -> Shape

>>> help(Part.Face.fix)

Help on method_descriptor:

fix(...)
Tries to fix a broken shape.
fix(working precision, minimum precision, maximum precision) -> bool
--
True is returned if the operation succeeded, False otherwise.

>>> help(Part.Face.fixTolerance)

Help on method_descriptor:

fixTolerance(...)
Sets (enforces) tolerances in a shape to the given value
fixTolerance(value, [ShapeType=Shape])
--
ShapeType = Vertex : only vertices are set
ShapeType = Edge : only edges are set
ShapeType = Face : only faces are set
ShapeType = Wire : to have edges and their vertices set
ShapeType = other value : all (vertices,edges,faces) are set

>>> help(Part.Face.fuse)

Help on method_descriptor:

fuse(...)
Union of this and a given (list of) topo shape.
fuse(tool) -> Shape
or
fuse((tool1,tool2,...),[tolerance=0.0]) -> Shape
--
Union of this and a given list of topo shapes.

Supports (OCCT 6.9.0 and above):
- Fuzzy Boolean operations (global tolerance for a Boolean operation)
- Support of multiple arguments for a single Boolean operation
- Parallelization of Boolean Operations algorithm

Beginning from OCCT 6.8.1 a tolerance value can be specified.

>>> help(Part.Face.generalFuse)

Help on method_descriptor:

generalFuse(...)
Run general fuse algorithm (GFA) between this and given shapes.
generalFuse(list_of_other_shapes, [fuzzy_value = 0.0]) -> (result, map)
--
list_of_other_shapes: shapes to run the algorithm against (the list is
effectively prepended by 'self').

fuzzy_value: extra tolerance to apply when searching for interferences, in
addition to tolerances of the input shapes.

Returns a tuple of 2: (result, map).

result is a compound containing all the pieces generated by the algorithm
(e.g., for two spheres, the pieces are three touching solids). Pieces that
touch share elements.

map is a list of lists of shapes, providing the info on which children of
result came from which argument. The length of list is equal to length of
list_of_other_shapes + 1. First element is a list of pieces that came from
shape of this, and the rest are those that come from corresponding shapes in
list_of_other_shapes.
hint: use isSame method to test shape equality

Parallelization of Boolean Operations algorithm

OCC 6.9.0 or later is required.

>>> help(Part.Face.getAllDerivedFrom)

Help on method_descriptor:

getAllDerivedFrom(...)
Returns all descendants

>>> help(Part.Face.getElement)

Help on method_descriptor:

getElement(...)
Returns a SubElement
getElement(elementName) -> Face | Edge | Vertex

>>> help(Part.Face.getElementTypes)

Help on method_descriptor:

getElementTypes(...)
Return a list of element types

>>> help(Part.Face.getFaces)

Help on method_descriptor:

getFaces(...)
Return a tuple of points and triangles with a given accuracy

>>> help(Part.Face.getFacesFromSubElement)

Help on method_descriptor:

getFacesFromSubElement(...)
Return vertexes and faces from a sub-element

>>> help(Part.Face.getLines)

Help on method_descriptor:

getLines(...)
Return a tuple of points and lines with a given accuracy

>>> help(Part.Face.getLinesFromSubElement)

Help on method_descriptor:

getLinesFromSubElement(...)
Return vertexes and lines from a sub-element

>>> help(Part.Face.getPoints)

Help on method_descriptor:

getPoints(...)
Return a tuple of points and normals with a given accuracy

>>> help(Part.Face.getTolerance)

Help on method_descriptor:

getTolerance(...)
Determines a tolerance from the ones stored in a shape
getTolerance(mode, ShapeType=Shape) -> float
--
mode = 0 : returns the average value between sub-shapes,
mode > 0 : returns the maximal found,
mode < 0 : returns the minimal found.
ShapeType defines what kinds of sub-shapes to consider:
Shape (default) : all : Vertex, Edge, Face,
Vertex : only vertices,
Edge : only edges,
Face : only faces,
Shell : combined Shell + Face, for each face (and containing
shell), also checks edge and Vertex

>>> help(Part.Face.getUVNodes)

Help on method_descriptor:

getUVNodes(...)
Get the list of (u,v) nodes of the tessellation
getUVNodes() -> list
--
An exception is raised if the face is not triangulated.

