```
'beginning of python module: ADI tags defined; this is a library 4 ADI pack & unpack'
#ideas for both programs                                          4/04/2024 JRJ
#   - don't output lower than XX record; higher are newer
#   - end date on QSOs is sometimes left out, but not time_off or length
#   - /.local/share/Trash/files/ deleted files area?

import asyncore,cgi,math,os,shelve,socket,sys,time    #subroutines for other 2 pgms
import pyhamtools as hBx #from pyhamtools missing locator, frequency, qsl. why???
import tkinter as tKw
#import dataDriver2a as dDx      #optional
VSlog,MRdat,Cre = hBx.callinfo.logging,hBx.callinfo.datetime,hBx.callinfo.re
#shortcuts for testing, last reg expr
vs,selL,selL2,lYr = '20a',0,3,True                        #2024 is 1! q=49 is
significant
line,lastline,lend,nwLn,sep = '','','\r\n','\n',','     #2nd # above: best 0,5  good
short 3,1  echo 2

#used only in the 1st program, conversion
cols = [0,3,4,8,9,10,12,13,14,31,33,35,40,41]   #pack sequence -1, see processLn1 more
info
#zneS =
{'PST':8,'MST':7,'CST':6,'EST':5,'AST':4,'PDT':7,'MDT':6,'CDT':5,'EDT':4,'ADT':3,'UTC':
#dictionary form

#used only in the 2nd program, reversion
#tgEcpts =
['QSO_len','QSO_date_off','qso_date_off','CNTY','cnty','cont','cqz','email','ituz','app
#app_ needs wildcard *?
tgStr = [
'QSO_no','','freq','BAND','mode','TX_pwr','','','time_on','call','RST_sent','RST_rcvd',
'QSL_sent_via','QSL_sent','QSL_rcvd','','LoTW_QSL_sent','freq_RX','BAND_RX','contest_id
'eQSL_QSL_sent','rig','RX_pwr','antenna','MY_tzn','','operator','PFX','COUNTRY','DXCC',
```

```
'address','QSO_random','lat','lon','notes','','HAMlog_sent', ]  #put data types here 2? both sparse lookup lists, esp. the 2nd

upStr = ['state','MY_state','cont','call','operator','station_callsign','PFX','QSL_sent','QSL_r #upper case
capSt = ['QTH','name','MY_name','COUNTRY','MY_COUNTRY','rig','MY_rig','mode','gridsquare','QSL_ #or title?

ctFlgNm = [  'FwD',    'PwMC',   'FSp',    'PwD',    'PnC',    'FAT',    'StdC',  'Std',   'nVb',   'AlT',   'Rev',   'diag' ] #linked to arrays below
pgmCtFlgs = ['ttfttfx','FTTFTFx','100010x','ftfttfx','FFTFTFx','111111x','111 1 ','111011','11xx1x','111111','010111','t000t0' ] #6 in @
#above 2 used for commands, below for standard case; make a dictionary?
tgStrL = [ 'qso_no','','','band','','tx_pwr','','','','','rst_sent','rst_rcvd','','qth','','','',
'qsl_sent_via','qsl_sent','qsl_rcvd','','lotw_qsl_sent','freq_rx','band_rx','','','qrzc
'','rx_pwr','','my_tzn','','','pfx','country','dxcc','','sat_name','sfi','qso_date','',
'qso_random','','','','','hamlog_sent', ]                  #change to the standard form, here and in miniMy

myDat = [ 'station_callsign','MY_CNTY','MY_lat','MY_lon','MY_DXCC','MY_gridsquare','MY_cq_zone','
]
myDatL = ['','my_cnty','my_lat','my_lon','my_dxcc','my_gridsquare','my_cq_zone','my_itu_zone','m # +enumerated 2 upStr
#doySt= [0,31,59,90,120,151,181,212,243,273,304,334,365]     #no leap year, m-1
dyPm = [31,28,31,30, 31, 30, 31, 31, 30, 31, 30, 31]        #feb changes Ly; both not used

```

```
42 def ele_pak(t,v,d,p,q):     #flag on end for data type                    t,v,d,p,q
   local
43     'creates an ADI datagram from user provided data. 2 control flags. 5 parms'
44     l = len(v)              #V value, t a tag string                      l local
   0.0 p1
45     s1 = '<'+t+':'+str(l)   #what type is v?
46     if q: s1 = s1+':'+d     #added the data type here
47     s1 = s1+'>'+v           #tested, works!                               s1 local
48     if p: s1 = s1+' '       #in python3 print is a function!
49     if (v=='' or l==0 or t==''):  return ''
50     else:  return s1
51
52 def ele_unpak(s,v):         #format 't:l(:d)>v'
   1.0 replaced by xxx1
53     'takes apart a TLDV ADI format datagram. 2 parms, tuple back'
54     t,l,d,f ='err',0,'e',False  #set defaults
55 #    print (s,v)               # change > to a :
56     if len(s) == 0:  return (f,t,l,d,v)  #check length first
57     tdat = s.split(':')         #3 cols, 2 colons etc.
