GIS 540 Exam I

Course number: GIS 540

Test ID: exam I

Instructor: Dr. Tateosian

Time Limit: 120 minutes

Materials allowed: Scrap paper (students may not take scrap paper away from the exam).

General instructions: **Write clearly**. Use complete sentences for short answer questions. Write name and unity login id (e.g. jkrowlin) on each page. Return all pages of the exam and scrap paper to the proctor. Do not discuss the exam with other students before the exams are returned.

Total points available   ___135___

Total points deducted   _____

Total points earned   _____

1. **(5pts)** Given the Python code statements in the left column, specify if the statement in the middle column is true or false.

Python code	Statement	True or False?
species = 'trout'	species is a *string literal*.	FALSE
x = 5.0	x is an integer type variable.	FALSE
"256" > "2400"	The statement evaluates to True.	TRUE
str(5) == '5'	The statement evaluates to True.	TRUE
foo = "GIS" foo[:2]==foo[2]	The second line of code evaluates to True.	FALSE

2. *(18pts)* Label each item in the list as **M** for built-in module, **F** for built-in function, **C** for built-in constant, **E** for built-in exception, **K** for Python keyword, or **N** for none of the other choices. *There are three of each.*

_E_ IndexError	_M_ os	_F_ max	_K_ for	_N_ arcpy
_M_ sys	_E_ SyntaxError	_N_ by	_K_ in	_E_ TypeError
_C_ False	_N_ elseif	_M_ math	_C_ None	
_F_ type	_C_ True	_K_ and	_F_ range	

3. **(5pts)** Write a Python FOR-loop equivalent to the following WHILE-loop

WHILE-loop	FOR-loop

x = 1
while x < 500:
   print(x)
x = x + 1
	for x in range(1,500):
	    print(x)

4. **(18pts)** Beside each code fragment in the table below, give the output. If the code would cause an error, write *ERROR* and give a *brief explanation*.

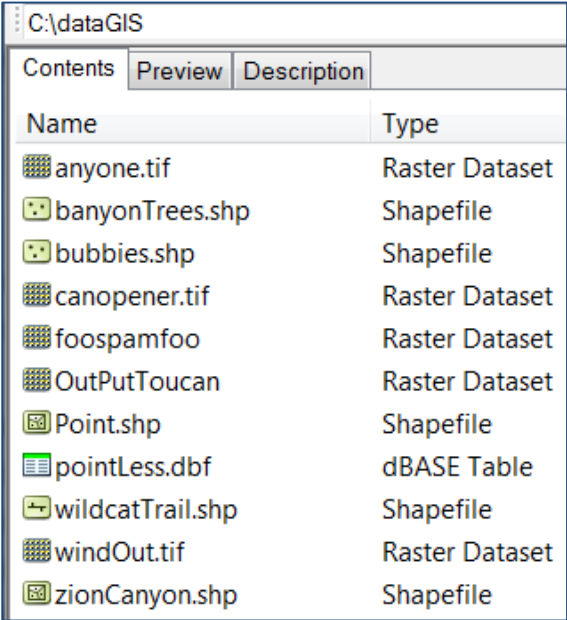| Python code	Output or cause of error
count = 40 name = 'Toucan' **print**('There are' + count + name)	ERROR. The Python interpreter doesn't know whether to add or concatenate. strings and numbers can't be combined using +. Attempting to do this results in a TypeError.
x = (8**2) – (5//10) + (5.0/10) **print**(x)	64.5
y = 2 – 5 * 6 **print**(y)	-28
fieldName = 'rock age' fieldName.upper( ) **print**(fieldName)	rock age
myList = ['C','A','B'] myList.sort() **print**(myList)	['A','B','C']
myList = list(range(4, 12, 2)) **print**(myList)	[4,6,8,10]
print(min(1, 2, 3)) min = 5 **print**(min(1, 2, 3))	The first statement prints 1 The 2nd statement gives an ERROR->The code changed the built-in 'min' function into an Integer type variable.

theTime = "10:33:06" #HH:MM:SS
timeParts = theTime.split(":")
print(timeParts)
print(timeParts[3])
	The first statement prints ['10','33','06']
	
	The 2ND statement gives an ERROR->Trying to index beyond the end of the list throws an IndexError.

