

```

1: #!/usr/local/bin/python
2: # -----
3: #   Program:  merra_ar_6hly_regional_analysis_web.py
4: #
5: #   Purpose:
6: #       Manipulates the pre-calculated IVT files (see merra_ivt_process.py)
7: #       to isolate atmospheric rivers (ARs) from within positive IVT anomalies,
8: #       using the pre-processed FFT file for the long-term mean and seasonal
9: #       cycle (see merra_ivt_6hly_fftcals.py).
10: #
11: #   Defined Functions:
12: #       days_in_month          - Provides the number of days in a month
13: #       retrieve_data          - Ingests global data and shifts grid
14: #       save_binary            - Saves array as NumPy binary file
15: #       skeleton_length        - Approximates feature length in km
16: #       uv2polarangle          - Converts u & v wind components to polar angle
17: #
18: #   History:
19: #       B. Mundhenk,      Sep 2014; Created for exploratory global analysis of ARs
20: #                       Mar 2015; Adjusted for regional use and relaxed thresholds
21: #                       Jul 2015; Modified for web posting (see below)
22: #
23: # -----
24: #
25: # ----- Web Code Modifications -----
26: #   The following functions have been removed from this version for clarity:
27: #       regional_daily_ivt_plot
28: #       regional_ivt_anom_plot
29: #       regional_ivt_ars_plot
30: #       regional_blob_ars_plot
31: #       plot_chars
32: #       plot_skeleton
33: #
34: #   Also, the following portions have been trimmed:
35: #       Flagged development areas, though the logic_flag and *_cnt variables remain
36: #       Option to save output as netCDF via PyNIO
37: #       Path variables and host names, though the variable names remain
38: #
39: #   * Superfluous code remnants may remain as a result of this trimming *
40: # -----
41: #
42: import csv
43: import datetime
44: import glob
45: import operator
46: import socket
47: import sys
48: import time
49: #
50: import numpy as np
51: import numpy.ma as ma
52: #
53: from scipy import ndimage
54: from skimage.measure import regionprops
55: from skimage.feature import peak_local_max
56: from skimage.segmentation import random_walker
57: from skimage.morphology import skeletonize
58: #
59: # ----- Setting Output and Control Variables -----
60: save_csv = False          # Optional saving of AR characteristics in CSV file
61: save_ar_ivt = False       # Optional saving of arrays of IVT anom. w/in ARs
62: #
63: wind_based_orient = True  # Correct the feature orientation based on LL winds?
64: #
65: data_str = 'MERRA'        # for file naming
66: method_str = 'FFT3_79-14S' # to retrieve correct seasonal cycle of IVT file
67: #
68: bounds = (67., 5., 252., 118.) # As: N, S, E, W
69: region_str = 'NPAC'        # for file naming
70: #
71: # Dates to process...
72: target_years = ['1979','1980','1981','1982','1983','1984','1985','1986','1987',\
73:                 '1988','1989','1990','1991','1992','1993','1994','1995','1996',\
74:                 '1997','1998','1999','2000','2001','2002','2003','2004','2005',\
75:                 '2006','2007','2008','2009','2010','2011','2012','2013','2014']
76: target_months = ['01','02','03','04','05','06','07','08','09','10','11','12']
77: target_hours = ['00','06','12','18']
78: #
79: #
80: # ----- Setting Path Variables -----
81: host = socket.gethostname()
82: if host == 'cluster_name':
83:     sys.exit()
84: elif host == 'mac_name':
85:     datapath = 'X'
86:     outpath = 'X'

```

```

87:     resultpath = 'X'
88:     ncpath = 'X'
89: elif host[0:4] == 'node':
90:     datapath = 'Y'
91:     outpath = 'Y'
92:     resultpath = 'Y'
93:     ncpath = 'Y'
94: else:
95:     print 'Error: Host not found'
96:     sys.exit()
97:
98: print 'Executing on',host,'\n'
99:
100:
101: # ----- Setting Variables for Feature Testing -----
102: min_IVT = 250.          # IVT anomaly threshold used to identify blobs
103: size_mask = 150.        # Minimum blob size allowed for testing
104: min_aspect = 1.6        # Minimum length/width ratio for retained feature
105: min_orientation = 0.95  # Minimum orientation (taken as np.abs())
106: min_length = 25.        # Shortest feature length (in pixels)
107: min_meanintensity = 305. # Lowest mean IVT anomaly intensity within a feature
108: min_eccentricity = .87   # Minimum eccentricity of a retained feature