58 # remove trailing spaces in v if any? done outside, see __
59     if len(tdat)==3:    d = tdat[2]
60     elif len(tdat)==2:  d = 'n/a'
61     elif len(tdat)==1:  return (f,t,l,d,v)
62     else:   return (f,t,l,d,v)  #an error too
63     if tdat[1]=='':     return (f,t,l,d,v)
64     t,l = tdat[0],int(tdat[1])  #failing here #1 null
65     if len(str(v))==l:  f = True
66     else:  print (1.0,'  data: inconsistant L- tag',s)
67     return (f,t,l,d,v)          #tuple output, v pass thru
68
69 def wrt_1line(k1,g1):           #there is no writeline only write...      g,k local
70     'writes out a line of text to the screen and a logfile. 2 parms'
```

```
71      if k1>0:  print (k1,line)   #line is a main global var
   2.0 p2
72      g1.write(lend)
73      g1.write(str(abs(k1))+' ')
74      g1.write(line)              #diff from 1row, need parameter line input
75      return -1                   #appears not to be used
76
77  def ele_unpak1(s,v):                                                #t,l,d local
78      'takes apart a TLDV datagram, the latest version. 2 parms, tuple back'
79      t,l,d,f = s,0,'',False      #3w +: +#
80      if len(s) <5:  return(True,t,l,d,v)
81      if s.find(':') > -1: pass   #ok if 1 or 2
   3.0 p2
82      else:  return(True,t,l,d,v)
83      sp = s.split(':')           #well controlled? picks up eor error
84      if len(sp)==2:              #no d
85          t,l=sp[0],sp[1]
86      elif len(sp)==3:            #all of them
87          t,l,d=sp[0],sp[1],sp[2]
88      else:  f=True               #error flag
89      if l=='':  l=0              #numeric
90      else:  l = int(str(l))      #is v always string?
91      if (len(v)==l) or (len(v.strip())==l):  pass    #ARRL uses // for a comment
92      else:  print (3,'  data inconsistant L- tag',s)
93      return (f,t,l,d,v)
94
95  def wrt_1linA(s,k,g):           #goal: s might be a list?
96      'writes out a line of text, or opt string to the screen or a logfile. 3 parms'
97      if (s=='') or (s==None):    #invert for true
98          wrt_1line(k,g)
99      else:                       #not null & not none
   4.0 p2
```

```
100         if k>0:  print(k,s)
101         g.write(lend)
102         g.write(str(abs(k))+' ')
103         g.write(str(s))          #g logging file usually
104 #         g.write(lend)
105     return -1
106 
107 def opn_outp(k,k1):              # originally, samOutput for sample output
    5.0 p1
108     'opens the output file for conversion to ADI, 1st program'
109     ks,ks1 = str(k),str(k1)     #creates file if; append mode +fiNo/                k
    local
110     return open('hamOutput'+ks+ks1+'.adi','a')
111 
112 def fmt_out(ls,y,f):
113     'output a formatted-spaced file, with sep character. 3 parms'
114     ll = len(ls)
115     ct = ll
116     while ct > 0:
117         idx = ll-ct              #start at 0
    6.0 p2
118         if f and idx==12:
119             y.write('QSO_len')  #what about a legit null?
120         else:
121             y.write(str(ls[idx]))
122         y.write(sep)
123         ct += -1                 #above, seperator
124     y.write(nwLn)
125     return ll
126 
127 def det_newln(s):
128     'looks for (detects) the OS newline in a text string'
```

```
129     se = s[-1]
130     return (str(se)==nwLn)  #with \n
    7.0 p2
131
132 def det_newlnA(s):
133     'looks for the OS newline plus, in a text string'
134     se = s[-2:]             #ok, w/ shortcut?                                   s[] local
135     return (str(se)==lend)  #w \r too
136
137 def rem_newln(s1):
138     'removes the newline character from a line of read-in text'
139     if det_newln(s1):
140 #        print( 8,'  newline slash n present' )
141         return s1[0:-1]     #seems to work
    8.0 p2
142     else:   return s1
143
144 def lod_tagS():             #binary created more issues
145     'loads the tags in a read line for ADI lookup'
146     line1 = lastline.split(',')
147     line2 = line.split(',') #global in caller?
148     del line1[-1]           #works!
149     del line2[-1]           #removes \n appended by OS
    9.0 p1
150     return line1+line2      #added more commas to data
151
152 def sercLst(ch,ls):         #a primative for a few subr's
153     if ch in ls:    idx = ls.index(ch)
154     else:           idx = -len(ls)
155     return idx              #returns neg if not found
156
```

```
157 #increases diversity in Case; allow easy classification by eye & standardization; tag
    portions!
158 ltTags =
    ['qso','qsl','iota','lotw','my','qth','sig','stx','sat','rst','intl','rx','tx','qrz','c
159 'country','arrl','itu','nr','app','ham','eoh','eor',]
160 def tradL4B(s):             #capitalize, subr; add 1 more
    10.0 used here
161     'with the array pair, trades the l/c for u/c string. used for testing'
162     bgTags =
    ['QSO','QSL','IoTA','LoTW','MY','QTH','SIG','STX','SAT','RST','INTL','RX','TX','QRZ','I
163 'COUNTRY','ARRL','ITU','NR','APP','HAM','EOH','EOR',]   #used with outboard testing of
    lists; +SoTA PoTA BoTA
164     if s in ltTags:
165         id = ltTags.index(s)
166         s1 = bgTags[id]     #see lists above, portions of tags
167     else:  return ''
168     return s1
169
170 def multIdx(c,s):           #2nd pgm decoding function
171     'for all c in string s, report positions in the list. tuple back'
172     ls = [None]             #init a list
173 #    print(11,c,s)
174     p = s.find(c)           #get 1st one, beginning
    11.0 here
175     ct,ls = 1,[p]
176     while (p>-1):           #last one found
177         p = s.find(c,p+1)   #do a next one
178         if p > -1:          #found one
179             ls += [p]       #add to list
180             ct += 1
181         else:  pass         #remove Nones
182     return (ls,ct)
```

```
183
184 def lod_dtypes():           #same issues as above
    12.0 p1
185     'in a matching line, gets the ADI datatypes'
186     line1 = line.split(',')
187     del line1[-1]           #delete stray endings on both
188     return line1
189
190 def roll_ck(h,ut,dh):           #more general than it needs to be         h local
191     'given a hour and timezone calculates if the day has advanced. 3 parms'
192     if ut:      return (h>=24)  #false means ok, just checks hours
193     pass     #h, dh are integers, ut false now
194     if dh==5:   return (h>=19)  #true means rolled
195     elif dh==4: return (h>=20)  #can do this in hundreds
    13.0 p1
196     else:    return (h+dh>=24)  #a non Eastern zone?
