### route  
#version 1.0  
###Author Austin Kelly  
####This module generates a five leg and six leg journey from a set of 5 airports when traveling in a given aircraft. it will calculate  
#####the cheapest route and associated cost and distance travelled  
#####  
  
  
**from** currencyrate **import** \*  
**from** \_\_airport **import** \*  
**from** countrycurrency **import** \*  
**from** calculatedistance **import** \*  
**from** itertools **import** \*  
**from** legdistances **import** \*  
**from** aircraft **import** \*  
  
  
  
**class Route**:  
 ###Route class will initiallize with 5 ports and aircraft  
 **def** \_\_init\_\_(self,port1,port2,port3,port4,port5,craft='777',genperms=**True**): ##genperm can be set to False to turn off the permutations and refactor and use class to be of use in other apps  
 self.port1 = Airport(port1)  
 self.port2 = Airport(port2)  
 self.port3 = Airport(port3)  
 self.port4= Airport(port4)  
 self.port5 = Airport(port5)  
 self.port6 = Airport(port1)## need to reasign these for permutations  
 self.port7=Airport(port1)## defaults are home  
 self.genperms=genperms  
  
 #self.calclegs()  
 self.portlist=[self.port1,self.port2,self.port3,self.port4,self.port5,self.port6,self.port7] # a convenience array of airports in submitted route  
 self.craft = Aircraft(craft)##create an aircraft object from the craft parameter  
 # self.portlookup={port1: self.port1,port2: self.port2,port3: self.port3,port4: self.port4,port5: self.port5} ## handy field to access todo rermove this nonsense  
 # print(self.portlookup)  
 # print('CHECK CODE LOOK UP', self.portlookup['TNI'])  
 # print(self.port1.code, self.port2.code, self.port3.code, self.port4.code, self.port5.code, self.port6.code, self.port7.code)  
 # print('aircratf model :',self.craft.model)  
 #print('range :',self.craft.range)  
 **if** self.genperms==**True** :  
 candidates={}  
 route5 = list(permutations([self.port2.code,self.port3.code,self.port4.code,self.port5.code]))  
 # print(route5)  
 route6a = list(permutations([self.port2.code,self.port2.code, self.port3.code, self.port4.code, self.port5.code]))  
 #print(route6a)  
 route6b = list(permutations([self.port2.code,self.port3.code ,self.port3.code, self.port4.code, self.port5.code]))  
 # print(route6b)  
 route6c = list(permutations([self.port2.code, self.port3.code, self.port4.code,self.port4.code, self.port5.code]))  
 # print(route6c)  
 route6d = list(permutations([self.port2.code,self.port3.code, self.port4.code,self.port5.code,self.port5.code]))  
 # print(route6d)  
 # print(len(route6a)) check number of permutations  
  
 **for** index, eachlist **in** (enumerate(route5)): # will set off and return to home to build a complete route from each permutation  
 candidates[index] =[port1] + list(eachlist) + [port1]  
 **for** index, eachlist **in** (enumerate(route6a)):  
 candidates[24+index] = [port1] + list(eachlist) + [port1]  
 **for** index, eachlist **in** (enumerate(route6b)):  
 candidates[144 + index] = [port1] + list(eachlist) + [port1]  
 **for** index, eachlist **in** (enumerate(route6c)):  
 candidates[264 + index] = [port1] + list(eachlist) + [port1]  
 **for** index, eachlist **in** (enumerate(route6d)):  
 candidates[384 + index] = [port1] + list(eachlist) + [port1]  
 self.candidates=candidates #these are the possible routes  
 #print(self.candidates)  
 #print(len(self.candidates))  
 # self.portsinrange()  
 **else**:**pass** #self.findbestroute()  
  
