1. # Import system modules
2. import math
3. import arcpy
4. from arcpy import env
5. from arcpy.sa import \*
6. import csv
8. #-------------------------------------------------
10. # Set all the files that are input in model
12. # ECONOMIC criteria
13. inpSlope = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\Tlokweng\_MCD.gdb\\Slope\_WA"
14. inpRoads = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Roads\_line\_buffer\\Roads.shp"
15. inpDrainage = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Strah\_Diss.shp"
16. inpFlooding = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\flood\_fin\_rec"
17. inpGeology = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Geology\_Tlokw.shp"
18. inpSoil = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Soil\_Dissolved.shp"
20. # ENVIRONMENTAL criteria
21. inpNatural = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\Tourism\_zone.shp"
22. inpSurfaceWater = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Surface\_water.shp"
23. inpRivers = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\Rivers.shp"
24. inpGroundWater = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\grWat\_poll.shp"
26. # SOCIAL criteria
27. inpLULC = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\lu\_lc\_build.tif"
28. inpNoisePoll = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\LU\_res\_dissol\_1.shp"
29. inpAirPoll = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\LU\_res\_dissol\_1.shp"
30. inpVillageExp = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\Village\_expansion.shp"
32. #-------------------------------------------------
34. # Set environment settings
35. outFolder = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokweng\\"
36. outFolderWS = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengWS\\"
37. outFolderWO = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengWO\\"
38. outFolderCostDistance = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengCD\\"
39. outFolderCostPath = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengCP\\"
40. outFolderPolyline= "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengPolyline\\"
41. outFolderNumCells= "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengNumCells\\"
42. outFolderNumMask= "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengMask\\"
43. outFolderFinal= "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\FinalResultTlokwengFinal\\"
45. #-------------------------------------------------
47. arcpy.env.overwriteOutput = True
49. # Set extent variable
50. extFeatureClass = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\Tlokweng\_MCD.gdb\\Tlokweng\_boundary"
51. desc = arcpy.Describe (extFeatureClass)
52. extent = desc.Extent
53. xMin = extent.XMin
54. xMax = extent.XMax
55. yMin = extent.YMin
56. yMax = extent.YMax
58. # Set extent for the feature classes
59. arcpy.env.extent = "MAXOF"
60. arcpy.env.extent = arcpy.Extent (xMin, yMin, xMax, yMax)
61. arcpy.env.cellSize = "MINOF"
63. # Set < extract by mask > feature classes
64. extMask = "C:\\Users\\mis\\Desktop\\MCD\_Baza\\ModelData\\StandardizedData\\Tlokweng\_boundary.shp"
66. #-------------------------------------------------
67. # Check out the ArcGIS Spatial Analyst extension license
68. arcpy.CheckOutExtension("Spatial")
70. **try**:
71. #------------
72. # 1
73. # Criteria SLOPE
74. # Reclassify SLOPE
75. outRecSlope = outFolder + "RecSlopeEcon.tif"
76. RemapRangeSlope =  arcpy.sa.RemapRange([[0, 3, 1], [3, 5, 2], [5, 8, 3],[8, 12, 4], [12, 100000, 5]])
77. rec\_Slope = arcpy.sa.Reclassify (inpSlope, "VALUE", RemapRangeSlope, "NODATA")
78. rec\_Slope.save (outRecSlope)
80. print "Criterion 1 - SLOPE is reclassified successfully !!!"
81. print " "
83. # 2
84. # Criteria ROADS
85. # Polygon to Raster
86. valFieldRoads = "Category"
87. outRasterRoads = outFolder + "PolToRast\_roads.tif"
88. assignmentTypeRoads = "CELL\_CENTER"
89. priorityFieldRoads = "NONE"
90. cellSize = 5
91. # Execute PolygonToRaster
92. arcpy.PolygonToRaster\_conversion(inpRoads, valFieldRoads, outRasterRoads, assignmentTypeRoads, priorityFieldRoads, cellSize)
93. print "PolygonToRaster for ROADS is done successfully !!!"
95. # Reclassify  ROADS
96. outRecRoads = outFolder + "RecRoadsEcon.tif"
97. RemapValueRoads = arcpy.sa.RemapValue([["Primary", 5], ["Secondary", 4], ["Teritary", 3], ["Tracks", 1], ["Access",2],["NODATA",0]])
98. rec\_Roads = arcpy.sa.Reclassify (outRasterRoads, "Category", RemapValueRoads, "")
99. rec\_Roads.save (outRecRoads)
100. print "Criterion 2 - ROADS is reclassified successfully !!!"
101. print " "
103. #------------
104. # 3
105. # Criteria DRAINAGE
106. # Polygon to Raster
107. valFieldDrainage = "Strahler"
108. outRasterDrainage = outFolder + "PolToRast\_drainage.tif"
109. assignmentTypeDrainage = "CELL\_CENTER"
110. priorityFieldDraiange = "NONE"
111. cellSize = 5
112. # Execute PolygonToRaster
113. arcpy.PolygonToRaster\_conversion(inpDrainage, valFieldDrainage, outRasterDrainage, assignmentTypeDrainage, priorityFieldDraiange, cellSize)
114. print "Success\_drainage\_PolygonToRaster !!!"