109: lat_cutoff = 3.0        # Will restrict features centered about the equator
110: lat_forceorient = 20.0   # Threshold for increased testing (mostly TC-focused)
111: min_forceorient = .5     # Orientation threshold for near-EQ features
112: # -----
113:
114:
115: # ----- Function: days_in_month -----
116: def days_in_month(year,month):
117:     " Returns the number of days in the month provided "
118:
119:     if month in ['01','03','05','07','08','10','12']:
120:         return 31
121:     elif month == '02':
122:         if operator.mod(int(year),4) == 0:
123:             return 29
124:         else:
125:             return 28
126:     elif month in ['04','06','09','11']:
127:         return 30
128:
129:
130: # ----- Function: save_binary -----
131: def save_binary(data_array,yr,mo,dy,data_label):
132:     " Saves array of data in NumPy binary format "
133:
134:     outfile = data_str+'_'+data_label+'_'+yr+mo+dy+'.npy'
135:     np.save(outpath+outfile,data_array)
136:
137:
138: # ----- Function: skeleton_length -----
139: def skeleton_length(array):
140:     " Skeletonizes the feature to estimate feature length in km "
141:
142:     sk = skeletonize(array)
143:     ir,ic = np.nonzero(sk)
144:     numri = np.max(ir) - np.min(ir)
145:     numci = np.max(ic) - np.min(ic)
146:     if numri*1/2. > numci*2/3.:
147:         arr_flag = 'vert'
148:     else:
149:         arr_flag = 'horiz'
150:
151:     newr,newc = [],[]
152:     if arr_flag == 'horiz':
153:         duplicates = []
154:         for ind,col in enumerate(ic):
155:             if np.count_nonzero(np.where(ic == ic[ind])) > 1 and col not in duplicates:
156:                 duplicates.append(col)
157:                 newc.append(ic[ind])
158:                 newr.append(np.min(ir[np.where(ic == ic[ind])]))
159:             elif col in duplicates:
160:                 continue
161:             else:
162:                 newr.append(ir[ind])
163:                 newc.append(ic[ind])
164:         snewr, snewc = (list(el) for el in zip(*sorted(zip(newr, newc))))
165:         z = np.polyfit(snewc,snewr,3)
166:         p = np.polyld(z)
167:         x = np.arange(np.min(snewc),np.max(snewc)+1,1)
168:         y = p(x)
169:         y = np.where(y >= bounds[3], bounds[3]-0.1, y)
170:
171:     if arr_flag == 'vert':
172:         duplicates = []

```

```

173:         for ind,row in enumerate(ir):
174:             if np.count_nonzero(np.where(ir == ir[ind])) > 1 and row not in duplicates:
175:                 duplicates.append(row)
176:                 newr.append(ir[ind])
177:                 newc.append(np.min(ic[np.where(ir == ir[ind])]))
178:             elif row in duplicates:
179:                 continue
180:             else:
181:                 newr.append(ir[ind])
182:                 newc.append(ic[ind])
183:             snwr, snwc = (list(el) for el in zip(*sorted(zip(newr, newc))))
184:             z = np.polyfit(snwr,snwc,3)
185:             p = np.poly1d(z)
186:             y = np.arange(np.min(snwr),np.max(snwr)+1,1)
187:             y = np.where(y >= bounds[3], bounds[3]-0.1, y)
188:             x = p(y)
189:
190:         y_vals = np.copy(y)
191:         y_pixel_km = (1./2)*(111.2) # Approx. pixel extent in km
192:         piecewise_length = []
193:
194:         for i in xrange(0,len(x)-1,1):
195:             # Visiting Pythagoras to estimate length of skeleton in km
196:             x_pixel_km = (2./3)*(np.pi/180.0)*6378.137*abs(np.cos((np.pi/180.0)*sub_lats[y_vals[i]]))
197:             len_km = np.sqrt((abs(y_vals[i+1]-y_vals[i])*x_pixel_km)**2+(abs(x[i+1]-x[i])*y_pixel_km)**2)
198:             piecewise_length.append(len_km)
199:
200:         return x,y,int(np.sum(piecewise_length))
201:
202:
203: # ----- Function: retrieve_data -----
204: def retrieve_data(date_str,hr_ind):
205:     " Ingests and transforms array of 6-hourly IVT"
206:
207:     raw_array = np.load(outpath+data_str+'_IVT_6hly_'+date_str+'.npy')
208:
209:     # Transforming the array's grid from -180 > 180 to 30 > 390
210:     data_array = np.empty([361, 540])
211:     data_array[:,0:225] = raw_array[hr_ind,:,315:541]
212:     data_array[:,225:541] = raw_array[hr_ind,:,0:315]
213:
214:     return data_array
215:
216:
217: # ----- Function: uv2polarangle -----
218: # Converts meteorological u, v wind components to a polar angle.