  
 # def calclegs(self):  
 # self.leg1 = int(Calcdistance(self.port1.longitude, self.port1.latitude,self.port2.longitude,self.port2.latitude).distance)  
 # self.leg2 = int(Calcdistance(self.port2.longitude,self.port2.latitude,self.port3.longitude,self.port3.latitude).distance)  
 # self.leg3 = int(Calcdistance(self.port3.longitude,self.port3.latitude,self.port4.longitude,self.port4.latitude).distance)  
 # self.leg4 = int(Calcdistance(self.port4.longitude,self.port4.latitude,self.port5.longitude,self.port5.latitude).distance)  
 # self.leg5 = int(Calcdistance(self.port5.longitude,self.port5.latitude,self.port6.longitude,self.port6.latitude).distance)  
 # self.leg6 = int(Calcdistance(self.port6.longitude,self.port6.latitude,self.port7.longitude,self.port7.latitude).distance)  
 # self.legslist = [self.leg1, self.leg2, self.leg3 ,self.leg4, self.leg5, self.leg6]  
 #  
 # self.routedist = (self.leg1 + self.leg2 + self.leg3 + self.leg4 + self.leg5 + self.leg6 )  
  
 # print('leg1', self.leg1,' leg2', self.leg2,' leg3', self.leg3,' leg4', self.leg4,' leg5', self.leg5,' leg6', self.leg6)  
 #print(self.routedist)  
 ##########################################self.cleanup\_candidate\_routes()  
 self.cleanup\_candidate\_routes()  
 self.portsinrange()  
 self.cheapest()  
# craft= Aircraft('777')  
# print('range :',craft.range)  
  
  
 **def cleanup\_candidate\_routes**(self):####GET RID OF ROUTES THAT HAVE A LEG GREATER THAN FUEL TANK CAPACITIY  
 todelete = []  
 maxfuelrange = self.craft.range ## get fuel tank range  
 **for** routenumber, route **in** self.candidates.items():  
 # get each candidate route  
 **if** routenumber < 24: # for five legged journeys  
 legslist = Legdistances(route[0], route[1], route[2], route[3], route[4], route[5]).legslist  
 legslist.pop()  
 **for** leg **in** legslist:  
 **if** leg > maxfuelrange:  
 todelete.append(routenumber)  
 **break** ####note if not here it will evaluate again for a second leg not valid therefore get routenumber appended a couple of times CAREFUL  
  
 **if** routenumber >= 24: ##then will have 7 airports and 6 legs  
 legslist = Legdistances(route[0], route[1], route[2], route[3], route[4], route[5], route[6]).legslist  
 # print(candkey,' ',legslist)  
 **for** leg **in** legslist:  
 **if** leg > maxfuelrange:  
 todelete.append(routenumber)  
 **break** #print(todelete) todo  
  
 **for** each **in** todelete:  
 #print('will delete',each)  
 **del** self.candidates[each]  
 #print(self.candidates) todo  
  
  
  
  
 **def portsinrange**(self):# FOR EACH ROUTE GENERATE A SET OF SUBROUTES WITH AIRPORT AHEAD WITHIN FLYING RANGE OF ANY GIVEN PORT FOR EACH HUB  
  
 self.candidates\_legs={}  
 todelete=[]  
 maxfuelrange = self.craft.range  
 self.inrange = {} ##THIS WILL HOLD THE LIST OF SUBROUTES KEYED WITH SAME KEY AS THE MASTER ROUTE  
 **for** routenumber, route **in** self.candidates.items(): ## a route is a collection of airports todo call dictionary something info  
 #todo want to collect dictionary of reachable ports and store them against routenumber  
 aroutessubroutecollection=[]  
  
 **if** routenumber < 24: # for five legged journeys  
 legsdist = Legdistances(route[0], route[1], route[2], route[3], route[4], route[5]).legslist  
 legsdist.pop()  
 self.candidates\_legs[routenumber]=legsdist  
 # print(route)  
 #print('legsdist',legsdist)  
 aportssubroute=[] #todo kk  
  
 **for** port **in** range(5):#todo for each port on a route build its itinary and store each itinary as an elemment  
 #print('just to know what route working for route number',routenumber ,route)  
  
 # print('at port',route[port])##check port name starting from  
 #prepare a list to be filled by second members clicks second loop will add a entry on each of its clicks  
 #print('fresh list for to fill',aportssubroute)  
 # list i in dictionary info\_about will store the chain of airports reachable from ith airport on a fuel tank  
 fuel = maxfuelrange #from each port can start with full tank  
 #print('start fuel',fuel)  
 aportssubroute = []  
 **for** leg **in** range(port, 5): #second loop to fill work list with reachable airportsfor any given port  
 #print(' at port', route[port]) #for each port will look at successive legs and see if can reach it  
 #print('leg',leg)  
 #build chain  
  