116. # Reclassify  DRAINAGE
117. outRecDraiange = outFolder + "RecDrainageEcon.tif"
118. RemapValueDrainage = arcpy.sa.RemapValue([[1, 2], [2, 3], [3, 3], [4, 4], [5, 4], [6, 5],[7, 5], [8, 5], ["NODATA",0]])
119. rec\_Drainage = arcpy.sa.Reclassify (outRasterDrainage, "Value", RemapValueDrainage, "")
120. rec\_Drainage.save (outRecDraiange)
121. print "Criterion 3 - DRAINAGE is reclassified successfully !!!"
122. print " "
124. #------------
125. # 4
126. # Criteria FLOODING
127. ##    # Euclidian Distance
128. ##    outEucDistFlooding = outFolder + "EucDist\_floodzone.tif"
129. ##    cellSize = 5
130. ##    EucDistFlooding = arcpy.sa.EucDistance(inpFlooding,"", cellSize)
131. ##    EucDistFlooding.save (outEucDistFlooding)
132. ##    print "Success\_flooding\_EucDist !!!"
134. # Reclassify  FLOODING
135. outRecFlooding = outFolder + "RecFloodingEcon.tif"
136. RemapRangeFlooding = arcpy.sa.RemapRange([[1, 1], [2, 2], [3, 3],[4, 4],[5, 5], ["NODATA","NODATA"]])
137. rec\_Flooding = arcpy.sa.Reclassify (inpFlooding, "VALUE", RemapRangeFlooding, "NODATA")
138. rec\_Flooding.save (outRecFlooding)
139. print "Criterion 4 - FLOODING is reclassified successfully !!!"
140. print " "
142. #------------
143. # 5
144. # Criteria GEOLOGY
145. # Polygon to Raster
146. valFieldGeology = "Category"
147. outRasterGeology = outFolder + "PolToRast\_geology.tif"
148. assignmentTypeGeology = "CELL\_CENTER"
149. priorityFieldGeology = "NONE"
150. cellSize = 5
151. # Execute PolygonToRaster
152. arcpy.PolygonToRaster\_conversion(inpGeology, valFieldGeology, outRasterGeology, assignmentTypeGeology, priorityFieldGeology, cellSize)
153. print "Success\_geology\_PolygonToRaster !!!"
155. # Reclassify  GEOLOGY
156. outRecGeology = outFolder + "RecGeologyEcon.tif"
157. RemapValueGeology = arcpy.sa.RemapValue([[1, 1], [2, 4], [3, 3], [4, 2], [5, 4], ["NODATA",0]])
158. rec\_Geology = arcpy.sa.Reclassify (outRasterGeology, "VALUE", RemapValueGeology, "")
159. rec\_Geology.save (outRecGeology)
160. print "Criterion 5 - GEOLOGY is reclassified successfully !!!"
161. print " "
163. #------------
164. # 6
165. # Criteria SOIL
166. # Polygon to Raster
167. valFieldSoil= "Sifra1"
168. outRasterSoil = outFolder + "PolToRast\_soil.tif"
169. assignmentTypeSoil= "CELL\_CENTER"
170. priorityFieldSoil = "NONE"
171. cellSize = 5
172. # Execute PolygonToRaster
173. arcpy.PolygonToRaster\_conversion(inpSoil, valFieldSoil, outRasterSoil, assignmentTypeSoil, priorityFieldSoil, cellSize)
174. print "Success\_soil\_PolygonToRaster !!!"
176. # Reclassify  SOIL
177. outRecSoil = outFolder + "RecSoilEcon.tif"
178. RemapValueSoil = arcpy.sa.RemapValue([["A1-A4b-A30a", 2], ["A4b-A30a", 4], ["A9", 4], ["A9-A11", 4], ["B3a", 4], ["G10c", 1], ["G1a-G1c", 5], ["G2d-G1a", 3],["R", 4],["NODATA",0]])
179. rec\_Soil = arcpy.sa.Reclassify (outRasterSoil, "Sifra1", RemapValueSoil, "")
180. rec\_Soil.save (outRecSoil)
181. print "Criterion 6 - SOIL is reclassified successfully !!!"
182. print " "

185. #------------
186. # 7
187. # Criteria NATURAL\_AREAS
188. # Euclidian Distance
189. outEucDistNaturalAreas = outFolder + "EucDist\_natural.tif"
190. cellSize = 5
191. EucDistNatural = arcpy.sa.EucDistance(inpNatural,"", cellSize)
192. EucDistNatural.save (outEucDistNaturalAreas)
193. print "Success\_natural\_areas\_EucDist !!!"