>>> help(Part.Face.globalTolerance)

Help on method_descriptor:

globalTolerance(...)
Returns the computed tolerance according to the mode
globalTolerance(mode) -> float
--
mode = 0 : average
mode > 0 : maximal
mode < 0 : minimal

>>> help(Part.Face.hashCode)

Help on method_descriptor:

hashCode(...)
This value is computed from the value of the underlying shape reference and the location.
hashCode() -> int
--
Orientation is not taken into account.

>>> help(Part.Face.importBinary)

Help on method_descriptor:

importBinary(...)
Import the content to this shape of a string in BREP format.
importBinary(filename)

>>> help(Part.Face.importBrep)

Help on method_descriptor:

importBrep(...)
Load the shape from a file in BREP format.
importBrep(filename)

>>> help(Part.Face.importBrepFromString)

Help on method_descriptor:

importBrepFromString(...)
Load the shape from a string that keeps the content in BREP format.
importBrepFromString(string, [displayProgressBar=True])
--
importBrepFromString(str,False) to not display a progress bar.

>>> help(Part.Face.inTolerance)

Help on method_descriptor:

inTolerance(...)
Determines which shapes have a tolerance within a given interval
inTolerance(value, [ShapeType=Shape]) -> ShapeList
--
ShapeType is interpreted as in the method getTolerance

>>> help(Part.Face.isClosed)

Help on method_descriptor:

isClosed(...)
Checks if the shape is closed.
isClosed() -> bool
--
If the shape is a shell it returns True if it has no free boundaries (edges).
If the shape is a wire it returns True if it has no free ends (vertices).
(Internal and External sub-shepes are ignored in these checks)
If the shape is an edge it returns True if its vertices are the same.

>>> help(Part.Face.isCoplanar)

Help on method_descriptor:

isCoplanar(...)
Checks if this shape is coplanar with the given shape.
isCoplanar(shape,tol=None) -> bool

>>> help(Part.Face.isDerivedFrom)

Help on method_descriptor:

isDerivedFrom(...)
Returns true if given type is a father

>>> help(Part.Face.isEqual)

Help on method_descriptor:

isEqual(...)
    Checks if both shapes are equal.
        This means geometry, placement and orientation are equal.
    isEqual(shape) -> bool

>>> help(Part.Face.isInfinite)

Help on method_descriptor:

isInfinite(...)
    Checks if this shape has an infinite expansion.
    isInfinite() -> bool

>>> help(Part.Face.isInside)

Help on method_descriptor:

isInside(...)
    Checks whether a point is inside or outside the shape.
    isInside(point, tolerance, checkFace) => Boolean
    --
    checkFace indicates if the point lying directly on a face is considered to be inside or not

>>> help(Part.Face.isNull)

Help on method_descriptor:

isNull(...)
    Checks if the shape is null.
    isNull() -> bool

>>> help(Part.Face.isPartOfDomain)

Help on method_descriptor:

isPartOfDomain(...)
    Check if a given (u,v) pair is inside the domain of a face
    isPartOfDomain(u,v) -> bool

>>> help(Part.Face.isPartner)

Help on method_descriptor:

isPartner(...)
    Checks if both shapes share the same geometry.
    Placement and orientation may differ.
    isPartner(shape) -> bool

>>> help(Part.Face.isSame)

Help on method_descriptor:

isSame(...)
    Checks if both shapes share the same geometry
        and placement. Orientation may differ.
    isSame(shape) -> bool

>>> help(Part.Face.isValid)

Help on method_descriptor:

isValid(...)
    Checks if the shape is valid, i.e. neither null, nor empty nor corrupted.
    isValid() -> bool

>>> help(Part.Face.limitTolerance)