 **if** legsdist[leg]>fuel:  
 # append port and its reachable chain into database about a rounte dictionary as finished building chain due to rest arenot reachable  
 # here is where you need append info to the dictionary about a route as are about to leave the and may also need to update if reach end of loop  
 ##when port == 4 or maxed out about to finish with this route so updat its record  
 #print('aportssubroute',aportssubroute)  
 #print('entered break')  
 aroutessubroutecollection.append(aportssubroute)  
 **break** #stop building the chain when a leg to far is reached  
  
 **else**:#  
 aportssubroute.append(route[leg + 1])#append port ahead  
 #print('oringinal list',route,'could reach so append port to listforeachport',aportssubroute)  
 fuel= fuel - legsdist[leg]  
  
 **if** leg==4:  
 aroutessubroutecollection.append(aportssubroute)  
 **break** ##donot need this break but good to highlight logic  
  
 self.inrange[routenumber]=aroutessubroutecollection  
  
 **if** routenumber >= 24: # for five legged journeys  
 legsdist = Legdistances(route[0], route[1], route[2], route[3], route[4], route[5],route[6]).legslist  
 self.candidates\_legs[routenumber] = legsdist  
 # print(route)  
 # print('legsdist',legsdist)  
 aportssubroute = []  
  
 **for** port **in** range(  
 6): # todo for each port on a route build its itinary and store each itinary as an elemment  
 # print('just to know what route working for route number',routenumber ,route)  
  
 # print('at port',route[port])##check port name starting from  
 # prepare a list to be filled by second members clicks second loop will add a entry on each of its clicks  
 # print('fresh list for to fill',aportssubroute)  
 # list i in dictionary info\_about will store the chain of airports reachable from ith airport on a fuel tank  
 fuel = maxfuelrange # from each port can start with full tank  
 # print('start fuel',fuel)  
 aportssubroute = []  
 **for** leg **in** range(port,  
 6): # second loop to fill work list with reachable airportsfor any given port  
 # print(' at port', route[port]) #for each port will look at successive legs and see if can reach it  
 # print('leg',leg)  
 # build chain  
  
 **if** legsdist[leg] > fuel:  
 # append port and its reachable chain into database about a rounte dictionary as finished building chain due to rest arenot reachable  
 # here is where you need append info to the dictionary about a route as are about to leave the and may also need to update if reach end of loop  
 ##when port == 4 or maxed out about to finish with this route so updat its record  
 # print('aportssubroute',aportssubroute)  
 # print('entered break')  
 aroutessubroutecollection.append(aportssubroute)  
 **break** # stop building the chain when a leg to far is reached  
  
 **else**: #  
 aportssubroute.append(route[leg + 1]) # append port ahead  
 # print('oringinal list',route,'could reach so append port to listforeachport',aportssubroute)  
 fuel = fuel - legsdist[leg]  
  
 **if** leg == 5:  
 aroutessubroutecollection.append(aportssubroute)  
 **break** ##donot need this break but good to highlight logic  
 self.inrange[routenumber] = aroutessubroutecollection  
 # print('routenumber',routenumber,'route', route) todo  
 #print(self.candidates\_legs[routenumber]) todo  
 # print('inrangeport', self.inrange[routenumber]) todo  
  
  
  
  
  
 **def cheapest**(self):  
 self.fuelstrategy=[]  
  
 cheapestsofar=999999999999999  
 costofroute=999999999999999999999  
 cheapestdistance=0  
 cheapestroutelist=[]  
 costofcheapest=9999999999999999999  
 self.fuelstrategy=[]  
 self.purchase\_strategy=[]  
  
  
 **for** routenumber, route **in** self.candidates.items():  
  
  
  