194. # Reclassify  NATURAL\_AREAS
195. outRecNatural = outFolder + "RecNaturalEnv.tif"
196. RemapRangeNatural = arcpy.sa.RemapRange([[0, 100, 5], [100, 200, 4], [200, 300, 3],[300, 500, 2],[500, 100000, 1]])
197. rec\_Natural = arcpy.sa.Reclassify (outEucDistNaturalAreas, "VALUE", RemapRangeNatural, "NODATA")
198. rec\_Natural.save (outRecNatural)
199. print "Criterion 7 - NATURAL\_AREAS is reclassified successfully !!!"
200. print " "
202. #------------
203. # 8
204. # Criteria SURFACE\_WATER
205. # Euclidian Distance
206. outEucDistSurfaceWater = outFolder + "EucDist\_SurWater.tif"
207. cellSize = 5
208. EucDistSurWater = arcpy.sa.EucDistance(inpSurfaceWater,"", cellSize)
209. EucDistSurWater.save (outEucDistSurfaceWater)
210. print "Success\_surface\_water\_EucDist !!!"
211. # Reclassify  SURFACE\_WATER
212. outRecSurfWat = outFolder + "RecSurfWatEnv.tif"
213. RemapRangeSurfWat = arcpy.sa.RemapRange([[0, 100, 5], [100, 200, 4], [200, 350, 3],[350, 500, 2],[500, 100000, 1]])
214. rec\_SurfWat = arcpy.sa.Reclassify (outEucDistSurfaceWater, "VALUE", RemapRangeSurfWat, "NODATA")
215. rec\_SurfWat.save (outRecSurfWat)
216. print "Criterion 8 - SURFACE\_WATER is reclassified successfully !!!"
217. print " "
219. #------------
220. # 9
221. # Criteria RIVERS
222. # Euclidian Distance
223. outEucDistRivers = outFolder + "EucDist\_Rivers.tif"
224. cellSize = 5
225. EucDistRivers = arcpy.sa.EucDistance(inpRivers,"", cellSize)
226. EucDistRivers.save (outEucDistRivers)
227. print "Success\_rivers\_EucDist !!!"
228. # Reclassify  RIVERS
229. outRecRivers = outFolder + "RecRiversEnv.tif"
230. RemapRangeRivers = arcpy.sa.RemapRange([[0, 100, 5], [100, 200, 4], [200, 350, 3],[350, 500, 2],[500, 100000, 1]])
231. rec\_Rivers = arcpy.sa.Reclassify (outEucDistRivers, "VALUE", RemapRangeRivers, "NODATA")
232. rec\_Rivers.save (outRecRivers)
233. print "Criterion 9 - RIVERS is reclassified successfully !!!"
234. print " "
236. #------------
238. # 10
239. # Criteria GROUND\_WATER
240. # Euclidian Distance
241. outEucDistGrndWat = outFolder + "EucDist\_GrndWat.tif"
242. cellSize = 5
243. EucDistGrndWat = arcpy.sa.EucDistance(inpGroundWater,"", cellSize)
244. EucDistGrndWat.save (outEucDistGrndWat)
245. print "Success\_ground\_water\_EucDist !!!"
246. # Reclassify  GROUND\_WATER
247. outRecGrndWat = outFolder + "RecGrndWatEnv.tif"
248. RemapRangeGrndWat = arcpy.sa.RemapRange([[0, 150, 5], [150, 250, 4], [250, 350, 3],[350, 500, 2],[500, 100000, 1]])
249. rec\_GrndWat = arcpy.sa.Reclassify (outEucDistGrndWat, "VALUE", RemapRangeGrndWat, "NODATA")
250. rec\_GrndWat.save (outRecGrndWat)
251. print "Criterion 10 - GROUND\_WATER is reclassified successfully !!!"
252. print " "
254. #------------

257. # 11
258. # Criteria LC\_LU
259. # Reclassify LC\_LU
260. outRecLCLU = outFolder + "RecLCLUSocio.tif"
261. RemapValueLCLU =  arcpy.sa.RemapValue([["No\_Data", 2], ["SparseForest", 4], ["Woodland", 5],["OpenGrassland", 1], ["ClosedShrubland", 2], ["OpenShrubland", 2],["WaterBody", 5], ["Settlement", 5], ["Buildings", 5], ["LI", 2],["GI", 2], ["MXD-G", 5], ["C-3", 4], ["CIV", 4], ["R-1", 5], ["RSU", 5],["R", 5], ["MXD-C", 5], ["OS", 2],["R-2", 5], ["C-1", 5], ["iA-1", 5], ["Woodlands", 5], ["RU", 5], ["eA-2", 5], ["CBD", 5], ["'iA-1", 5], ["NODATA", "NODATA"]])
262. rec\_LCLU = arcpy.sa.Reclassify (inpLULC, "LULC", RemapValueLCLU, "NODATA")
263. rec\_LCLU.save (outRecLCLU)
264. #------------
265. print "Criterion 11 - LC-LU is reclassified successfully !!!"