219: # WARNING: Not in METEOROLOGICAL conventions
220: # Increases anticlockwise from the (+) x-axis
221: # 0 degrees = wind vector pointing towards the east,
222: # 90 degrees = wind vector pointing towards the north
223: def uv2polarangle(u,v):
224:     " Calculates polar angle from u & v wind components "
225:
226:     return (180/np.pi) * np.arctan2(v,u)
227:
228:
229: # ----- Begin Main Program -----
230: starttime = time.time() # Setting variable to calculate execution time
231:
232: # Establishing grid variables
233: lats = np.arange(-90, 90.5, 0.5)
234: lons = np.arange(30, 390, 2./3)
235: ur_lat_ind,ur_lat = min(enumerate(lats), key=lambda i: abs(i[1]-bounds[0]))
236: ur_lon_ind,ur_lon = min(enumerate(lons), key=lambda i: abs(i[1]-bounds[2]))
237: ll_lat_ind,ll_lat = min(enumerate(lats), key=lambda i: abs(i[1]-bounds[1]))
238: ll_lon_ind,ll_lon = min(enumerate(lons), key=lambda i: abs(i[1]-bounds[3]))
239: sub_lats = lats[ll_lat_ind:ur_lat_ind]
240: sub_lons = lons[ll_lon_ind:ur_lon_ind]
241: nrows = len(sub_lats)
242: ncols = len(sub_lons)
243:
244: if save_csv:
245:     # Preparing CSV file in which to save AR characteristics
246:     csvfname = 'AR_features_'+region_str+'_'+str(datetime.date.today().toordinal())+'.csv'
247:     fullcsvfile = open(resultpath+csvfname, 'wb')
248:     cw_full = csv.writer(fullcsvfile, delimiter=',')
249:     header = ['yr','mo','dy','hr','id','com_lat','com_lon','n_ext','s_ext','e_ext',\
250:             'w_ext','extent_rat','len_km','len_epix','width_epix','orient_deg',\
251:             'aspect_elw','area_pix','max_intensity','mean_intensity','total_ivta',\
252:             'u_mean','v_mean','qc_flag']
253:     cw_full.writerow(header)
254:
255: # Loading array with seasonal cycle of IVT
256: mean_array = np.load(outpath+data_str+'_IVT_'+method_str+'.npy')\
257:    [:,ll_lat_ind:ur_lat_ind,ll_lon_ind:ur_lon_ind]
258:

```



```

345:         area = blob.area
346:         if width > 0.:
347:             aspect = length/np.float(width)
348:         else:
349:             aspect = 0
350:         meanint = blob.mean_intensity
351:         y0, x0 = blob.weighted_centroid
352:
353:         if (aspect < min_aspect or length < min_length):
354:             # Clearing the blobs that do not meet the aspect ratio or min length
355:             label_array[label_array == blob.label] = 0
356:             logic_flag = 'asp-len'
357:             asplen_cnt += 1
358:
359:         elif (abs(sub_lats[y0]) < lat_forceorient and \
360:              abs(orientation) < min_orientation) or \
361:              meanint < min_meanintensity:
362:             # Clearing the blobs that do not meet the orientation or mean intensity
363:             label_array[label_array == blob.label] = 0
364:             logic_flag = 'or-int'
365:             orint_cnt += 1
366:
367:         elif (abs(sub_lats[y0]) < lat_forceorient and \
368:              blob.eccentricity < min_eccentricity) or \
369:              (abs(sub_lats[y0]) >= lat_forceorient and \
370:              blob.eccentricity < min_eccentricity - 0.1):
371:             # Attempting to filter out blobs that lack "filament-like" character
372:             label_array[label_array == blob.label] = 0
373:             logic_flag = 'filament'
374:             fila_cnt += 1
375:
376:         elif abs(sub_lats[y0]) < (lat_forceorient + 5.) and \
377:              blob.eccentricity > min_eccentricity and \
378:              blob.mean_intensity > 375 and blob.extent > .50:
379:             # Attempting to remove solid, round, intense blobs
380:             # that may be TCs or similar storms
381:             label_array[label_array == blob.label] = 0
382:             logic_flag = 'tc-like'
383:             tcish_cnt += 1
384:
385:         elif abs(sub_lats[y0]) < lat_forceorient and \
386:              (int(blob.area) - blob.filled_area) <= -5:
387:             # Removing blobs with sizeable holes (mostly TCs "eyes")
388:             label_array[label_array == blob.label] = 0