x = -5
if x > 0 or 1:
    print ('IN')
else:
    print('OUT')
	IN

5. **(8pts)** Under each arcpy listing method in the table below, give the list contents. If the list is empty, write EMPTY and give a brief explanation. Assume the user passed in the following arguments:

<div align="center">"C:/dataGIS" *foo</div>

Assume the following code has been executed:

<div align="center">import arcpy, sys</div>
<div align="center">arcpy.env.workspace = sys.argv[1]</div>

L1 = arcpy.ListFeatureClasses("*anyon*") "banyonTrees.shp", "zionCanyon.shp"	
L2 = arcpy.ListRasters("sys.argv[2]") EMPTY. This statement tries to list files that are *named* sys.argv[2]	C:\dataGIS Contents Preview Description **Name** — **Type** anyone.tif — Raster Dataset banyonTrees.shp — Shapefile bubbies.shp — Shapefile canopener.tif — Raster Dataset foospamfoo — Raster Dataset OutPutToucan — Raster Dataset Point.shp — Shapefile pointLess.dbf — dBASE Table wildcatTrail.shp — Shapefile windOut.tif — Raster Dataset zionCanyon.shp — Shapefile
L3 = arcpy.ListRasters("Out*", "TIF") EMPTY The names of the TIF type rasters in this workspace do not start with 'Out'.	
L4 = arcpy.ListFeatureClasses("*", "Point") "banyonTrees.shp", "bubbies.shp"	**Figure 1**: Contents of C:\dataGIS in ArcCatalog.

6. **(6pts)** Write a script which prints the names of all the fields in a given input Shapefile. The script has been started in the box below.

1	import arcpy, sys
2	inputFile = sys.argv[1]
3	fields = arcpy.ListFields(inputFile)
4	for f in fields:
5	print (f.name)

7. **(13pts)** Write a script which takes a year as input. Print 'CURRENT', if the input is this year (2015), 'RECENT' if the input is within the 5 years before this one (2009 through 2014), and 'TARDIS' otherwise. Also, list 3 or more sample inputs that would test the code thoroughly.

```
import sys
year = int(sys.argv[1])
if year == 2015:
    print('CURRENT')
elif 2009 <= year <= 2014:
    print('RECENT')
else:
    print('TARDIS' )
```

Sample inputs: 2015, 2010, 2020, -2000, 2000.5, '200AD' (The first three inputs represent categories that are mandatory to test, since they test each of the three branches of code). The code would fail for the last 2 inputs but it still meets the requirements that were specified. This is something you might want to consider how to handle depending on your application.

8. **(4pts)** Answer the following questions about the traceback error shown in the box below:
   a. Which line number of 'findAvg.py' did the error occur on? **Line 8**

   b. What is the name of the exception? **ZeroDivisionError**

   c. The line of code that triggered the error is shown in bold. Use variable names in this line of code to explain why the exception was thrown. **rangeVal has a value of zero.**

```
Traceback (most recent call last):
File "C:\Python27\ArcGIS10.2\Lib\site-packages\scriptutils.py", line 326, in RunScript
    exec codeObject in __main__.__dict__
  File "C:\sampleScripts\findAvg.py", line 8, in <module>
val = maxv - minv/rangeVal
ZeroDivisionError: integer division or modulo by zero
```

9. **(14pts)** Identify the six mistakes in this pseudocode. Mistakes can pertain to logic, syntax, or efficiency. Line numbers for lines that have mistakes are given below (Some lines have 2 mistakes). CLEARLY show how to correct the mistakes *inside the code box.* Then, in parts 1)-7) below the box, explain the corrections.

1	GET a list of shapefiles from workspace
2	FOR each shapefile in the workspace
3	DETERMINE the geometry of the current shapefile
4	IF the geometry is polygon ~~then~~ THEN
5	CALCULATE the area of the polygonal lake in square feet.
6	COMPUTE buffer of the polygonal lake.
7	PRINT area of lake
8	ELSE IF the geometry is line THEN
9	CALCULATE the length of the linear river in feet.
10	COMPUTE buffer of the linear river.
11	**---->**PRINT length of river
12	ENDIF
13	ENDFOR

1) **Line 2 (something is missing!)**   Should be a FOR loop (FOR each shapefile). The first line gets a list of shapefiles to work on.
2) **Line 4**   then should be uppercase, THEN
3) **Line 8**   IF should be ELSE IF. This would be more efficient than repeat IF's, since these conditions are mutually exclusive.
4) **Line 8**   missing THEN

5) **Line 11**   needs to be indented, otherwise it would be printed for any kind of shapefile.