Help on method_descriptor:

limitTolerance(...)
    Limits tolerances in a shape
    limitTolerance(tmin, [tmax=0, ShapeType=Shape]) -> bool
    --
    tmin = tmax -> as fixTolerance (forces)
    tmin = 0   -> maximum tolerance will be tmax
    tmax = 0 or not given (more generally, tmax < tmin) ->
    tmax ignored, minimum will be tmin
    else, maximum will be max and minimum will be min
    ShapeType = Vertex : only vertices are set
    ShapeType = Edge   : only edges are set
    ShapeType = Face   : only faces are set
    ShapeType = Wire   : to have edges and their vertices set
    ShapeType = other value : all (vertices,edges,faces) are set
    Returns True if at least one tolerance of the sub-shape has been modified

>>> help(Part.Face.makeChamfer)

Help on method_descriptor:

makeChamfer(...)
    Make chamfer.
    makeChamfer(radius,edgeList) -> Shape
    or
    makeChamfer(radius1,radius2,edgeList) -> Shape

>>> help(Part.Face.makeEvolved)

Help on method_descriptor:

makeEvolved(...)
    Profile along the spine

>>> help(Part.Face.makeFillet)

Help on method_descriptor:

makeFillet(...)
    Make fillet.
    makeFillet(radius,edgeList) -> Shape
    or
    makeFillet(radius1,radius2,edgeList) -> Shape

>>> help(Part.Face.makeHalfSpace)

Help on method_descriptor:

makeHalfSpace(...)
    Make a half-space solid by this face and a reference point.
    makeHalfSpace(pos) -> Shape

>>> help(Part.Face.makeOffset)

Help on method_descriptor:

makeOffset(...)
    Offset the face by a given amount.
    makeOffset(dist) -> Face
    --
    Returns Compound of Wires. Deprecated - use makeOffset2D instead.

>>> help(Part.Face.makeOffset2D)

Help on method_descriptor:

makeOffset2D(...)
makes an offset shape (2d offsetting).
makeOffset2D(offset, [join = 0, fill = False, openResult = false, intersection = false]) -> Shape
--
The function supports keyword
arguments. Input shape (self) can be edge, wire, face, or a compound of those.

* offset: distance to expand the shape by. Negative value will shrink the shape.

* join: method of offsetting non-tangent joints. 0 = arcs, 1 = tangent, 2 = intersection

* fill: if true, the output is a face filling the space covered by offset. If false, the output is a wire.

* openResult: affects the way open wires are processed. If False, an open wire is made. If True, a closed wire is made from a double-sided offset, with rounds around open vertices.

* intersection: affects the way compounds are processed. If False, all children are offset independently. If True, and children are edges/wires, the children are offset in a collective manner. If compounding is nested, collectiveness does not spread across compounds (only direct children of a compound are taken collectively).

Returns: result of offsetting (wire or face or compound of those). Compounding structure follows that of source shape.

>>> help(Part.Face.makeOffsetShape)

Help on method_descriptor:

makeOffsetShape(...)
makes an offset shape (3d offsetting).
makeOffsetShape(offset, tolerance, [inter = False, self_inter = False, offsetMode = 0, join = 0, fill = False]) -> Shape
--
The function supports keyword arguments.

* offset: distance to expand the shape by. Negative value will shrink the shape.

* tolerance: precision of approximation.

* inter: (parameter to OCC routine; not implemented)

* self_inter: (parameter to OCC routine; not implemented)

* offsetMode: 0 = skin; 1 = pipe; 2 = recto-verso

* join: method of offsetting non-tangent joints. 0 = arcs, 1 = tangent, 2 = intersection

* fill: if true, offsetting a shell is to yield a solid

Returns: result of offsetting.

>>> help(Part.Face.makeParallelProjection)

Help on method_descriptor:

makeParallelProjection(...)