389:             logic_flag = 'tc-hole'
390:             tchole_cnt += 1
391:
392:         elif abs(sub_lats[y0]) <= lat_cutoff or \
393:              (abs(sub_lats[y0]) <= lat_forceorient and \
394:              abs(orientation) <= min_forceorient):
395:             # Further scrutinizing blobs in the tropics to remove
396:             # west-east features along ITCZ
397:             label_array[label_array == blob.label] = 0
398:             logic_flag = 'tropics'
399:             tropics_cnt += 1
400:
401:         elif aspect < 1.90 and blob.solidity > 0.90 and blob.extent > 0.40:
402:             # Attempting to allow for curvature of AR, but removing
403:             # "solid" elliptical blobs
404:             label_array[label_array == blob.label] = 0
405:             logic_flag = 'solid'
406:             solid_cnt += 1
407:
408:         else:
409:             # Performing additional tests on a blob that would otherwise be retained
410:             temp_mask = np.zeros([nrows,ncols])
411:             temp_mask[label_array == blob.label] = 1
412:             peaks = peak_local_max(ivt_anom*temp_mask, min_distance=14, num_peaks=4)
413:
414:             if len(peaks) > 1:
415:                 blob_image_array = ivt_anom * (label_array == blob.label)
416:                 bigblob_mask = np.ones([nrows,ncols]) * -1
417:                 bigblob_mask[label_array == blob.label] = 0
418:
419:                 for peak_ind, peak_coords in enumerate(peaks):
420:                     bigblob_mask[peak_coords[0],peak_coords[1]] = peak_ind+1
421:
422:                 segmented_blob = random_walker(blob_image_array, bigblob_mask, beta=75)
423:
424:                 for sub_label in np.unique(segmented_blob):
425:                     if sub_label == -1:
426:                         continue
427:                     # -1 is the background; negative values are ignored
428:                     else:
429:                         segmented_blob_mask = np.zeros([nrows,ncols],dtype='int')
430:                         segmented_blob_mask[segmented_blob == sub_label] = sub_label

```

```

431:         segment_props = regionprops(segmented_blob_mask,\
432:                                     intensity_image=ivt_anom)
433:         sy0, sx0 = segment_props[0].weighted_centroid
434:
435:         if abs(sub_lats[sy0]) < (lat_forceorient + 7.5) and \
436:            int(segment_props[0].area) != segment_props[0].filled_area:
437:             # Removing segments with noticeable holes (i.e., mostly TCs)
438:             segmented_blob_mask[segmented_blob == sub_label] = 0
439:             label_array[segmented_blob == sub_label] = 0
440:             qc_flag = 'segment'
441:             logic_flag = 'multi-peak-hole'
442:             mphole_cnt += 1
443:
444:         elif segment_props[0].eccentricity <= min_eccentricity - 0.1:
445:             # Removing segments that don't seem filament-like
446:             segmented_blob_mask[segmented_blob == sub_label] = 0
447:             label_array[segmented_blob == sub_label] = 0
448:             qc_flag = 'segment'
449:             logic_flag = 'multi-peak-1a'
450:             mpla_cnt += 1
451:
452:         elif segment_props[0].mean_intensity < min_meanintensity:
453:             # Removing segments that appears as weak connectors
454:             #   between two IVT peaks
455:             segmented_blob_mask[segmented_blob == sub_label] = 0
456:             label_array[segmented_blob == sub_label] = 0
457:             qc_flag = 'segment'
458:             logic_flag = 'multi-peak-1b'
459:             mplb_cnt += 1
460:
461:         elif abs(sub_lats[sy0]) <= lat_cutoff or \
462:            ((abs(sub_lats[sy0]) <= lat_forceorient - 7.5) and \
463:             abs(segment_props[0].orientation) <= min_forceorient):
464:             # Removing segments with a west-east orientation
465:             #   along the ITCZ
466:             segmented_blob_mask[segmented_blob == sub_label] = 0