197
198 #def adv_tm(s,f1,f2):        #for EST, only 2 legit cases                  f1,s local
199 #    'original time shift using flags'
200 #    f3=(s.find(':')==-1)    #test for no colons
201 #    if (len(s) == 4):           pass    #remove :
202 #    elif (len(s)==3 and f3):    pass
203 #    else:
204 #        print (13,'  flag in advance time- incorrect format')
205 #        return -100
206 #    t = int(s)                                                            t,d local
207 #    d = 400             #no rollover here, see roll warn
208 #    if f1: d = 500      #adds 4 or 5 hours
209 #    if f2:              #alternate zone AST replaces, up USA
210 #        d = 300
211 #        if f1: d = 400
212 #    return t+d          #not rolling sum preserves going back?
```

```
213
214 def adv_tm1(s,z):       #different flag usage then first
215     'adds a time shift to a four digit time- hhmm. 2 parms, tuple back'
216 #    print (212,'input: ',s,z)
217     f3=(s.find(':')==-1)    #test for no colons                          f2, f3 local
218     if (len(s) == 4):           pass    #ditto
    15.0 p1
219     elif (len(s)==3 and f3):    pass
220     else:
221         print (15,'  flag in advance time1- incorrect format')
222         return (-100,False)
223     t,d = int(s),100*z
224     md = 2400          #starts w/ given zone
225     if (t+d >= md):   return (t+d-md,True)
226     return (t+d,False)  #assumes only up, like the other
227
228 def tme_splt(s):  #needs leading 0's- lower members
229     'splits a time string into its component parts from text, tuple back'
230     f1 = ( s.find(':') > -1 )           #sets, if found                  f1 local
231     hs,ms,ss='0','0','0'
232     if ( (len(s)==8) and f1):           #hh:mm:ss 8 only #'s'
233         hs,ms,ss=s[0:2],s[3:5],s[6:]
234     elif ( (len(s)==7) and f1):         #h:mm:ss  7
235         hs,ms,ss=s[0:1],s[2:4],s[5:]
236     elif ( (len(s)==6) and (not f1)):   #hhmmss   6 or h:mmss? last unlikely
237         hs,ms,ss=s[0:2],s[2:4],s[4:]
238     elif ( (len(s)==5) and (not f1)):   #hmmss    5 likely error, no colon(s)
239         hs,ms,ss=s[0],s[1:3],s[3:]
240     elif ( (len(s)==5) and f1):         #hh:mm also 5, has a :
241         hs,ms=s[0:2],s[3:]
242     elif len(s)==4:                     #hhmm     4
    16.0 p1
```

```
243         hs,ms=s[0:2],s[2:]
244     elif len(s)==3:                     #hmm      3
245         hs,ms=s[0:1],s[1:]
246     else:
247         print(16,'  decoding- unable to split time string')
248         ss='66'             #output defaults then
249     return (int(hs),int(ms),int(ss))
250 
251 def solv_tof(sth,stm,lq):
252     'computes time_off, adds small lengths of QSO times. 3 parms, hr mn len, tuple
    back'
253 #    print (sth,stm,lq)     #echo check #s in
254     df,eh = False,str(sth)  #input time fragments; df end date flag
255     m = stm+lq              #lq max ~120 minutes                          m local
256     if m<60:                #done :mm
257         em = str(m)         #string out
258         if m<10:  em='0'+em     #fix
259     else:                   # >= 60, roll hour
260         c=m//60             #fix for larger numbers, addd c*40            c local
261         if c>1:  print (17,'  flag in solvTof- larger sum than expected',c)
262         m += 40             #hr left, min right w/ h:mm, how many 40's
263         b = str(m)          #convert to string, width?
264         h = int(b[0])       #hour 1 digit only                            b, h local
265         em = b[1:]          #recast
266 #--- min, above; hours below ------
267         esm = h+sth         #carry plus :hh 2w?
268         if esm < 24:        #check roll hrs
    17.0 p1
269             eh = str(esm)   #done, df same
270         else:               # >= 24, new day! ------
271             esm += 76       #hour sum /w, 100s comp, how many 76's
272             f1 = str(esm)   #convert to string                            f1 local
```

```
273             eh = f1[1:]     # of dhh
274             df = True       #leftmost flag, but 1
275     return (eh+em,df)       #time sum at end; date roll would be to _off
276
277 def getQSOlen(f,s):     #comes in as strings, process numeric. as hhmm, Not hhmmss
278     'to get the QSO length back from end time. 2 parms, finish & start. tuple back'
279     ed,bg = tme_splt(f),tme_splt(s)     #tuple back
280     print (18,'  end first: ',ed,bg)   #echo check
281     df,borw = False,True    #flags, forcing borrow
282     len1 = 60+ ed[1]-bg[1]  #always? >0, dM+60;             issue, len is a def
    function!
283     if len1 >= 60:          #over 60, didn't borrow!