6) **Line 12 (something is missing!)** missing ENDIF

7) **Line 12 (something is missing!)** missing ENDFOR

10. **(8pts)** The script 'polygonNeighbors_nonBatch.py' in the first box below creates a table with statistics pertaining to polygon contiguity for polygons in the 'C:/data/COVER63p.shp' file. Rewrite this as 'polygonNeighbors_Batch.py' in the second box (where the script is started for you). The batch version should call the Polygon Neighbor (Analysis) on all the polygon Shapefiles in the 'C:/data' directory whose names start with "COVER".

1	*# polygonNeighbors_nonBatch.py*
2	import arcpy, os
3	arcpy.env.workspace = 'C:/data'
4	inputFile = 'COVER63p.shp'
5	outputFile = os.path.splitext(inputFile)[0] + 'PN.dbf'
6	arcpy.PolygonNeighbors_analysis(inputFile, outputFile)
7	print('{0} created.'.format(outputFile))

1	*# polygonNeighbors_Batch.py*
2	import arcpy, os
3	arcpy.env.workspace = 'C:/data'
4	fcs = arcpy.ListFeatureClasses('COVER*', 'Polygon')
5	for fc in fcs:
6	outputFile = os.path.splitext(fc)[0] + 'PN.dbf'
7	arcpy.PolygonNeighbors_analysis(fc, outputFile)
8	print('{0} created.'.format(outputFile))

11. **(6pts)** The screen shot in Figure 2 shows a debugging session. Use the information in the Watch Window, the current position of the cursor, and the code you can see to answer the following questions.

    a. What is the name of the file that was input into the script?

       jack.jpg

    b. Which line number is the cursor currently pointing to?

       Line 27

    c. Starting at the current cursor position, when the user presses the 'Step Over' button 10 times, which sound(s) will play?

       Based on the code shown here, 'haha.wav', 'doh.wav', and 'yeehaa.wav' are the sounds that will play.

    d. Give an example (if possible) of an input that won't result in any sound being played.

       Any valid data would result in 'yeehaa.wav' being played. If the user enters a non-existent data path (e.g., C:/Temp/bogus.txt) or something else that the Describe method can't handle (e.g., 5 or 'foo') the Describe method will throw and error, so no sounds will be played, because the script will stop running before it reaches the sounds. Similarly, if the user doesn't enter an argument, the script will fail on line 21.
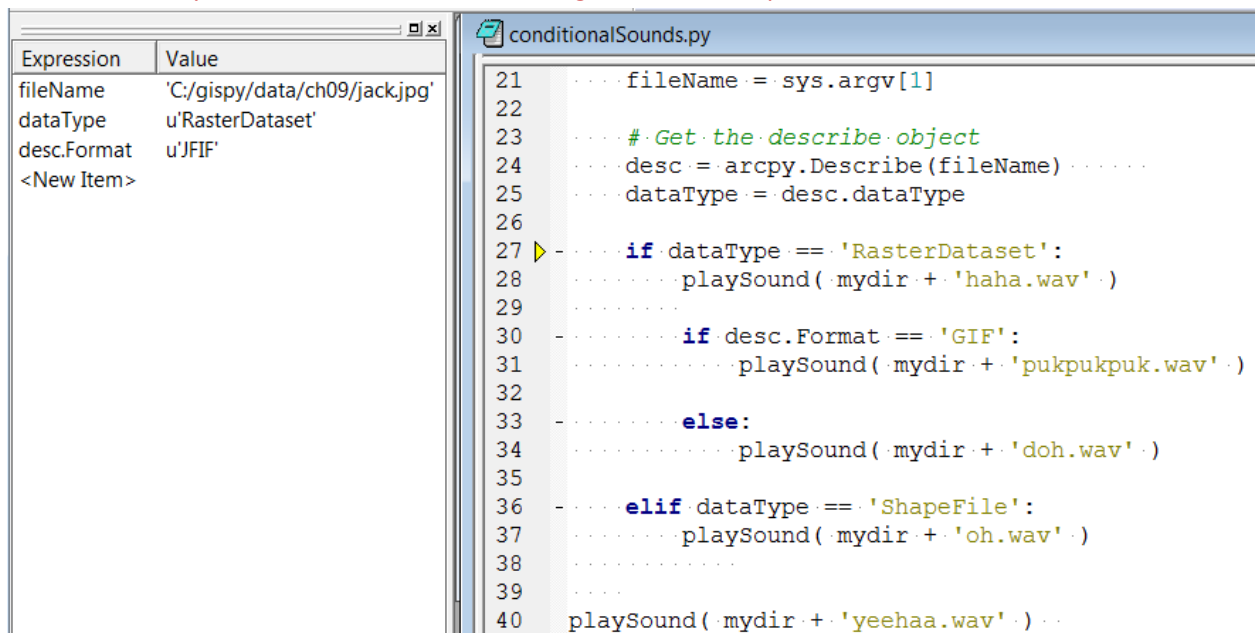
Expression	Value
fileName	'C:/gispy/data/ch09/jack.jpg'
dataType	u'RasterDataset'
desc.Format	u'JFIF'
<New Item>	