467:             label_array[segmented_blob == sub_label] = 0
468:             qc_flag = 'segment'
469:             logic_flag = 'multi-peak-1c'
470:             mp1c_cnt += 1
471:
472:         elif (sub_lats[sy0] > 0. \
473:              and sub_lats[sy0] < lat_forceorient \
474:              and abs(-1*np.rad2deg(segment_props[0].orientation))<10)\
475:              or (sub_lats[sy0] < 0. \
476:                  and sub_lats[sy0] > -1*lat_forceorient \
477:                  and abs(-1*np.rad2deg(segment_props[0].orientation))\
478:                     < 10):
479:             # Scrutinizing the orientation of
480:             #   tropical/subtropical segments
481:             segmented_blob_mask[segmented_blob == sub_label] = 0
482:             label_array[segmented_blob == sub_label] = 0
483:             qc_flag = 'segment'
484:             logic_flag = 'multi-peak-deg'
485:             mpdeg_cnt += 1
486:
487:         else:
488:             segmented_blob_mask[segmented_blob == sub_label] = 1
489:
490:         # Recalculating CoM based on remaining blob segments
491:         label_array[label_array*segmented_blob_mask == blob.label] = blob.label
492:         temp_props = regionprops(label_array == blob.label, \
493:                                 intensity_image=ivt_anom)
494:         if len(temp_props) > 0:
495:             y0, x0 = temp_props[0].weighted_centroid
496:         else:
497:             continue
498:
499:     # Testing the portion of the blob greater than its own average intensity
500:     interior_array = np.where(ivt_anom*temp_mask > meanint, 1, 0)
501:     interior_mass = regionprops(interior_array, intensity_image=ivt_anom)
502:     if len(interior_mass) > 0. and \
503:        interior_mass[0].eccentricity < min_eccentricity - 0.075:
504:         # Finally, removing blobs the interior of which
505:         #   aren't sufficiently filament-like
506:         label_array[label_array == blob.label] = 0
507:         qc_flag = 'segment'
508:         logic_flag = 'multi-peak-4'
509:         mp4_cnt += 1
510:
511:     else:
512:         retained_cnt += 1
513:         all_retained_cnt += 1
514:         clats.append(sub_lats[sy0])
515:         clons.append(sub_lons[x0])
516:         label_array[label_array == blob.label] = retained_cnt # Renumbering

```

```

517:         blabels.append(retained_cnt)
518:         # (Re)Calculating characteristics of retained features
519:         if qc_flag == 'segment':
520:             orient_deg = int(-1*np.rad2deg(temp_props[0].orientation))
521:             if blob.bbox[2] >= nrows:
522:                 n_ext = bounds[0]
523:             else:
524:                 n_ext = sub_lats[temp_props[0].bbox[2]]
525:             if blob.bbox[3] >= ncols:
526:                 e_ext = bounds[2]
527:             else:
528:                 e_ext = np.around(sub_lons[temp_props[0].bbox[3]], decimals=1)
529:                 s_ext = sub_lats[temp_props[0].bbox[0]]
530:                 w_ext = np.around(sub_lons[temp_props[0].bbox[1]], decimals=1)
531:                 length = temp_props[0].major_axis_length
532:                 width = temp_props[0].minor_axis_length
533:                 if width > 0.:
534:                     aspect = length/width
535:                 else:
536:                     aspect = 0
537:         else:
538:             orient_deg = int(-1*np.rad2deg(orientation))
539:             # -1 since the array is actually flipped
540:             if blob.bbox[2] >= nrows:
541:                 n_ext = bounds[0]
542:             else:
543:                 n_ext = sub_lats[blob.bbox[2]]
544:             if blob.bbox[3] >= ncols:
545:                 e_ext = bounds[2]
546:             else:
547:                 e_ext = np.around(sub_lons[blob.bbox[3]], decimals=1)
548:                 s_ext = sub_lats[blob.bbox[0]]
549:                 w_ext = np.around(sub_lons[blob.bbox[1]], decimals=1)
550:
551:         if wind_based_orient:
552:             mean_u = np.nanmean(np.where(\