284         len1 += -60         #length in minutes
285         borw = False        #100's complement, sorta
286     len2 = ed[0]-bg[0]      #hours, dH hours
287     if borw:  len2 += -1
288     if len2 <0:             #check for day roll
    18.0 p2
289         len2 += 24
290         df = True
291     min = len1 +60*len2     #desired
292     if len1 <10:     ot = '0'+str(len1)
293     else:       ot = str(len1)
294     return (str(len2)+ot,str(min),df)
295
296 def trans_xlt(v,p,g,a,b,c): #translation, Value tyPe taG... maybe here?
297     'for experimentation and testing only. tuple back'
298     #if true/false convert to Y N. convert case, maybe not yet?
299     if v == 'TRUE':
300         v = "Y"         #works ok now
    19.0 here
301         return (v,p,g)
```

```
302     if v == 'FALSE':    #most of these are e(numerated) new p
303         v = "N"         #boolean comes from Sheets
304         return (v,p,g)
305     if a:               #add tag time_off; main global
306         if g=='QSO_len':
307             if v == "":  print(19,'  missing QSO length- cant convert')
308             else:
309                 stt=tme_splt(b)     #echo check stt, Tm4Col main global, row?
310                 oft=solv_tof(stt[0],stt[1],int(v))
311 #                print(stt,v,oft)            #look for null somewhere
312                 v=oft[0]            #check day flag- oft[1]
313                 if oft[1]:  print (19.1,'  flag in trans_xlt- day rolled over')
314                 p,g='t','time_off'          #add #8 to val, replace more input
315     # pair are exclusive
316     if ((g=='QSO_date') and c):
317         nwdat = roll_1day(v)                #process date roll, zone change
318         print(19.2,'  in date subs.
    old1',v,'new1',nwdat[0],'year',nwdat[1],'month',nwdat[2])  # +2 flags
319         v = nwdat[0]                        #wrn #1 flag or rw[1]?
320     return (v,p,g)                          #Value, tyPe, taG; p already t
321
322 def LpYrs(s):               #no century year logic, but 2000 was LY
323     'checks if a recent year is a leap year'
324     if len(s)==4: pass
325     else:  return False
326     j = int(s)              #check 1984, 2000, 2016 etc
    20.0 here
327     return ((j % 4) == 0)
328
329 def roll_1day(s):               #f1 removed, up in the USA
330     '30 days hath Sept, April, June & November, all the rest have 31. Except... tuple
    back'
```

```
331     g1,h=False,False            #clear year/month rolled flags            g1,h local
332     if len(s)==8:   pass        #must be eight
333     else:           return ('',g1,h)
334     yr,md=s[0:4],s[4:]          #strip off year, save n/c                 s[] local
335     dLpF = LpYrs(yr)                #boolean Flag back
336     m,d=int(md[0:2]),int(md[2:])    #string to # actual; error, invalid in-day no leap
    year
337     nyr = int(yr)+1             #if needed?
338 #---- dom section starts, use Dec as year roll check, new day?           d local
339     m1 = m                      #save input month
340     m2 = dyPm[m-1]              #days pm
341 #    print(m1,m2)                                                        m m1 m2
    local
342     if (m1==2 and dLpF):  m2=29 #override col data
343 #    print(s,m,d,nyr,dLpF)       #echo check
344     if d+1 > m2:                #pre-rolled month 2
345         m1,d = m+1,0            #convert dom to string, ns
    21.0 p1
346         h = True                #month rolled
347         g1 = (m1>12)            #if true 1/1 of next year else...
348         if g1:                  #rolled year too
349             m1=1
350             yr = str(nyr)       #next year, print note?
351     else:  pass                 #month didnt roll
352     d += 1                      #ok to roll just a day
353     ds,ms = str(d),str(m1)      #d and m2 convert 2str but pad
354     if len(ds)==1:  ds = "0"+ds
355     if len(ms)==1:  ms = "0"+ms     #pad singles
356 #---- doy section, lower probability of year roll <0.5% but...
357 #    dy = d+doySt[m-1]+1    #convert to next day of year
358     md = ms+ds
359     return (yr+md,g1,h)         #year rolled, false no issue w/ year
```

```
360
361 def revAset(iL1,vL2,g):          #1st is a list of int's to be dragged along
362     'reverses a set assuming a string list V with index included. tuple back'
363     if g:
364         ct = len(vL2)         #to create iL1 if can be int & in-order
365         cSv = ct
366         while ct > 0:
367             ix = cSv-ct
368             iL1[ix]=ix           #over writes the input
    22.0 here
369             ct += -1
370     la,lb = len(iL1),len(vL2)   #lower case L
371     if la == lb:  pass  #good
372     else:
373         print (22,'  entering data: ',iL1,vL2)
374         return (vL2,iL1,True)   #an input error
375     idx = la-1                  #start 1 less than size
376     s21,iL2,vL1 = ['m']*la,['n']*la,['o']*la    # init the other arrays s21...
377 #    print (s21,s21s,iL2,vL1)
378     while idx >=0:
379         tmp = int(iL1[idx])
380         tmp1 = str(tmp)
381 #         print (tmp1,vL2[idx])
382         s21[idx] = vL2[idx]+ '|'+ tmp1
383 #         print ('in loop1',s21)
384         idx += -1
385 #    print (22.1,'end 1st loop',s21)
386     s21.sort()                  #sorts in place
387     idx = lb-1                  #restart
388     while idx >=0:
389         pr = s21[idx].split('|')   #left is 0, right is 1
390         iL2[idx],vL1[idx] = pr[0],int(pr[1])
```

```
391         idx += -1
392     return (iL2,vL1,False)
393 
394 def buf_mgr(s,b,lef,adv):
395     'manages a 1D queue with pop, push, R L, advance, etc. Null issue! 4 parms.
    returns new buffer'
396     sp = str(s) #b is an incoming string
397     if adv:
398         if sp == '':    #pop, from left- easier
399             if lef:
400                 ps = b.find(sep)
401                 if ps > -1:     #discard sep
    23.0 p2
402                     sp = b[0:ps]
403                     b = b[ps+1:]
404                 else:    pass   #null
405                 print (400,'  pop L',sp,'new b',b)
406                 return (b,sp)   #need tuple; no pop R needed if push L,R
407             else:
408 #                print (410,'  advance only- no data')
409                 if len(b)==0:    pass
410                 else:    b = b+sep
411         else:                           #push
412             if lef:
413 #                wrt_1linA('  push L- lef True',-407,gg)
414                 b = sp+sep+b        #add on left
415             else:
416 #                print(418,'  push R- lef False')
417                 if b == '':
418                     b = sp          #new
419                 else:
420                     b = b+sep+sp    #push right?