266. print " "
268. #------------
270. # 12
271. # Criteria NOISE\_POLLUTION
272. # Euclidian Distance
273. outEucDistNoise = outFolder + "EucDist\_Noise.tif"
274. cellSize = 5
275. EucDistNoise = arcpy.sa.EucDistance(inpNoisePoll,"", cellSize)
276. EucDistNoise.save (outEucDistNoise)
277. print "Success\_noise\_pollution\_EucDist !!!"
278. # Reclassify  NOISE\_POLLUTION
279. outRecNoise = outFolder + "RecNoiseSocio.tif"
280. RemapRangeNoise = arcpy.sa.RemapRange([[0, 25, 5], [25, 50, 4], [50, 125, 3],[125, 250, 2],[250, 100000, 1]])
281. rec\_Noise = arcpy.sa.Reclassify (outEucDistNoise, "VALUE", RemapRangeNoise, "NODATA")
282. rec\_Noise.save (outRecNoise)
283. print "Criterion 12 - NOISE\_POLLUTION is reclassified successfully !!!"
284. print " "

287. #------------
289. # 13
290. # Criteria AIR\_POLLUTION
291. # Euclidian Distance
292. outEucDistAir = outFolder + "EucDist\_Air.tif"
293. cellSize = 5
294. EucDistAir = arcpy.sa.EucDistance(inpAirPoll,"", cellSize)
295. EucDistAir.save (outEucDistAir)
296. print "Success\_air\_pollution\_EucDist !!!"
297. # Reclassify  AIR\_POLLUTION
298. outRecAir = outFolder + "RecAirSocio.tif"
299. RemapRangeAir = arcpy.sa.RemapRange([[0, 100, 5], [100, 200, 4], [200, 300, 3],[300, 500, 2],[500, 100000, 1]])
300. rec\_Air = arcpy.sa.Reclassify (outEucDistAir, "VALUE", RemapRangeAir, "NODATA")
301. rec\_Air.save (outRecAir)
302. print "Criterion 13 - AIR\_POLLUTION is reclassified successfully !!!"
303. print " "

306. except:
307. print arcpy.GetMessages ()
308. print "ERRORR!!!"

311. # ------------------------------------------------
313. # Create a list of raster datasets for ECONOMIC theme
314. #Criteria weights for thema - Economy
315. baseSlope = 0.30
316. baseRoads = 0.22
317. baseDrainage = 0.21
318. baseFlooding = 0.12
319. baseGeology = 0.08
320. baseSoil = 0.07
322. #Criteria weights for thema - Enviro
323. baseNatural = 0.30
324. baseSurWater = 0.35
325. baseRivers = 0.25
326. baseGrdWater = 0.1

329. #Criteria weights for thema - Socio
330. baseNoise = 0.09
331. baseAir = 0.14
332. baseLCLU = 0.77
333. #baseExpension = 0.3

336. # thema economy
337. themaEconomicNames = [outRecSlope, outRecRoads, outRecDraiange, outRecFlooding, outRecGeology, outRecSoil]
338. themaEconomicWeights = [baseSlope, baseRoads, baseDrainage, baseFlooding, baseGeology, baseSoil]
339. themaEconomicOut = ["Slope", "Roads", "Drain", "Flood", "Geol", "Soil"]
341. # thema enviro
342. themaEnviroNames = [outRecNatural, outRecSurfWat, outRecRivers ,outRecGrndWat]
343. themaEnviroWeights = [baseNatural, baseSurWater, baseRivers, baseGrdWater]
344. themaEnviroOut = ["Natr", "SurWat", "Rivr", "GrdWat"]
346. # thema socio
347. themaSocioNames = [outRecNoise, outRecAir, outRecLCLU]
348. themaSocioWeights = [baseNoise, baseAir, baseLCLU]
349. themaSocioOut = ["Noise", "Air", "LCLU"]
351. # theams combined
352. allThemasNames = [themaEconomicNames,themaEnviroNames,themaSocioNames ]
353. allThemasWeights = [themaEconomicWeights, themaEnviroWeights, themaSocioWeights]
354. allThemasOut = [themaEconomicOut, themaEnviroOut, themaSocioOut]

357. # ------------------------------------------------
358. # WEIGHTED SUM FOR ECONOMICAL CRITERIA
359. dictEcon = dict(zip(themaEconomicNames, themaEconomicWeights))
360. print dictEcon
361. print " "
362. print " "
363. wsTableEcon = []
364. dictEconKeys = dictEcon.keys ()
365. **for** s in dictEconKeys:
366. dictEconValues = dictEcon[s]
367. wsTableEcon.append ([s,"VALUE", dictEconValues])
368. print wsTableEcon
369. print " "
370. print " "
371. WSumTableObjEcon = WSTable(wsTableEcon)
372. outWeightedSumEcon = WeightedSum (WSumTableObjEcon)
373. outWeightedSumEcon.save (outFolder + "EconBase\_WS")
375. # RECLASSIFY ECONOMY WEIGHTED SUM
376. outRecEcn = outFolder + "RecEconBase.tif"
377. RemapRangeEcon = arcpy.sa.RemapRange([[0, 1, 1], [1, 2, 2], [2, 3, 3],[3, 4, 4],[4, 5, 5]])
378. rec\_econ = arcpy.sa.Reclassify (outWeightedSumEcon, "VALUE", RemapRangeEcon, "NODATA")
379. rec\_econ.save (outRecEcn)
380. print "RECLASSIFACTION OF ECONOMY BASE is done successfully !!!"
