In [ ]:

```
from ambiance import Atmosphere
import os
import sys
import re
import subprocess
import time
'''
# class Plane:
#     def __init__(self,surfaces):
#         self.surfaces = surfaces
#         # print(range(0,len(self.surfaces[0].sections[:-1])-2))
#         # print(len(self.surfaces[0].sections))
#         self.Sref = 0
#         for i in range(0,len(self.surfaces[0].sections)-1):

#             # self.Sref += self.surfaces.sections
#             self.Sref +=((self.surfaces[0].sections[i+1][1]+self.surfaces[0].sections[i][1])*(self.surface
#         self.Bref = 2*self.surfaces[0].sections[-1][0][1]
#         self.Cref = self.Sref/self.Bref
#         # for i = len(self.surfaces[0].sections[:-1])
#         # self.Sref = self.surfaces[0].sections[:-1]




#         # self.bodies = bodies

# class Surface:
#     def __init__(self, name, symmetry, component, translate, incidence, Nchord, Cspace, sections):
#         """
#         name: string
#             Name of the Surface
#         symmetry: boolean
#             0 = no symmetry
#             1 = symmetry over X-Z plane
#         component: integer
#             Component #
#         translate: vector
#             Location of LE at root chord
#         incidence: float
#             Incidence Angle
#         Nchord: integer
#             Number of chordwise points
#         Cspace: integer
#             Chordwise point distribution
#         sections: array
#             Section Array
#         """
#         self.name = name
#         self.symmetry = symmetry
#         self.component = component
#         self.translate = translate
#         self.incidence = incidence
#         self.Nchord = Nchord
#         self.Cspace = Cspace
#         self.sections = sections

# def WriteSection(section):
#     # section = [[Xle,Yle,Zle],chord,angle,Nspan,Sspace,airfoil]
#     sect = "SECTION\n{} {} {} {} {} {} {} \nAFIL\nAirfoils\\{}\n".format(section[0][0],section[0][1],secti
#     return sect

# def WriteSurface(surface):
#     surf = "SURFACE\n{}\n{} {}\nCOMPONENT\n{}\n".format(surface.name,surface.Nchord, surface.Cspace,surfac
#     if surface.symmetry == 1:
#         surf = surf+"YDUPLICATE\n0.0\n"
#     surf = surf + "TRANSLATE\n{} {} {}\nANGLE\n{}\n".format(surface.translate[0],surface.translate[1],surf
#     for section in surface.sections:
#         surf = surf + WriteSection(section)
#     # print(surf)
#     return surf
```

```
# def ComponentHeader(text):
#     text = text.replace("", " ")[1: -1]
#     length = len(text)
#     header = ["\n"+"#" * (length+8)+"\n","#   {}   #\n".format(text),"#" * (length+8)+"\n"]
#     return header

# def SectionHeader(text):
#     header = "###  {}  ###\n".format(text.replace("", " ")[1: -1])
#     return header

# def CreateAVLPlane(name,mach,plane):
#     # print('AAA')
#     cd = os.getcwd() # Get current location of python script
#     if not os.path.exists('{}\\Planes'.format(cd)): # create folders if nonexistent
#         os.mkdir('{}\\Planes'.format(cd))
#         print("/Planes folder created")
#     if not os.path.exists('{}\\Planes\\{}'.format(cd,name)):
#         os.mkdir('{}\\Planes\\{}'.format(cd,name))
#         print("Planes/{} folder created".format(name))


#     # Cref = 10/

#     with open('{}\\Planes\\{}\\{}.avl'.format(cd,name,name),'w') as file:
#         file.write('{}\n{}\n0 0 0\n{} {} {}\n0 0 0\n0\n'.format(name,mach,plane.Sref,plane.Cref,plane.Bref
#         for surface in plane.surfaces:
#             file.write(WriteSurface(surface))

#         # file.writelines(ComponentHeader('MASS DEFINTION'))
#         # file.write(SectionHeader('MASS DEFINTION'))
'''
class AVL:
    def __init__(self):
        self.inputList = ''
        self.cd = os.getcwd()
        # print(self)
    def addInput(self,input):
        self.inputList += '{}\n'.format(input)
        # print(self.inputList)
    def clearInput(self):
        self.inputlist = ''
    def runAVL(self): # opens avl and runs all of the stored commands
        self.AVLsp = subprocess.Popen('{}\\avl.exe'.format(self.cd),
            shell=False,
            stdin=subprocess.PIPE,
            stdout=open('AVLsession.log', 'w'),         # Put the output of this terminal into the open log
            stderr=subprocess.PIPE)
        self.AVLsp.stdin.write(self.inputList.encode('utf-8'))
        self.AVLsp.stdin.flush()
        self.AVLsp.communicate()

        # print(self.inputList)
        # log = open('AVLsession.log').read()
        # os.path.getsize('AVLsession.log')
        # print(len(log))
        # start = log.rfind('  Alpha =')
        # end = log.rfind('| Plane')