```

```
421     else:
422         if sp == '':                #null
423             if lef:
424 #               print (426,'  cleared buffer, was:',b)
425                 return ''
426             else:  print(421,'  read buffer, is:',b)
427         else:                       #sp exists
428             if lef:
429 #               print(431,'  add left',sp)
430                 b = sp+b
431             else:
432 #               print(434,'  add right-default',sp)
433                 b = b+sp            #default
434     return b
435
436 def fnd_tg(s,ln):       #find tag subr; pos 1 < than tag, see #1 version b
437     'copied here- but used as a local subroutine. see improvement below. 2 parms,
    tuple back'
438     if s=='':  return (False,1,s)
439     s = '<'+s           #not found -1; needs < for tag too
    24.0 p2
440     p2 = ln.find(s)     #the read line global
441     f1 = (p2 > -1)      #ignores L...  ARRL & QRZ have different formats
442     return (f1,p2+1,s)  #tuple back, p2' of tag start!
443
444 def fnd_tg1(s,ln):      #find ADI tag; pos 1 < than tag
445     'improvement on find for ADI tags, which uses the colon end too. tuple back'
446     if s=='':  return (False,1,s)
447     s = '<'+s+':'       #wrap param for tag2
448     p2 = ln.find(s)     #the read line in parameter, not found -1
449     f1 = (p2 > -1)      #ARRL & QRZ have different formats
450     #issue: tags with a similar suffix, but a different ending!
```

```
451     return (f1,p2+1,s)  #tuple back, s', p2' tag start!
    25.0 here?
452
453 def crt_dic(ls):    #based on a 2nd program segment
454     'creates a simple dictionary for more than 1 value'
455     j0 = len(ls)
456     kvl = [None]*j0    #next step join two lists- real dict
457     while j0 > 0:                #last 2 are line arrays
458         k0 = j0-1                #loads 0 last
459         kvl[k0] = (ls[k0],k0)  #temp tuple
    26.0 here
460         j0 += -1
461     return dict(kvl)
462
463 def crt_dic1(ky,vl):    #based on a 2nd program segment
464     'creates a proper key value pair dictionary'
465     j0 = len(ky)
466     if len(vl) < j0:    return {}
467     dl = [None]*j0      #next join two lists- real dict
468     while j0 > 0:                    #last 2 are line arrays
469         k0 = j0-1                    #loads 0 last
470         dl[k0] = (ky[k0],vl[k0])    #temp tuple
    27.0 p2
471         j0 += -1
472     return dict(dl)
473 cmds = crt_dic1(ctFlgNm,pgmCtFlgs)
474
475 def LnkLstTst(ls1,ls2,nully,f):
476     'tests 2 linked lists 4 standard details & creates the bigger dictionary. 4 parms,
    tuple back (7)'
477     f2,f3,hi = False,False,0    #NOT, sparse flags
478     dname = 'no match'          #default
```

```
479     f1 = ( len(ls1)==len(ls2) )
480     if not f1:  return (f1,f2,0,hi,f3,0,dname)
481     hi = len(ls1)                #they match, use list 1
482     ct,ct2,ct3 = hi,1,1          #or 5% of list length +/-
    28.0 here
483     while ct > 0:
484         id = hi-ct               #up from 0; see null types
485         if ls1[id] == nully:  ct2 += -1
486         else:  ct2 += 1
487         if ls2[id] == nully:  ct3 += -1
488         else:  ct3 += +1         #ie None,'', or 0
489         ct += -1
490     if ct2 > 0:  f2=True         #defaults to 1st list
491     if ct3 > 0:  f3=True
492     if ct2 >= ct3:
493         if f:  dname = crt_dic(ls1) #dname the created dictionary
494     else:       #print to test alignment of the data streams
495         if f:  dname = crt_dic(ls2)
496     return (f1,f2,ct2/hi,hi,f3,ct3/hi,dname) #match L, sparse-1, full+1, dict bigger1
497
498 def std_case(k,lin,g1):    #need a standard format so matching happens! use k?
499     'makes the ADI data insensitive to case- the key to the 2nd program. 3 parms'
500     ls,vl,tg,rs,tp = [],[],[],[],[]     #null lists for below; save found tag indexes
501
502     def adj_vals(v3,j0):                #for j1 loop
503         j2 = j0.split(':')              #remove innards of < :l(:t)>
504         fd,os1,os2 = True,'',''         #below are exclusive; add '>'?
505         if j2[0] in upStr:
506             os1 = '>'+v3.upper()
507             lin = lin.replace('>'+v3,os1,1)
508         elif j2[0] in capSt:
509             os2 = '>'+v3.title()
```

```
510             lin = lin.replace('>'+v3,os2,1)
511         else:                               #above, alter value
512             fd = False
513 #            print ( 29.4,'value not changed 4: ',tg[k3],'standardize',j2[0] )
514         print ( 29.4,j0,j2,'os1 & 2:',os1,'|',os2,'|' )
515         return fd
516 
517     lin = lin.lower()                     #get rid of the pesky comma w/ tilde or accent?