381. print " "

384. # ------------------------------------------------
385. # WEIGHTED SUM FOR ENVIRONMENTAL CRITERIA
386. dictEnv = dict(zip(themaEnviroNames, themaEnviroWeights))
387. print dictEnv
388. print " "
389. print " "
390. wsTableEnv = []
391. dictEnvKeys = dictEnv.keys ()
392. **for** s in dictEnvKeys:
393. dictEnvValues = dictEnv[s]
394. wsTableEnv.append ([s,"VALUE", dictEnvValues])
395. print wsTableEnv
396. print " "
397. print " "
398. WSumTableObjEnv = WSTable(wsTableEnv)
399. outWeightedSumEnv = WeightedSum (WSumTableObjEnv)
400. outWeightedSumEnv.save (outFolder + "EnvBase\_WS")
402. # RECLASSIFY ENVIRONMENTAL WEIGHTED SUM
403. outRecEnv = outFolder + "RecEnvBase.tif"
404. RemapRangeEnv = arcpy.sa.RemapRange([[0, 1, 1], [1, 2, 2], [2, 3, 3],[3, 4, 4],[4, 5, 5]])
405. rec\_env = arcpy.sa.Reclassify (outWeightedSumEnv, "VALUE", RemapRangeEnv, "NODATA")
406. rec\_env.save (outRecEnv)
407. print "RECLASSIFACTION OF ENVIRO BASE is done successfully !!!"
408. print " "
409. # --------------------------------------------------------
411. # WEIGHTED SUM FOR SOCIAL CRITERIA
412. dictSocio = dict(zip(themaSocioNames, themaSocioWeights))
413. print dictSocio
414. print " "
415. print " "
416. wsTableSocio = []
417. dictSocioKeys = dictSocio.keys ()
418. **for** s in dictSocioKeys:
419. dictSocioValues = dictSocio[s]
420. wsTableSocio.append ([s,"VALUE", dictSocioValues])
421. print wsTableSocio
422. print " "
423. print " "
424. WSumTableObjSocio = WSTable(wsTableSocio)
425. outWeightedSumSocio = WeightedSum (WSumTableObjSocio)
426. outWeightedSumSocio.save (outFolder + "SocioBase\_WS")
428. # RECLASSIFY ENVIRONMENTAL WEIGHTED SUM
429. outRecSocio = outFolder + "RecSocioBase.tif"
430. RemapRangeSocio = arcpy.sa.RemapRange([[0, 1, 1], [1, 2, 2], [2, 3, 3],[3, 4, 4],[4, 5, 5]])
431. rec\_socio = arcpy.sa.Reclassify (outWeightedSumSocio, "VALUE", RemapRangeSocio, "NODATA")
432. rec\_socio.save (outRecSocio)
433. print "RECLASSIFACTION OF SOCIO BASE is done successfully !!!"
434. print " "

437. # --------------------------------------------------------
439. # Universal remap for Weighted Overlay
440. remapWO = RemapValue([[1,1],[2,2],[3,3],[4,4],[5,5],["NODATA","NODATA"]])



445. **for** sLoop in range (0,3):
447. procent = 0.01 # 1% chanching ing
449. crit = 0
450. critAdj = 0
451. critRemTot = []
452. restList = []
453. product = []
454. adjs = []
455. names = []
456. dictionary = {}

459. print " "
460. print " "
461. print ("IT STARTED" + str(sLoop))
462. print " "
463. print " "
465. #crit\_index = len(themaEconomicWeights)-1
466. #print("list length is " + str(len(themaEconomicWeights)))
468. **for** i, crit in enumerate(allThemasWeights[sLoop]):
470. print " "
471. print(allThemasNames[sLoop][i])
472. print " "
474. critRemTot = [critWeight **for** k, critWeight in enumerate (allThemasWeights[sLoop]) **if** k != i]
475. print critRemTot
476. restList = [name **for** n, name in enumerate (allThemasNames[sLoop]) **if** n != i]
477. print restList
478. product = [s **for** z, s in enumerate(allThemasWeights[sLoop]) **if** z != i]
479. allCritWeigh = sum(product)
480. print "Sum of the weights, not including the main criteria is: " + str (allCritWeigh)
481. print " "
482. print zip(restList, critRemTot)
483. finWeight = [x/allCritWeigh **for** x in critRemTot]
484. print finWeight
485. print " "
487. listCellsTot = []
488. countCells = ""
489. checkList = [1, 2, 3, 4, 5]
491. # Input for comparision among the cells
492. listCellsTotRast = []
493. checkRasterList = [11, 12, 13, 14, 15, 21, 22, 23, 24, 25, 31, 32, 33, 34, 35, 41, 42, 43, 44, 45, 51, 52, 53, 54, 55]

496. **for** j in range(-20 ,21, 1):
498. print " "
500. print("procent: %s" % j)
502. critInc =crit \* procent \* j
504. critAdj = crit + critInc
505. print critAdj
506. print " "
