```
import uuid
import os
from pathlib import Path

import reporting

def is_real_number(x):
    #return isinstance(x, (int, float))  # need numpy.int32, int36, float64, etc
    try:
        xx = float(x)
        return True
    except Exception:
        return False

class UserGeometryBase():

    def __init__(self, params, d_materials):
        self._geom_name = ''
        self._num_type_materials=0
        self._is_curvilinear=False
        self._d_materials = d_materials
        self._d_params = params
        self._descrip = 'Unimplemented user template'
        self._gmsh_template_filename = ''  # internal meshes use this. User meshes should not

        self._req_params =[]
        self._allowed_params =[]
        self._num_named_materials = 0

    def set_properties(self, nm, n_materials, is_curvi, desc):
        self.set_name(nm)
        self.set_num_type_materials(n_materials)
        self.set_is_curvilinear(is_curvi)
        self.set_description(desc)

    def set_required_parameters(self, l_nms, num_mats):
        self._req_params.extend(['domain_x','domain_y', 'lc_bkg'])
        self._req_params.extend(l_nms)
        self._num_named_materials = num_mats

    def set_allowed_parameters(self, l_nms, num_allowed_mats):
        self._allowed_params.extend(l_nms)
        for im in range(num_allowed_mats): # need mat_a, mat_b, mat_c, etc
            self._allowed_params.append('material_'+'abcdefghijklmnopqrtstuvwxyz'[im])

    def set_parameter_help(self, d_help):
        self.d_param_help = {
            'domain_x': "length of simulation domain along x",
            'domain_y': "length of simulation domain along y"}
        self.d_param_help.update(d_help)

    def help_on_parameters(self):
        print(self.get_parameter_help_string())

    def get_parameter_help_string(self):
        msg = f'Waveguide parameters for shape {self._geom_name}:\n'
        for k,v in self.d_param_help.items():
            msg += f'{k:>20} : {v}\n'
        return msg

    def check_parameters(self, user_params):
```

```
        if not self._req_params: # not yet defined for this template
            return

        reqkws = self._req_params
        reqkws.append('material_bkg')

        for im in range(self._num_named_materials-1): # need mat_a, mat_b, mat_c, etc, -1 because one is mat_bkg
            reqkws.append('material_'+'abcdefghijklmnopqrtstuvwxyz'[im])

        for key in reqkws:
            if key not in user_params:
                msg =f"Waveguide type '{self._geom_name}' requires a value for the parameter '{key}' in the call to make_structure()."

                msg+= '\n\nNote that some waveguide types have changed their required parameters to adopt more intuitive names.'

                msg+=f'\n\nFor this waveguide type, the following guidelines apply:\n\n'

                msg+=self.get_parameter_help_string()

                reporting.report_and_exit(msg)

        # report unexpected keys
        goodkeys = reqkws + self._allowed_params
        goodkeys.append('lc') # remove special case once everything is moved to lc_bkg
        for key in user_params.keys():
            if key not in goodkeys:
                reporting.report(
                    f"Waveguide '{self._geom_name}' will ignore the parameter '{key}' in make_structure().")


    def check_dimensions(self):  # override to implement check for each mesh design
        dims_ok = True
        msg=''
        return dims_ok, msg

    def validate_dimensions(self):
        '''Checks that the combination of user parameters defines a well-defined consistent geometry.'''

        dims_ok, msg = self.check_dimensions()

        if not dims_ok: reporting.report_and_exit(f'There is a problem with the waveguide structure:\n{msg}')

    def set_num_type_materials(self, n):
        self._num_type_materials = n

    def set_is_curvilinear(self, b):
        self._is_curvilinear = b

    def set_name(self, nm):
        self._geom_name = nm

    def set_description(self, desc):
        self._descrip = desc

    def get_param(self, k):
        return self._d_params.get(k, None)

    def geom_name(self):
        return self._geom_name

    def gmsh_template_filename(self):
        if self._gmsh_template_filename:
            return self._gmsh_template_filename
        else:
            return self.geom_name()

    def num_type_materials(self):
        return self._num_type_materials

    def is_curvilinear(self):
        return self._is_curvilinear

    def __str__(self):
        return self._descrip

    def apply_parameters(self):
        print('IMPLEMENT ME', __file__, 'make_geometry')
        return ''

    def make_geometry(self, p_dir_templates):
        subs = self.apply_parameters()

  #$      msh_template = self.wg_geom.gmsh_template_filename()
#$
    #        geo = self._load_mesh_template(msh_template)

        geo = open(Path(p_dir_templates,
                   f'{self.gmsh_template_filename()}_msh_template.geo'), 'r').read()

        for (olds, news, val) in subs:
            if val is None:  # unset value not overridden or dropped
                continue
            elif is_real_number(val):
                geo = geo.replace(olds, news % val)
            else:
                geo = geo.replace(olds, news)

        return geo


    def get_instance_filename(self):   # , l_dims):
        '''Make name for the concrete instantiation of a given mesh .geo template'''
        msh_fname = self._geom_name

        # made crazy long names, not helping
        # for v in l_dims:
        #    if is_real_number(v): msh_name += '_%s' % dec_float_str(v)

        msh_fname += f'--pid-{os.getpid()}'

        # need to make name unique to support parallel processing
        msh_fname += '--'+str(uuid.uuid4())

        return msh_fname

    def draw_mpl_frame(self, ax):
        '''Add elements to a matplotlib axis to draw outline of the structure'''
```

```
        pass
        # print('base class drawmplframe on ', ax)
```