518     #wrt_1linA(lin,497,g1)                #first view of
519     ll = len(tgStrL)   #same as upper?
    29.0 p2
520     hi = ll
521     while ll > 0:       #loop thru l/c elements, lower --> std form
522         idx = hi-ll                             #starts at 0 in l/c elements
523         ti = fnd_tg1(tgStrL[idx],lin) #use sub2, is tagL there?
524         if ti[0]:                               #some low MISSing; no change needed!
525             ls += [idx]                         #save found index for use elsewhere?
526             lin = lin.replace(tgStrL[idx],tgStr[idx],1)  #replace w/ std
527         else:
528             if ti[2] != '':  pass               #or print found?
529 #                print (29,'not found: |',ti[2],'|','in standardize')  #sticks w/ old
    value?
530         ll += -1                                #next 1, idx goes ^
531     bg = multIdx('<',lin)     #get all the begin markers- <eor> last
532     ed = multIdx('>',lin)     #and the end's; the L's should match; all tuples
533     cn = multIdx(':',lin)     #info markers, some :'s may Bin values, or dtypes
534         #perform tests 1 & 2, if fails use #3 (req.); need 1 for all tags
535     if bg[1] == ed[1]:  l2=bg[1]          #can't split group at end of line! n
    pairs/line
536     else:                                   #check lists for L, : must be btw the
    first 2.
```

```
537         print ( 29.1,'  error in standardize- different list L/
    line?',bg[1],ed[1],cn[1] )
538         return str(bg[1])  #check 3 values; patterns: <:> <::> ok, >:< not, <eor>,
    others- not
539 #    print (517,bg,ed,cn)
540 
541     cnn = cn[1]+1                           #1 less : due to eor; values given below?
542     if cnn > l2:    rato = cnn/l2           #ratio of course
543     elif cnn < l2:  rato = -1               #unexpected error condx, trap? or 0
544     else:           rato = 1                #always 1 less colon, none in end of
    record
545     dtypcnn = (rato > 1.5)                  #a guess, w/ dtypes+; log this?
546     if dtypcnn:  print( k,'data types suspected- standardize: extra colons:',rato )
547 
548     k2,k3 = 0,0  #an integer, below strings
549     for k1 in myDatL:                   #iterate on lower myData, some are MISSing
550         if k1 != '':
551             lin = lin.replace(k1,myDat[k2],1)   #if null skip
552         k2 += 1     #goes up with k1 1 step
553                     #capitalize, base it on the tag; all lists like above thus
    changeable
554 
555     wrt_1linA(lin,29.2,g1)                  #2nd view of
556     hi2 = ed[1]     #counted > symbols, or values, should also equal beginning
557     while l2 > 1:           #start while, on end index
558         idx2 = hi2-l2       #values portion- last1 has no next (<). Maybe, use end of
    line?
559         idx21 = idx2 +1     #goes to the end, but stops @2
560         vl += [lin[(ed[0][idx2]+1):bg[0][idx21]]]   #get corresponding value for pairs
561         tg += [lin[(bg[0][idx2]+1):ed[0][idx2]]]    #and the tag (inside too)- save
    for later
562         tp += (tg[idx2],vl[idx2])
```

```
563         l2 += -1                                #tags determine action on values; use 'in'
    again?
564     print ( 29.3,'extracted t&v: ',tp )      #strip inside tg? +1 would be at end of
    string...
565     if len(tg)==0:  return str(l2)          #nothing found
566 
567     for j1 in tg:                           #loop thru found tags, should be 1 or more
568         bk = adj_vals(vl[k3],j1)            #new
569 #        j2 = j1.split(':')                  #remove innards of < :l(:t)>
570 #        fd,os1,os2 = True,'','''            #below are exclusive; add '>'?
571 #        if j2[0] in upStr:
572 #            os1 = '>'+vl[k3].upper()
573 #            lin = lin.replace('>'+vl[k3],os1,1)
574 #        elif j2[0] in capSt:
575 #            os2 = '>'+vl[k3].title()
576 #            lin = lin.replace('>'+vl[k3],os2,1)
577 #        else:                                #above, alter value
578 #            #os1,os2 = '',''
579 #            fd = False
580 #            print ( 29.4,'value not changed 4: ',tg[k3],'standardize',j2[0] )
581 #        print ( 29.4,j1,j2,'os1 & 2:',os1,'|',os2,'|' )
582 #        fd,lin = bk[0],bk[1]    #new
583         rs += [bk]                          #store each result, then?
584         k3 += 1                             #next
585     wrt_1linA(lin,29.5,g1)                  #3rd/last view of
586     print (29.6,rs,'values')                #boolean list
587     return lin   #use the case list & caps, for both: see csub library
588 
589 def rem_ARL (vl):   #splits off ARRL comment
590     'removes but saves the ARRL comment field part of V. tuple back'
591     p = vl.find('//')
```

```
592     f = (p>-1)      #found
    30.0 p2
593     if f:
594         v2 = vl.split('//')     #in half, are 3 possible?
595         return (f,v2[0],v2[1])  #both halves
596     return (f,vl,'none')
597 
598 def get_Bol (s):                #creates boolean list
599     'creates a boolean list given a prototype string of ch.'