  
 **if** routenumber >= 24: ##for routes with six legs  
 ##set up some worker variables used later  
 currentfuel = 0  
 fueltobuy = 0  
 fuelstrategy=[]  
 purchase\_strategy=[]  
 legs = self.candidates\_legs[routenumber] ##generate the six legs list  
 subroutes = self.inrange[routenumber] # a list of subroutes for each port stopped at en route  
 # print('routenumber', routenumber, 'route', route)  
 # print('subroutes for routenumber', routenumber, 'subroutes', subroutes)  
 airports = [Airport(route[0]), Airport(route[1]), Airport(route[2]), Airport(route[3]),  
 Airport(route[4]), Airport(route[5]),Airport(route[6])]  
 # print('airport index 0 airportcode',Airport(route[0]).code)  
 costofthisroute = 9999999999999999999  
 fuelstrategyatthisport = []  
 totaldistance = legs[0] + legs[1] + legs[2] + legs[3] + legs[4] + legs[5]  
 # print('totaldistance', totaldistance)  
 accumulated\_distance = 0  
 distanceleft = totaldistance  
 fuelboughtatthisroute = 0  
 # print('entering subroutes')  
 distancetravelled = 0  
 costfuelbought = 0  
 costtobuy = 0  
  
 ## todo  
  
 # as you visit each port see if you should buy fuel and how much  
 # todo self.craft.fuel exist use it  
 # print('about to enter for ateachport in range(5) : line 326')  
 **for** stopped\_at\_port **in** range(  
 6): # load in a given ports subroute nb ateachport is the index of port in mainroute and first port ahead of it is ateachport+1  
  
 subroute = subroutes[stopped\_at\_port] ##get a list of subroute in range from a st0pped at ports  
 **if** stopped\_at\_port == 0:  
 fuelused = 0  
 distanceleft = totaldistance ##TODO  
  
 **else**:  
 fuelused = legs[stopped\_at\_port - 1] ##fuel used is fuel from last port  
 distancetravelled += legs[stopped\_at\_port - 1]  
 currentfuel -= fuelused  
 distanceleft = totaldistance - distancetravelled  
 # print('portindex',ateachport,'subroute',subroute)  
 fuelboughtatthisport = 0  
 ##initialize so variables for each iteration  
 startleg = stopped\_at\_port ##first leg index on subroutes is the same as the index of stopped at port  
 indexofcheaper = stopped\_at\_port ##initially set cheapest port in the subroute to BE the stopped at port until find otherwise  
 cheapestrate = airports[  
 stopped\_at\_port].currencyeurorate ## initially set cheapest rate in the subroute to the stopped at ports  
  
 ##tood# print('about to enter for idx2,aheadports in enumerate(subroute): ')  
 **for** idx2, aheadports **in** enumerate(subroute): ## see if there is a cheaper port ahead within range  
 distancetocheaper = 0 ##will buy fuel to get to cheaper port  
  
  
 aheadportsindex = stopped\_at\_port + 1 + idx2 ##get the proper index of a port in subroute  
 # print('aheadports indexs', aheadportsindex)  
 **if** (airports[aheadportsindex].currencyeurorate <= cheapestrate): ##find next cheapest airport as will buy enough fuel to get to it  
 indexofcheaper = aheadportsindex # reset to index of cheapest port  
 cheapestrate = airports[aheadportsindex].currencyeurorate  
 **break  
 else**:  
 **pass** # reset to index of cheapest port  
 # legidforcheapest = stopped\_at\_port + 1 + idx2  
 # print('in for x in range ..ateach to indexof cheaper')  
 **if** stopped\_at\_port == indexofcheaper:##there is no cheaper airport in range  
 **if** distanceleft <= self.craft.range:  
 fueltobuy = distanceleft - currentfuel  
 costtobuy = (airports[stopped\_at\_port].currencyeurorate) \* (fueltobuy)  
 currentfuel = currentfuel + fueltobuy  
 costfuelbought += costtobuy  
 **if** stopped\_at\_port == 0:  
 fuelstrategy = [fueltobuy]  
 purchase\_strategy=[costtobuy]  
 **else**:  
 fuelstrategy = fuelstrategy + [fueltobuy]  
 purchase\_strategy= purchase\_strategy + [costtobuy]  
 **else**: ##distanceleft >range yet current is cheapest in range so max up to range  
 fueltobuy = (self.craft.range - currentfuel) ##top up to max  
 costtobuy = (airports[stopped\_at\_port].currencyeurorate) \* (fueltobuy)  
 currentfuel = currentfuel + fueltobuy  
 **if** stopped\_at\_port == 0:  
 fuelstrategy = [fueltobuy]  
 purchase\_strategy = [costtobuy]  
 **else**:#append fuel and cost to their arrays  
 fuelstrategy = fuelstrategy + [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
 **else**: ## there is a cheaper airport in reach so refuel at, so compute distance to it and top up to reach it  
 **for** x **in** range(stopped\_at\_port, indexofcheaper+1):  
 distancetocheaper += legs[x]  
 fueltobuy = distancetocheaper - currentfuel  
 costtobuy = (airports[stopped\_at\_port].currencyeurorate) \* (fueltobuy)  
 currentfuel = currentfuel + fueltobuy  
 costfuelbought += costtobuy  
 **if** stopped\_at\_port == 0:  
 fuelstrategy = [fueltobuy]  
 purchase\_strategy = [costtobuy]  
 **else**:  
 fuelstrategy = fuelstrategy + [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
  