Parallel projection of an edge or wire on this shape
makeParallelProjection(shape, dir) -> Shape

>>> help(Part.Face.makePerspectiveProjection)

Help on method_descriptor:

makePerspectiveProjection(...)
Perspective projection of an edge or wire on this shape
makePerspectiveProjection(shape, pnt) -> Shape

>>> help(Part.Face.makeShapeFromMesh)

Help on method_descriptor:

makeShapeFromMesh(...)
Make a compound shape out of mesh data.
makeShapeFromMesh((vertex,facets),tolerance) -> Shape
--
Note: This should be used for rather small meshes only.

>>> help(Part.Face.makeThickness)

Help on method_descriptor:

makeThickness(...)
Hollow a solid according to given thickness and faces.
makeThickness(List of faces, Offset (Float), Tolerance (Float)) -> Shape
--
A hollowed solid is built from an initial solid and a set of faces on this solid,
which are to be removed. The remaining faces of the solid become the walls of
the hollowed solid, their thickness defined at the time of construction.

>>> help(Part.Face.makeWires)

Help on method_descriptor:

makeWires(...)
make wire(s) using the edges of this shape
makeWires([op=None])
--
The function will sort any edges inside the current shape, and connect them
into wire. If more than one wire is found, then it will make a compound out of
all found wires.

This function is element mapping aware. If the input shape has non-zero Tag,
it will map any edge and vertex element name inside the input shape into the
itself.

op: an optional string to be appended when auto generates element mapping.

>>> help(Part.Face.mirror)

Help on method_descriptor:

mirror(...)
Mirror this shape on a given plane.
mirror(base, norm) -> Shape
--
The plane is given with its base point and its normal direction.

>>> help(Part.Face.multiFuse)

Help on method_descriptor:

multiFuse(...)
Union of this and a given list of topo shapes.

multiFuse((tool1,tool2,...),[tolerance=0.0]) -> Shape
--
Supports (OCCT 6.9.0 and above):
- Fuzzy Boolean operations (global tolerance for a Boolean operation)
- Support of multiple arguments for a single Boolean operation
- Parallelization of Boolean Operations algorithm

Beginning from OCCT 6.8.1 a tolerance value can be specified.
Deprecated: use fuse() instead.

>>> help(Part.Face.normalAt)

Help on method_descriptor:

normalAt(...)
Get the normal vector at the given parameter [0|Length] if defined
normalAt(pos) -> Vector

>>> help(Part.Face.nullify)

Help on method_descriptor:

nullify(...)
Destroys the reference to the underlying shape stored in this shape.
As a result, this shape becomes null.
nullify()

>>> help(Part.Face.oldFuse)

Help on method_descriptor:

oldFuse(...)
Union of this and a given topo shape (old algorithm).
oldFuse(tool) -> Shape

>>> help(Part.Face.optimalBoundingBox)

Help on method_descriptor:

optimalBoundingBox(...)
Get the optimal bounding box
optimalBoundingBox([useTriangulation = True, useShapeTolerance = False]) -> bound box

>>> help(Part.Face.overTolerance)

Help on method_descriptor:

overTolerance(...)
Determines which shapes have a tolerance over the given value
overTolerance(value, [ShapeType=Shape]) -> ShapeList
--
ShapeType is interpreted as in the method getTolerance

>>> help(Part.Face.project)

Help on method_descriptor:

project(...)
Project a list of shapes on this shape
project(shapeList) -> Shape

>>> help(Part.Face.proximity)

Help on method_descriptor:

proximity(...)
Returns two lists of Face indexes for the Faces involved in the intersection.

proximity(shape,[tolerance]) -> (selfFaces, shapeFaces)

>>> help(Part.Face.read)

Help on method_descriptor:

read(...)
Read in an IGES, STEP or BREP file.
read(filename)

>>> help(Part.Face.reflectLines)

Help on method_descriptor:

reflectLines(...)
Build projection or reflect lines of a shape according to a view direction.
reflectLines(ViewDir, [ViewPos, UpDir, EdgeType, Visible, OnShape]) -> Shape (Compound of edges)
--
This algorithm computes the projection of the shape in the ViewDir direction.
If OnShape is False(default), the returned edges are flat on the XY plane defined by
ViewPos(origin) and UpDir(up direction).
If OnShape is True, the returned edges are the corresponding 3D reflect lines located on the shape.
EdgeType is a string defining the type of result edges :
- IsoLine : isoparametric line
- OutLine : outline (silhouette) edge
- Rg1Line : smooth edge of G1-continuity between two surfaces
- RgNLine : sewn edge of CN-continuity on one surface
- Sharp : sharp edge (of C0-continuity)
If Visible is True (default), only visible edges are returned.
If Visible is False, only invisible edges are returned.