600     ct,lo = -len(s),[]
601     while ct < 0:
602         c = s[ct]
603         vl = ( c=='T' or c=='t' or c=='1' )  #
    31.0 used below
604         lo += [vl]
605         ct += 1
606     print (31,'  loaded program control flags:',lo,'from:',s)
607     return (lo)
608 
609 cmNp1 = 6               #for next 2 subroutines
610 def rtnPflgs(pn):       #gets a control set, either pgm 1/2
611     'uses boolean subroutine (prev.) to pick a program control flag set'
612     if (pn > 2) or (pn < 1):  return ''
613     pn += -1                    #makes boolean, see new invert below
614     if pn:  idx = selL2+cmNp1   #pgm number 2
615     else:  idx = selL           #1st program but global here
    32.0 p2
616     print ( 32,'code:',ctFlgNm[idx],idx )
617     return get_Bol(pgmCtFlgs[idx])
618 
619 def ovrrFlgs(pgm,cmNo):     #use selLx to get the set (6/12 in array)
```

```
620     'for both conversion types, allows override of the default command set during
    either pgm run'
621     print ( 33, '  4 program:', pgm, 'the default Command Set is:', cmNo )
622     pgm += -1 #convert to boolean; first 6 goes with pgm 1
623     if pgm:  print ( 'the choices are:', ctFlgNm[cmNp1:], 'set2' )
624     else:    print ( 'the choices are:', ctFlgNm[0:cmNp1], 'set1' )
625     uin = input('do you want to override the current flags? y/n ')
626     if uin != 'y':  return (-1,'')      #do nothing
    33.0 in both 1 & 2
627     ui2 = input('enter a Mnemonic command string? ~3 ch. ')   #new Mnemonic another
    input
628     si = sercLst(ui2,ctFlgNm)
629     if si < 0:  return (-1,'')          #not found
630     else:
631         if pgm:
632             si += -cmNp1
633             selL2 = si      #save new index for future use, if any
634         else:  selL = si
635     return (si,ui2)         #new string/ default; tf equiv.
636
637 def Npadd(s):   #detects wh spc issue Wrt L
638     'tests for no padding in a string. Not this implies was padded'
639     w,wo = len(s),len(s.strip())
640     return (w==wo)              #for values
    34.0 p2
641
642 def invB(n):  #uses binary addition to complement
643     'inverts a simple binary number thru addition'
644     if n < 0: return 0  #an error
    35.0 here
645     n += 1              #invert via inc
646     if n > 1: return 0
```

```
647     return n
648 
649 #add subroutines based on import module values
650 def tmStr():
651     'print a nicely formatted time'
652     return time.asctime()
653 
654 def ppArgs():           #program arguments after invocation
655     'get the command line arguments'
656     return sys.argv
657 
658 def tmRel():            #from system epoch
659     'return a numeric time past epoch'
660     return time.time()
661 
662 def onWhat():           #in my case Linux
663     'what platform is python running on?'
664     return sys.platform
665 
666 def explr(s,v):
667     'used 4 testing to explore directories and classes for arg 2; 1st arg is supplied
    description'
668     print ( 40,s, '----------', nwLn, dir (v), nwLn, v.__doc__ )
669     print ('  end of:',v.__name__, nwLn)  #
    40.0 here
670     #print ( 40.1, help(v) )
671     return -1       #run pydoc.writedoc()
672 
673 def getHr(s):
674     'the missing subroutine- uses the military zone letter to get time delta'
675     Wzns = ['Z','A','B','C','D','E','F','G','H','I','K','L','M']    #hours behind UTC,
    EST zone R; 4 UTC conversion 1st program
```

```
    Ezns = ['Z','N','O','P','Q','R','S','T','U','V','W','X','Y']    #ISO 8601 zones 15d /long, this one -1*
    if len(s) != 1:  return 24
    if s in Wzns:
        h = Wzns.index(s)
    elif s in Ezns:
        r = Ezns.index(s)
#act. 42 (41.0)
        h = -r
    else:  h = 24
    return h

def Paddd(s):                                #shows how much too!
    'the Npadd function inverted, but now tells how much padding too!'
    l1,l2 = len(s.strip()),len(s)
    return (l2-l1)
# (42.0)

# add meta class statement? make sure methods have access to the globals herein?
class Hart:
    'test class: variable names, wrapper for functions from above; write example program to try?'
#    def ADI_ele_pak(self,p1,p2,p3,p4,p5): #5
#        return ele_pak(p1,p2,p3,p4,p5)
#    def ADI_unpak1(self,p1,p2): #2
#        return ele_unpak1(p1,p2)
#    def fIO_fmt_out(self,p1,p2,p3): #3
#        return fmt_out(p1,p2,p3)
#    def Dtme_adv_tm1(self,p1,p2): #2
#        return adv_tm1(p1,p2)
#    def Dtme_tme_splt(self,p1): #1
#        return tme_splt(p1)
```

```
#    def Dtme_solv_tof(self,p1,p2,p3): #3
#        return solv_tof(p1,p2,p3)
#    def Dtme_getQSOlen(self,p1,p2): #2
#        return getQSOlen(p1,p2)
#    def Dtme_roll_1day(self,p1): #1
#        return roll_1day(p1)
#    def Dtme_LpYrs(self,p1): #1
#        return LpYrs(p1)
#
#    def arry_buf_mgr(self,p1,p2,p3,p4): #4
#        return buf_mgr(p1,p2,p3,p4)
#    def arry_revAset(self,p1,p2,p3): #3
#        return revAset(p1,p2,p3)
#    def arry_crt_dic1(self,p1,p2): #2
#        return crt_dic1(p1,p2)
#    def arry_get_Bol(self,p1): #1
#        return get_Bol(p1)
#    def plogs_wrt_1line(self,p1,p2): #2
#        return wrt_1line(p1,p2)
#    def plogs_wrt_1linA(self,p1,p2,p3): #3
#        return wrt_1linA(p1,p2,p3)
#
#    def strHd_det_newln(self,p1): #1
#        return det_newln(p1)
#    def strHd_det_newlnA(self,p1): #1
#        return det_newlnA(p1)
#    def strHd_rem_newln(self,p1): #1
#        return rem_newln(p1)
#    def strHd_Npadd(self,p1): #1
#        return Npadd(p1)
#    def strHd2_fnd_tg1(self,p1,p2): #2
#        return fnd_tg1(p1,p2)
```

```
#    def strHd2_fnd_tg(self,p1,p2): #2
#        return fnd_tg(p1,p2)
#
    def strHd2_std_case(self,p1,p2,p3): #3 the heart of pgm 2?