 **if** costfuelbought < cheapestsofar:  
 cheapestsofar = costfuelbought  
 cheapestroute = routenumber  
 self.cheapest\_distance = totaldistance  
 cheapestroutelist = route  
 self.costofcheapest = costfuelbought  
 self.cheapestlegs = legs  
 self.cheapest\_cost = int(cheapestsofar)  
 self.cheapest\_route = cheapestroutelist  
 self.fuelstrategy =fuelstrategy  
 self.purchase\_strategy = [int(x) **for** x **in** purchase\_strategy]  
 **if** routenumber < 24:##for routes with five legs  
 ##set up some worker variables used later  
 purchase\_strategy=[]  
 fuelstrategy=[]  
 currentfuel = 0  
 fueltobuy = 0  
 legs = self.candidates\_legs[routenumber] ##generate the five legs list  
 subroutes = self.inrange[routenumber] # a list of subroutes for each port stopped at en route  
 # print('routenumber', routenumber, 'route', route)  
 # print('subroutes for routenumber', routenumber, 'subroutes', subroutes)  
 airports = [Airport(route[0]), Airport(route[1]), Airport(route[2]), Airport(route[3]),  
 Airport(route[4]), Airport(route[5])]  
 # print('airport index 0 airportcode',Airport(route[0]).code)  
 costofthisroute = 9999999999999999999  
 fuelstrategyatthisport = []  
 totaldistance = legs[0] + legs[1] + legs[2] + legs[3] + legs[4]  
 # print('totaldistance', totaldistance)  
 accumulated\_distance = 0  
 distanceleft = totaldistance  
 fuelboughtatthisroute = 0  
 # print('entering subroutes')  
 distancetravelled = 0  
 costfuelbought = 0  
 costtobuy=0## todo  
  
 # as you visit each port see if you should buy fuel and how much  
 # print('about to enter for ateachport in range(5) : line 326')  
 **for** stopped\_at\_port **in** range( 5): # load in a given ports subroute nb ateachport is the index of port in  
 # mainroute and first port ahead of it is ateachport+1  
 subroute = subroutes[stopped\_at\_port] ##load subroute in range for a stopped at port  
 **if** stopped\_at\_port == 0:##at initial port  
 fuelused = 0  
 distanceleft = totaldistance  
  
 **else**:  
 fuelused = legs[stopped\_at\_port - 1]##fuel used is fuel travelling from last port  
 distancetravelled += legs[stopped\_at\_port - 1] ## distanced travelled  
 currentfuel -= fuelused  
 distanceleft = totaldistance - distancetravelled  
 # print('portindex',ateachport,'subroute',subroute)  
 fuelboughtatthisport = 0  
 ##initialize so variables for each iteration  
 startleg = stopped\_at\_port ##first leg index on subroutes is the same as the index of stopped at port  
 indexofcheaper = stopped\_at\_port ##initially set cheapest port in the subroute to BE the stopped at port until find otherwise  
 cheapestrate = airports[  
 stopped\_at\_port].currencyeurorate ## initially set cheapest rate in the subroute to the stopped at ports  
  