507. adjs = [ round ((1 - critAdj) \* critInProd, 4)  **for** k, (critInProd) in enumerate(finWeight)]
508. names = [name **for** k, name in enumerate (allThemasNames[sLoop]) **if** k != i]
509. print zip(names, adjs)
510. print " "
511. adjs.append (critAdj)
512. names.append (allThemasNames[sLoop] [i])
513. print " The total sum of the weights is: " + str (sum(adjs))
515. print zip(names, adjs)
516. dictionary = dict(zip(names, adjs))
517. print " "
518. print dictionary
519. print " "
520. print " "
521. newList = []
522. sKeys = dictionary.keys()
523. **for** s in sKeys:
524. sValues = dictionary [s]
525. newList.append ([s,"VALUE", sValues])
526. # Create the WSTable object
527. print " "
528. print "New list for " + (("procent: %s" % j))
529. print newList
530. WSumTableObj = WSTable (newList)
531. outWeightedSum = WeightedSum (WSumTableObj)
532. currItt =   allThemasOut[sLoop][i]
533. outWeightedSum.save (outFolderWS + currItt + "\_" + str(j))
534. print " "
535. print " "
537. # DO RECLASIFFY for input in Weighted Overlay Table
538. outRec = outFolderWO + "RecSLoop" + str(sLoop)+ str(j) #+  ".tif" # can include **this** to seperate the rec files but not necessary ------> str(j)+ str(sLoop) +
539. RemapRangeWO = arcpy.sa.RemapRange([[0, 1, 1], [1, 2, 2], [2, 3, 3],[3, 4, 4],[4, 5, 5]])
540. rec\_baseSLoop = arcpy.sa.Reclassify (outWeightedSum, "VALUE", RemapRangeWO, "NODATA")
541. rec\_baseSLoop.save (outRec)
542. print "RECLASSIFACTION OF BASE sLopp is done successfully !!!"
543. print " "
545. **if** sLoop == 0:
547. # ECONOMY thema is the first element in defined python list, thus sLoop = 0
548. # Do Weighted Overlay (first raster within WOTable is always under loop), Economy
549. myWOTable = WOTable([[rec\_baseSLoop, 64, "VALUE", remapWO], [rec\_env, 15, "VALUE", remapWO], [rec\_socio, 21, "VALUE", remapWO]], [1, 5, 1]) # VIDII DA LI MOZES DA AUTOMATIZUJES DA NE BUDE HARD CODED
550. # Execute Weighted Overlay
551. outWeightedOverlay = WeightedOverlay(myWOTable)
552. # Save the output
553. outWeightedOverlay.save(outFolderWO + currItt + "\_" + str(j))
554. print "Raster datasets for Weighted Overlay was created "
555. print " "
556. # Do the extract  by mask
557. outExtractByMask = ExtractByMask (outWeightedOverlay,extMask)
558. outExtractByMask.save (outFolderNumMask + currItt + str(j))
560. elif sLoop == 1:
562. # ENVIRONMETANL thema is the second element in defined python list, thus sLoop = 1
563. # Do Weighted Overlay (first raster within WOTable is always under loop), Economy
564. myWOTable = WOTable([[rec\_baseSLoop, 63, "VALUE", remapWO], [rec\_econ, 11, "VALUE", remapWO], [rec\_socio, 26, "VALUE", remapWO]], [1, 5, 1]) # VIDII DA LI MOZES DA AUTOMATIZUJES DA NE BUDE HARD CODED
565. # Execute Weighted Overlay
566. outWeightedOverlay = WeightedOverlay(myWOTable)
567. # Save the output
568. outWeightedOverlay.save(outFolderWO + currItt + "\_" + str(j))
569. print "Raster datasets for Weighted Overlay was created "
570. print " "
571. # Do the extract  by mask
572. outExtractByMask = ExtractByMask (outWeightedOverlay,extMask)
573. outExtractByMask.save (outFolderNumMask + currItt + str(j))
575. elif sLoop == 2:
577. # SOCIALthema is the second element in defined python list, thus sLoop = 2
578. # Do Weighted Overlay (first raster within WOTable is always under loop), Economy
579. myWOTable = WOTable([[rec\_baseSLoop, 69, "VALUE", remapWO], [rec\_econ, 21, "VALUE", remapWO], [rec\_env, 10, "VALUE", remapWO]], [1, 5, 1]) # VIDII DA LI MOZES DA AUTOMATIZUJES DA NE BUDE HARD CODED
580. # Execute Weighted Overlay
581. outWeightedOverlay = WeightedOverlay(myWOTable)
582. # Save the output
583. outWeightedOverlay.save(outFolderWO + currItt + "\_" + str(j))
584. print "Raster datasets for Weighted Overlay was created "
585. print " "
586. # Do the extract  by mask