>>> help(Part.Face.removeInternalWires)

Help on method_descriptor:

removeInternalWires(...)
Removes internal wires (also holes) from the shape.
removeInternalWires(minimalArea) -> bool

>>> help(Part.Face.removeShape)

Help on method_descriptor:

removeShape(...)
Remove a sub-shape and return a new shape.
removeShape(shapeList) -> Shape
--
The parameter is a list of shapes.

>>> help(Part.Face.removeSplitter)

Help on method_descriptor:

removeSplitter(...)
Removes redundant edges from the B-REP model
removeSplitter() -> Shape

>>> help(Part.Face.replaceShape)

Help on method_descriptor:

replaceShape(...)
Replace a sub-shape with a new shape and return a new shape.
replaceShape(tupleList) -> Shape
--
The parameter is in the form list of tuples with the two shapes.

>>> help(Part.Face.restrictSurface)

>>> help(Part.Face.restoreContent)

Help on method_descriptor:

restoreContent(...)
Restore the content of the object from a byte representation as stored by "dumpContent".
It could be restored from any python object implementing the buffer protocol.
restoreContent(buffer) -- restores from the given byte array

>>> help(Part.Face.reverse)

Help on method_descriptor:

reverse(...)
Reverses the orientation of this shape.
reverse()

>>> help(Part.Face.reversed)

Help on method_descriptor:

reversed(...)
Reverses the orientation of a copy of this shape.
reversed() -> Shape

>>> help(Part.Face.revolve)

Help on method_descriptor:

revolve(...)
Revolve the shape around an Axis to a given degree.
revolve(base, direction, angle)
--
Part.revolve(Vector(0,0,0),Vector(0,0,1),360) - revolves the shape around the Z Axis 360 degree.

Hints: Sometimes you want to create a rotation body out of a closed edge or wire.
Example:

```
from FreeCAD import Base
import Part
V=Base.Vector

e=Part.Ellipse()
s=e.toShape()
r=s.revolve(V(0,0,0),V(0,1,0), 360)
Part.show(r)
```

However, you may possibly realize some rendering artifacts or that the mesh
creation seems to hang. This is because this way the surface is created twice.
Since the curve is a full ellipse it is sufficient to do a rotation of 180 degree
only, i.e. r=s.revolve(V(0,0,0),V(0,1,0), 180)

Now when rendering this object you may still see some artifacts at the poles. Now the
problem seems to be that the meshing algorithm doesn't like to rotate around a point
where there is no vertex.

The idea to fix this issue is that you create only half of the ellipse so that its shape
representation has vertexes at its start and end point.

```
from FreeCAD import Base
import Part
V=Base.Vector

e=Part.Ellipse()
s=e.toShape(e.LastParameter/4,3*e.LastParameter/4)
r=s.revolve(V(0,0,0),V(0,1,0), 360)
Part.show(r)
```

>>> help(Part.Face.rotate)

Help on method_descriptor:

rotate(...)
Apply the rotation (base,dir,degree) to the current location of this shape
rotate(base,dir,degree)
--
Shp.rotate(Vector(0,0,0),Vector(0,0,1),180) - rotate the shape around the Z Axis 180 degrees.