 # print('about to enter for idx2,aheadports in enumerate(subroute): ')  
 **for** idx2, aheadports **in** enumerate(subroute):## see if there is a cheaper port ahead within range  
 # print('subroute',subroute)  
 # print('current stopped at airport index',stopped\_at\_port,'go in through idx loop line395 idx2 = ',idx2 )  
 distancetocheaper = 0##will buy fuel to get to cheaper port  
 aheadportsindex = stopped\_at\_port + 1 + idx2 ##get the proper index of a port in subroute  
 # print('ahead airport index ',aheadportsindex,'aheadairport name',aheadports)  
 #print('aheadports indexs', aheadportsindex)  
 **if** (airports[aheadportsindex].currencyeurorate <= cheapestrate): ##find next cheapest airport as will buy enough fuel to get to it  
 indexofcheaper = aheadportsindex # reset to index of cheapest port  
 cheapestrate = airports[aheadportsindex].currencyeurorate  
 **break** # print('at airport',stopped\_at\_port,'but cheaper airport at index',aheadportsindex)  
  
 **else**:  
 **pass** # reset to index of cheapest port  
 # legidforcheapest = stopped\_at\_port + 1 + idx2  
 # print('in for x in range ..ateach to indexof cheaper')  
 **if** stopped\_at\_port == indexofcheaper:##current aiport is cheapest in range so fill up to get home or max up  
 **if** distanceleft <= self.craft.range:  
 fueltobuy = distanceleft - currentfuel  
 costtobuy = (airports[stopped\_at\_port].currencyeurorate) \* (fueltobuy)  
 fueltobuy = distanceleft - currentfuel  
 currentfuel = currentfuel + fueltobuy  
 costfuelbought += costtobuy  
 **if** stopped\_at\_port == 0:  
 fuelstrategy = [fueltobuy]  
 purchase\_strategy = [costtobuy]  
 **else**:  
 fuelstrategy = fuelstrategy + [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
  
 **else**: ##distance left greater than fuel tank range yet current airport is cheapest in range so max up to range  
 fueltobuy = (self.craft.range - currentfuel) ##top up to max  
 costtobuy = (airports[stopped\_at\_port].currencyeurorate) \* (fueltobuy)  
 currentfuel = currentfuel + fueltobuy  
 **if** stopped\_at\_port == 0:  
 fuelstrategy = [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
 **else**:  
 fuelstrategy = fuelstrategy + [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
 **else**:## there is a cheaper airport in reach do fuel so compute distance to it and top up to reach it  
 **for** x **in** range(stopped\_at\_port,indexofcheaper+1):  
 distancetocheaper += legs[x]  
 fueltobuy = distancetocheaper -currentfuel  
 costtobuy = (airports[stopped\_at\_port].currencyeurorate) \* (fueltobuy)  
 currentfuel = currentfuel + fueltobuy  
 costfuelbought += costtobuy  
 **if** stopped\_at\_port == 0:  
 fuelstrategy = [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
 **else**:  
 fuelstrategy = fuelstrategy + [fueltobuy]  
 purchase\_strategy = purchase\_strategy + [costtobuy]  
 **if** costfuelbought < cheapestsofar:  
 cheapestsofar = costfuelbought  
 cheapestroute = routenumber  
 self.cheapest\_distance = totaldistance  
 cheapestroutelist = route  
 self.costofcheapest = costfuelbought  
 self.cheapestlegs = legs  
 self.cheapest\_cost = int(cheapestsofar)  
 self.cheapest\_route = cheapestroutelist  
 self.fuelstrategy = fuelstrategy  
 self.purchase\_strategy = [int(x) **for** x **in** purchase\_strategy]  
  
**def main**():  
 #uncomment to check different starting permutations of same route to check consistency of algorithm  
 #x = Route('BUS', 'DUB', 'AOC', 'LHR', 'TUF', '747')  
 #y = Route('BUS', 'TUF', 'AOC', 'LHR', 'DUB', '747')  
 z = Route('LHR', 'DUB', 'BUS', 'AOC', 'TUF', '747')  
 print('cost',z.cheapest\_cost)  
 print('routedetails',z.cheapest\_route)  
 print('route distance',z.cheapest\_distance)  
 print('fuel to buy strategy',z.fuelstrategy)  
 print('cheapest legs',z.cheapestlegs)  
 print('purchase strategy',z.purchase\_strategy)  
 print('self.costofcheapest',z.costofcheapest)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()  
  
  
 # LDE Tarbes Fra  
 main()  
  
# TLS Toulouse France  
# TUF Tours France  
# BUS Batumi Georgia  
# KUT Kutaisi Georgia  
# TBS Tbilisi Georgia  
# AOC Altenburg Germany  
# SXF Berlin Germany  
# TXL Berlin Germany  
# BWE