587. outExtractByMask = ExtractByMask (outWeightedOverlay,extMask)
588. outExtractByMask.save (outFolderNumMask + currItt + str(j))
590. # Do Cost Distance raster
591. inSourceDataStart = "C:\\Users\\mis\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Points\\Start.shp"
592. inCostDistRast = outWeightedOverlay
593. outBkLinkRaster = (outFolderCostDistance + currItt + "bl\_" + str(j))
594. # Execute CostDistance
595. outCostDistance = CostDistance(inSourceDataStart, inCostDistRast, "", outBkLinkRaster)
596. # Save the output
597. outCostDistance.save(outFolderCostDistance + currItt + "cd\_" +  str(j))
599. print ("Raster datasets for Cost Distance was created " + currItt + "cd\_" + str(j))
601. print " "
603. # Do Cost Path raster
604. inSourceDataEnd = "C:\\Users\\mis\Desktop\\MCD\_Baza\\ModelData\\ProcessedData\\Points\\End1.shp"
605. inCostPathRas = outCostDistance
606. outCostBLinkRaster = outBkLinkRaster
607. method = "EACH\_CELL"
608. # Execute Cost Path
609. outCostPath = CostPath(inSourceDataEnd, inCostPathRas , outCostBLinkRaster, method, "")
610. # Save the output
611. outCostPath.save(outFolderCostPath +  currItt + "cp\_" +  str(j))
613. print ("Raster datasets for Cost Path was created " + "cp\_" + str(j))
615. print " "
617. # Convert raster path to polyine
618. inRasterToPolyline =  outCostPath
619. outPolyline = (outFolderPolyline +  currItt + "cp\_" +  str(j))
620. backgrVal = "ZERO"
621. field = "VALUE"
622. # Execute RasterToPolygon
623. arcpy.RasterToPolyline\_conversion(inRasterToPolyline, outPolyline, backgrVal, "", "SIMPLIFY", field)
625. print ("Raster datasets for Cost Path was converted to polyline shapefile " +  currItt + "cp\_"+ str(j))
627. print " "
628. # Following IF-statements can be removed !!!
629. **if** sLoop == 0:
631. # Perfrom raster calculator to calculate the number of cells
632. #numCells = (outCostPath & outWeightedOverlay) \* outWeightedOverlay
633. #numCells.save (outFolderNumCells+  currItt + "cp\_" + str(sLoop)+ str(j))
635. # start with exporting data
636. countCells = str(j)
637. rows = arcpy.SearchCursor (outExtractByMask)
638. nullList = [countCells, 0, 0, 0, 0, 0]
639. **for** row in rows:
640. **for** elem in checkList:
641. **if** row.Value == elem:
642. nullList [elem] = row.Count
643. listCellsTot.append (nullList)
644. elif sLoop == 1:
646. # Perfrom raster calculator to calculate the number of cells
647. #numCells = (outCostPath & outWeightedOverlay) \* outWeightedOverlay
648. #numCells.save (outFolderNumCells+  currItt + "cp\_" + str(sLoop)+ str(j))
650. # start with exporting data
651. countCells = str(j)
652. rows = arcpy.SearchCursor (outExtractByMask)
653. nullList = [countCells, 0, 0, 0, 0, 0]
654. **for** row in rows:
655. **for** elem in checkList:
656. **if** row.Value == elem:
657. nullList [elem] = row.Count
658. listCellsTot.append (nullList)
659. elif sLoop == 2:
661. # Perfrom raster calculator to calculate the number of cells
662. #numCells = (outCostPath & outWeightedOverlay) \* outWeightedOverlay
663. #numCells.save (outFolderNumCells+  currItt + "cp\_" + str(sLoop)+ str(j))
665. # start with exporting data
666. countCells = str(j)
667. rows = arcpy.SearchCursor (outExtractByMask)
668. nullList = [countCells, 0, 0, 0, 0, 0]
669. **for** row in rows:
670. **for** elem in checkList:
671. **if** row.Value == elem:
672. nullList [elem] = row.Count
673. listCellsTot.append (nullList)
674. # create list for each criteria
675. listCellsTot.insert (0, ["Iteration", "Cells\_1", "Cells\_2", "Cells\_3", "Cells\_4", "Cells\_5"])
676. print listCellsTot
677. with open (outFolderFinal + allThemasOut[sLoop][i] + ".csv", "wb") as fp:
678. a = csv.writer (fp, delimiter=",")
679. a.writerows (listCellsTot)

682. # Loop over raster datasets, create new .csv files to insert calculated values for cells chanching...