        return std_case(p1,p2,p3)
    def strHd2_multIdx(self,p1,p2): #2
        return multIdx(p1,p2)
    def arry_lnkLstTest(self,p1,p2,p3,p4): #4
        return LnkLstTest(p1,p2,p3,p4)
    def misc_trans(self,p1,p2,p3,p4,p5,p6): #6
        return trans_xlt(p1,p2,p3,p4,p5,p6) #Value tyPe taG...

# subroutine testing:
print ( '----------begin conversion subroutine(s) testing:',tmRel(),'epoch time----------' )
if __name__ == '__main__':
    print ( solv_tof(13,3,8),solv_tof(22,44,16),solv_tof(23,55,45),solv_tof(17,45,24),solv_tof(2,4,) #but add more
    print ( LpYrs('1984'), LpYrs('1983'), LpYrs('2000'), LpYrs('2015'), LpYrs('2016') ) #passed
    print ( roll_1day('20190227'), roll_1day('20231231'), roll_1day('20221031'), roll_1day('20180625'), roll_1day('19570122'), roll_1day('20160327') )
    print ( adv_tm1('1411',4),adv_tm1('1945',5),adv_tm1('2310',4) )
    print ( adv_tm1('0331',5),adv_tm1('1411',4),adv_tm1('155',5),adv_tm1('1945',4),adv_tm1('1123',4) )
    print ( ele_unpak('qso:2:s',25), ele_unpak('time:4:t',3), ele_unpak('zone:2:e','de'), ele_unpak('call:5:s',222), ele_unpak('abcd',1), ele_unpak('date:4',1335), ele_unpak('rst_sent:3','yes')  )
    ab,xy = ['b','a','d','c','','','z'],[3,4,5,6,7,8,9]
    print ( revAset(xy,ab,False), revAset(xy,ab,True) )
```

```
    #h,samLs = open('libTestFil.txt','a'),[5,6,7,8,9,10,11,12,13,14]
    #fmt_out(samLs,h,False)
    #h.close()
    print ( buf_mgr('a','1234',True,False), buf_mgr('1','abcd',False,False), buf_mgr('25','56434828',True,True), buf_mgr('44','eeeeee',False,False), buf_mgr('','qewpuiower',False,True), buf_mgr('','abcd',True,False), buf_mgr('','r4t5y6u7',False,False), buf_mgr('','asde,493902,54jks,ghj',True,True) )
    print ( getQSOlen('2245','2130'), getQSOlen('1225','1220'), getQSOlen('2349','0002'), getQSOlen('0058','0043') )
    print ( tradL4B('rx'), tradL4B('qso'), tradL4B('qrz'), tradL4B('dxcc'), tradL4B('intl'), '|',tradL4B('abcd'),'|' )
    print ( multIdx(',','eroi,540,alsk,235'), multIdx('/','gjs/t567/wer=567'), multIdx('<','adf<456>gkfd=45,45<we'), multIdx('>','45>678<995>rty') )
    print ( tmStr(), 'paths:', sys.path, 'arguments:', ppArgs(), 'platform:', onWhat(), 'sep:', os.sep, 'line:', os.linesep)
    print ( 'The QUICK brown FOX?'.lower(), 'jumped over THE lazy [dogs]'.upper(), 'back! 0123456789'.capitalize() ) #"/><|}{\-=+!@#$%^&*()_+".capitalize() )  issue 1 or more
    print ( crt_dic( ['y','z','a','b','f','m'] ) )
    print ( LnkLstTst(tgStr,tgStrL,'',True ) ) #, nwLn, LnkLstTst(bgTags,ltTags,'',True ) )
    print ( LnkLstTst(myDat,myDatL,'',True ), nwLn, LnkLstTst(ctFlgNm,pgmCtFlgs,'',True), nwLn, cmds )
    print ( get_Bol('fftftf'), get_Bol('111111'), get_Bol('011010'), get_Bol('TFFFTF'), rtnPflgs(1), rtnPflgs(2) )
    print ( 'binary test:',invB(1),invB(0),invB(3),invB(-5) )
else:
    print ( nwLn, __name__, nwLn, dir(__name__), nwLn, __doc__, nwLn, 'end of?', nwLn ) # the current set?
    print ( dir (hBx), nwLn, '  end of:', hBx.__name__, nwLn ) # the imported pyHamtools
```

```
776 #    print ( 'callinfo',nwLn, dir (hBx.callinfo),nwLn,nwLn, hBx.callinfo.__doc__ )
    #below based on this start
777     print ( explr ('hamtools: Callinfo',hBx.Callinfo), explr('&
    LookupLib',hBx.LookupLib), explr('tK based cgi',tKw) )  #add Cre
778     print ( explr ('callinfo & logging:',VSlog), explr('... & date time',MRdat),
    explr('... callinfo',hBx.callinfo), explr('regular expressions:',Cre) )
779 print ('---------- end of library subroutine:',vs,'----------')
```