553:                 label_array == retained_cnt, regional_u[dtg_ind,:,:), np.nan))
554:             mean_v = np.nanmean(np.where(\
555:                 label_array == retained_cnt, regional_v[dtg_ind,:,:), np.nan))
556:             vect_deg = uv2polarangle(mean_u, mean_v)
557:
558:             true_orient = 'na'
559:             if np.abs(orient_deg - vect_deg) <= 45.:
560:                 flag = 'similar'
561:                 true_orient = orient_deg
562:             elif np.abs(orient_deg - vect_deg) >= 135. \
563:                 and np.abs(orient_deg - vect_deg) <= 225.:
564:                 flag = 'opposite'
565:                 true_orient = orient_deg + 180
566:             else:
567:                 flag = 'not close'
568:                 true_orient = orient_deg
569:             if qc_flag == 'na':
570:                 qc_flag = 'off_orient'
571:             else:
572:                 qc_flag = qc_flag+'/'+'off_orient'
573:
574:         x,y,length = skeleton_length(np.where(\
575:             label_array == retained_cnt, 1, 0))
576:         total_IVTa = int(np.sum(ivt_anom * np.where(\
577:             label_array == retained_cnt, 1, 0)))
578:
579:         if qc_flag == 'segment':
580:             datarow_all = [yr,mo,d_str,hr,retained_cnt,sub_lats[y0],\
581:                 sub_lons[x0],n_ext,s_ext,e_ext,w_ext,\
582:                 np.around(temp_props[0].extent,decimals=2),\
583:                 length,np.around(length,decimals=2),\
584:                 np.around(width,decimals=2),true_orient,\
585:                 np.around(aspect,decimals=1),\
586:                 int(temp_props[0].area),\
587:                 np.around(temp_props[0].max_intensity,decimals=2),\
588:                 np.around(temp_props[0].mean_intensity,decimals=2),\
589:                 total_IVTa,mean_u,mean_v,qc_flag]
590:         else:
591:             datarow_all = [yr,mo,d_str,hr,retained_cnt,sub_lats[y0],\
592:                 sub_lons[x0],n_ext,s_ext,e_ext,w_ext,\
593:                 np.around(blob.extent,decimals=2),length,\
594:                 np.around(length,decimals=2),\
595:                 np.around(width,decimals=2),true_orient,\
596:                 np.around(aspect,decimals=1),int(area),\
597:                 np.around(blob.max_intensity,decimals=2),\
598:                 np.around(blob.mean_intensity,decimals=2),\
599:                 total_IVTa,mean_u,mean_v,qc_flag]
600:
601:         if save_csv:
602:             cw_full.writerow(datarow_all)

```

```

603:
604:         # Enable output of Anom IVT array for ARs for compositing
605:         if save_ar_ivt:
606:             ar_ivt_array = np.copy(ivt_anom)
607:             # Retaining IVT Anom. values only within the identified ARs
608:             ar_ivt_array[label_array == 0] = 0
609:             ar_ivt_array_3D[hr_ind, :, :] = ar_ivt_array
610:             del ar_ivt_array
611:
612:         if save_ar_ivt:
613:             save_binary(ar_ivt_array_3D, yr, mo, d_str, 'IVT_6hly_'+region_str+'_ARs')
614:
615: if save_csv:
616:     fullcsvfile.close()
617:
618:
619: # ----- End of Program -----
620: print '\nNumber of Total Features:', blob_cnt
621: print 'Number Retained:', all_retained_cnt, 'or', 100.*all_retained_cnt/blob_cnt, '%'
622: print 'Number Cut By: (* will not sum to total cut due to segmented logic)'
623: print '  asplen_cnt\t', asplen_cnt
624: print '  orint_cnt\t', orint_cnt
625: print '  fila_cnt\t', fila_cnt
626: print '  tcish_cnt\t', tcish_cnt
627: print '  tchole_cnt\t', tchole_cnt
628: print '  tropics_cnt\t', tropics_cnt
629: print '  solid_cnt\t', solid_cnt
630: print '  mphole_cnt\t', mphole_cnt
631: print '  mpla_cnt\t', mpla_cnt
632: print '  mplb_cnt\t', mplb_cnt
633: print '  mplc_cnt\t', mplc_cnt
634: print '  mpdeg_cnt\t', mpdeg_cnt
635: print '  mp4_cnt\t', mp4_cnt
636:
637: endtime = time.time()
638: elapsedtime_min = (endtime - starttime) / 60
639:
640: print '\nEnd of Program'
641: print 'Total Execution Time =', str(elapsedtime_min), 'mins'

```