683. # 1 --> 1    1 --> 2  1 --> 3  1 --> 4  1 --> 5 ;    2 --> 1  2 --> 2  2 --> 3  2 --> 4  2 --> 5 ;
684. # 3 --> 1    3 --> 2  3 --> 3  3 --> 4  3 --> 5   ;  4 --> 1  4 --> 2  4 --> 3  4 --> 4  4 --> 5 ;
685. # 5 --> 1    5 --> 2  5 --> 3  5 --> 4  5 --> 5
686. env.workspace = outFolderNumMask
687. rasterList = arcpy.ListRasters (currItt + "\*","")
688. print rasterList
689. print " "
690. print " "
691. rasterBase = Raster (outFolderNumMask + currItt + "0")
693. **for** num in range (-20 ,21,1): # the same range
694. rasterInp =  Raster (outFolderNumMask + currItt + str(num))
695. desc = arcpy.Describe (rasterInp)
696. print desc.dataType
697. print desc.name
699. # Perform raster calculator between BASE and input raster - rasterCal
700. rasterCalOut = outFolderNumMask + currItt + "rast" + str(num)
702. #rastTemp = Con(rasterInp > rasterBase, 1, 2)
703. #rastTemp = Con(rasterBase == 1, (Con(rasterInp == 1, 11, Con(rasterInp == 2, 12 , Con(rasterInp == 3, 13, Con(rasterInp == 4, 14, Con(rasterInp == 5, 15)))))))
704. #rastTemp = Con(rasterBase == 1 , 100, 200)
705. rastTemp =  Con(rasterBase == 1, (Con(rasterInp == 1, 11, Con(rasterInp == 2, 12 , Con(rasterInp == 3, 13, Con(rasterInp == 4, 14, Con(rasterInp == 5, 15)))))), \
706. Con(rasterBase == 2, (Con(rasterInp == 1, 21, Con(rasterInp == 2, 22 , Con(rasterInp == 3, 23, Con(rasterInp == 4, 24, Con(rasterInp == 5, 25)))))), \
707. Con(rasterBase == 3, (Con(rasterInp == 1, 31, Con(rasterInp == 2,32 , Con(rasterInp == 3, 33, Con(rasterInp == 4, 34, Con(rasterInp == 5, 35)))))), \
708. Con(rasterBase == 4, (Con(rasterInp == 1, 41, Con(rasterInp == 2, 42 , Con(rasterInp == 3, 43, Con(rasterInp == 4, 44, Con(rasterInp == 5, 45)))))), \
709. Con(rasterBase == 5, (Con(rasterInp == 1, 51, Con(rasterInp == 2, 52 , Con(rasterInp == 3,53, Con(rasterInp == 4, 54, Con(rasterInp == 5, 55)))))))))))
711. #rastTemp =  Con(rasterBase == 1, (Con(rasterInp == 1, 11, Con(rasterInp == 2, 12 , Con(rasterInp == 3, 13, Con(rasterInp == 4, 14, Con(rasterInp == 5, 15)))))), Con(rasterBase == 2, (Con(rasterInp == 1, 21, Con(rasterInp == 2, 22 , Con(rasterInp == 3, 23, Con(rasterInp == 4, 24, Con(rasterInp == 5, 25)))))), Con(rasterBase == 3, (Con(rasterInp == 1, 31, Con(rasterInp == 2,32 , Con(rasterInp == 3, 33, Con(rasterInp == 4, 34, Con(rasterInp == 5, 35)))))), Con(rasterBase == 4, (Con(rasterInp == 1, 41, Con(rasterInp == 2, 42 , Con(rasterInp == 3, 43, Con(rasterInp == 4, 44, Con(rasterInp == 5, 45)))))), Con(rasterBase == 5, (Con(rasterInp == 1, 51, Con(rasterInp == 2, 52 , Con(rasterInp == 3,53, Con(rasterInp == 4, 54, Con(rasterInp == 5, 55)))))))))))
713. print type(rastTemp)
715. rastTemp.save (rasterCalOut)
716. desc = arcpy.Describe (rasterCalOut)
717. print desc.dataType
718. print desc.name
720. countCellsRast = str(num)
721. rows = arcpy.SearchCursor (rasterCalOut)
722. nullListRast = [countCellsRast, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
723. **for** row in rows:
724. **for** elem in checkRasterList:
725. **if** row.Value == elem:
726. s = checkRasterList.index (elem)
727. nullListRast [s+1] = row.Count
728. listCellsTotRast.append (nullListRast)
729. # create list for each criteria
730. listCellsTotRast.insert (0, ["Iteration", "11", "12", "13", "14", "15" , "21", "22", "23", "24", "25", "31", "32", "33", "34", "35", "41", "42", "43", "44", "45", "51", "52", "53", "54", "55"])
731. print listCellsTotRast
733. with open (outFolderFinal + currItt + "Sensitivty" + ".csv", "wb") as fp:
734. a = csv.writer (fp, delimiter=",")
735. a.writerows (listCellsTotRast)
737. arcpy.CheckInExtension("Spatial")