        # caseOutput = log[start:end].replace('| Trefftz','').replace('=',' ').replace('**********','-1').sp
        # caseData = dict(zip(caseOutput[::2],list(map(float,caseOutput[1::2]))))
        # return caseData
    def oper(self):
        self.addInput('\n \n \n')
        self.addInput('oper')
    def loadPlane(self,plane):
        self.addInput('load Planes\\{}\\{}.avl'.format(plane,plane))
    def loadMass(self,plane):
        self.addInput('mass Planes\\{}\\{}.mass'.format(plane,plane))
        self.addInput('mset\n0')
    def setAtmosphere(self,altitude=0,temp_offset=0):
        # convert to units
        altitude = altitude/3.28084 # ft to m
        temp_offset = 5/9*temp_offset # F to C
        self.addInput('oper')
        self.addInput('M')
```

```
        self.addInput('G 32.17')
        atmo = Atmosphere(altitude)
        # print((atmo.temperature/(atmo.temperature+temp_offset)))
        self.addInput('D {}'.format((atmo.temperature[0]/(atmo.temperature[0]+temp_offset))*atmo.density[0]/
        self.addInput('\n')
    def setVelocity(self,velocity):
        self.addInput('oper')
        self.addInput('M')
        self.addInput('G 32.17')
        self.addInput('V {}'.format(velocity))
        self.addInput('\n')
    def saveOutput(self,output,name=0):
        self.addInput('MRF')
        self.addInput(output)
        if name == 0:
            self.addInput('FT.out')
        else:
            self.addInput('{}.out'.format(name))
        self.addInput('O\n')

    def readFT(self,name=0):
        if name == 0:
            FT = open('FT.out')
        else:
            FT = open(name)
        FTout = FT.readlines()
        out = {}
        for i in [7,8,11,12,13,14,15,16,17,18,19,20]:
            newLine = FTout[i].replace('\n','').replace(',',' ').replace('|',' ').replace('Trefftz Plane: ',
            h = int(len(newLine)/2) # get half length
            for j in range(h):
                out[newLine[j+h]] = float(newLine[j])
        FT.close()
        return out

    # def readFT(self):
    #     self.addInput('MRF')
    #     self.addInput('FT')
    #     self.addInput('FT.out')
    #     self.addInput('O\n')

    #     FTout = open('FT.out').read().replace('Trefftz Plane: ','').replace(',','').replace('E','e').split
    #     FTout=FTout[7:9]+FTout[11:21]
    #     print(FTout,'\n\n')
    #     FTout = [i.split("|") for i in FTout]
    #     print(FTout,'\n\n')
    #     print(FTout[0][0])
        # outputData = [dict(zip(i[:][1::2],i[:][::2])) for i in FTout]

        # print(outputData)

        #  = FTout[1::2]
        # values =  FTout[::2]
        # print(variables,values)
        # caseData = dict(zip(FTout[1::2],list(FTout[::2])))
        # print(FTout[1::2],FTout[::2])
        # return FTout




'''
# def alpha(plane,alphas):
#     AVLsp = AVL()
#     AVLsp.addInput('load Planes\\{}\\{}'.format(plane,plane))
#     AVLsp.addInput('oper')
#     for alpha in alphas:
#         AVLsp.addInput('A')
#         AVLsp.addInput('A')
#         AVLsp.addInput('{}'.format(alpha))
#         AVLsp.addInput('X')
#         output = AVLsp.runAVL()
#         # CLCD = output['CLtot']/output['CDtot']
```

```
#         # e = output['e']
#         # print(CLCD)
#         return output
#         AVLsp.clearInput()
#     # AVLsp.readAVL(['e','Alpha','CLtot'])
# def CL(plane,CLs):
#     AVLsp = AVL()
#     AVLsp.addInput('load Planes\\{}\\{}'.format(plane,plane))
#     AVLsp.addInput('oper')
#     for CL in CLs:
#         AVLsp.addInput('A')
#         AVLsp.addInput('C')
#         AVLsp.addInput('{}'.format(CL))
#         AVLsp.addInput('X')
#         output = AVLsp.runAVL()
#         # CLCD = output['CLtot']/output['CDtot']
#         # e = output['e']
#         # print(CLCD)
#         return output
#         AVLsp.clearInput()
# if __name__ == "__main__":
#     name = 'TestPlane'
# mach = 0
# # sections = [[Xle,Yle,Zle],chord,angle,Nspan,Sspace,airfoil]
# mainWing = Surface('MainWing',1,1,[0,0,0],0,10,1,
#     [
#     [[0,0,0],1,0,10,1,'S4022.dat'],
#     [[.25,5,0],.5,0,10,1,'S4022.dat'],
# ])
# hStab = Surface('hStab',1,2,[5,0,.5],0,5,1,
#                 [
#     [[0,0,0],.5,0,10,1,'S4022.dat'],
#     [[.125,2,0],.25,0,10,1,'S4022.dat'],
#                 ])
# TestPlane = Plane([mainWing,hStab])
# # TestPlane = Plane(surfs,bodies)
# CreateAVLPlane(name,mach,TestPlane)
# AVLsp = AVL()
# AVLsp.addInput('load Planes/{}/{}.avl'.format(name,name))
# AVLsp.addInput('oper')
# AVLsp.addInput('A')
# AVLsp.addInput('C')
# AVLsp.addInput('.6')
# AVLsp.addInput('C1')
# AVLsp.addInput('X')
# AVLsp.addInput('1')
# AVLsp.addInput('')
# AVLsp.addInput('X')
# AVLsp.runAVL()
'''
```