>>> help(Part.Face.rotated)

Help on method_descriptor:

rotated(...)
Create a new shape with rotation.
rotated(base,dir,degree) -> shape

>>> help(Part.Face.scale)

Help on method_descriptor:

scale(...)
Apply scaling with point and factor to this shape.
scale(factor,[base=Vector(0,0,0)])

>>> help(Part.Face.scaled)

Help on method_descriptor:

scaled(...)
Create a new shape with scale.
scaled(factor,[base=Vector(0,0,0)]) -> shape

>>> help(Part.Face.section)

Help on method_descriptor:

section(...)
Section of this with a given (list of) topo shape.
section(tool,[approximation=False]) -> Shape
or
section((tool1,tool2,...),[tolerance=0.0, approximation=False]) -> Shape
--
If approximation is True, section edges are approximated to a C1-continuous BSpline curve.

Supports:
- Fuzzy Boolean operations (global tolerance for a Boolean operation)
- Support of multiple arguments for a single Boolean operation (s1 AND (s2 OR s3))
- Parallelization of Boolean Operations algorithm

OCC 6.9.0 or later is required.

>>> help(Part.Face.sewShape)

Help on method_descriptor:

sewShape(...)
Sew the shape if there is a gap.
sewShape()

>>> help(Part.Face.slice)

Help on method_descriptor:

slice(...)

Make single slice of this shape.
slice(direction, distance) --> Wires

>>> help(Part.Face.slices)

Help on method_descriptor:

slices(...)
Make slices of this shape.
slices(direction, distancesList) --> Wires

>>> help(Part.Face.tangentAt)

Help on method_descriptor:

tangentAt(...)
Get the tangent in u and v isoparametric at the given point if defined
tangentAt(u,v) -> Vector

>>> help(Part.Face.tessellate)

Help on method_descriptor:

tessellate(...)
Tessellate the shape and return a list of vertices and face indices
tessellate() -> (vertex,facets)

>>> help(Part.Face.toNurbs)

Help on method_descriptor:

toNurbs(...)
Conversion of the complete geometry of a shape into NURBS geometry.
toNurbs() -> Shape
--
For example, all curves supporting edges of the basis shape are converted
into B-spline curves, and all surfaces supporting its faces are converted
into B-spline surfaces.

>>> help(Part.Face.transformGeometry)

Help on method_descriptor:

transformGeometry(...)
Apply geometric transformation on this or a copy the shape.
transformGeometry(matrix) -> Shape
--
This method returns a new shape.
The transformation to be applied is defined as a 4x4 matrix.
The underlying geometry of the following shapes may change:
- a curve which supports an edge of the shape, or
- a surface which supports a face of the shape;

For example, a circle may be transformed into an ellipse when
applying an affinity transformation. It may also happen that
the circle then is represented as a B-spline curve.

The transformation is applied to:
- all the curves which support edges of the shape, and
- all the surfaces which support faces of the shape.

Note: If you want to transform a shape without changing the
underlying geometry then use the methods translate or rotate.

>>> help(Part.Face.transformShape)

Help on method_descriptor:

transformShape(...)
Apply transformation on a shape without changing the underlying geometry.
transformShape(Matrix,[boolean copy=False, checkScale=False]) -> None
--
If checkScale is True, it will use transformGeometry if non-uniform
scaling is detected.

>>> help(Part.Face.transformed)

Help on method_descriptor:

transformed(...)
Create a new transformed shape
transformed(Matrix,copy=False,checkScale=False,op=None) -> shape

>>> help(Part.Face.translate)

Help on method_descriptor:

translate(...)
Apply the translation to the current location of this shape.
translate(vector)

>>> help(Part.Face.translated)

Help on method_descriptor:

translated(...)
Create a new shape with translation
translated(vector) -> shape

>>> help(Part.Face.validate)

Help on method_descriptor:

validate(...)
Validate the face.
validate()

>>> help(Part.Face.valueAt)

Help on method_descriptor:

valueAt(...)
Get the point at the given parameter [0|Length] if defined
valueAt(u,v) -> Vector

>>> help(Part.Face.writeInventor)

Help on method_descriptor:

writeInventor(...)
Write the mesh in OpenInventor format to a string.
writeInventor() -> string