conditionalSounds.py

```
21        fileName = sys.argv[1]
22
23        # Get the describe object
24        desc = arcpy.Describe(fileName)
25        dataType = desc.dataType
26
27 ▷ -    if dataType == 'RasterDataset':
28            playSound( mydir + 'haha.wav' )
29
30 -        if desc.Format == 'GIF':
31                playSound( mydir + 'pukpukpuk.wav' )
32
33 -        else:
34                playSound( mydir + 'doh.wav' )
35
36 -    elif dataType == 'ShapeFile':
37            playSound( mydir + 'oh.wav' )
38
39
40    playSound( mydir + 'yeehaa.wav' )
```

Figure 2: PythonWin debugging session.

12. **(10pts)** Rewrite the workflow described below in terms of pseudocode. Use appropriate **structural components** (sequences, decision-making, and repetition), **key words** and **indentation**. Use each of the pseudocode key words, GET, SET, CALCULATE, FOR, IF, THEN, CALL, ENDIF, and ENDFOR, one or more times in your solution.

Get a workspace from the user and get a list of the files in that workspace. Then check the modification time of each file and delete any files that have not been modified for 5 years.

GET workspace from user

SET theDate to today's date.

CALCULATE threshold = theDate – 5 years

GET a list of files in the workspace

FOR each file in the file list

    GET modifiedDate, the date of when the current file was last modified

    IF modifiedDate is prior to the threshold THEN

        CALL the deletion tool to delete the current file

    ENDIF

ENDFOR

13. **(20pts)** The following script is meant to use a WHILE-loop to create 1 mile, 2 mile, 3 mile, 4 mile, and 5 mile buffer output files using the Buffer (Analysis) tool, but at the moment, it has 10 mistakes.   Line numbers with mistakes are given (some lines have 2 mistakes). CLEARLY show how to correct the mistakes *inside the code box.* Then, in parts 1)-10) below the box, explain what would happen if the mistake were not fixed.

1	import arcpy, sys
2	inputFile = ~~sys.argv[0]~~ sys.argv[1]
3	count = 1
4	while ~~count <= 6:~~ count < 6:
5	distance = ~~count + "miles"~~ str(count) + " miles"
6	output = ~~inputFile[:-4] + 'buff.shp'~~ inputFile[:-4] + str(count) + 'buff.shp'
7	~~arcpy.Buffer(inputFile, output, distance)~~ arcpy.Buffer_analysis(inputFile, output, distance)
8	print("Output created.")
9	→count = ~~count + 2~~ count + 1

1) **Line 1**   The sys module is used without being imported, so it would throw a NameError when you get to line 2 and try to use sys.

2) **Line 2**   sys.argv[0] would return the path of the script, not the first user argument.

3) **Line 3**   The iterating variable needs to be initialized. The script would throw a name error upon reaching line 4.

4) **Line 4**   The problem description specifies that the script should find 1,2,3,4, and 4 mile buffers. This script would also find a 6 mile buffer. (count<=5 would also work)

5) **Line 5**   The count is an integer, which needs to be cast to string to be concatenated with the string literal that represents the linear unit. Change it to the following: str(count)

6) **Line 5** The number and linear unit need to be separated by a space; otherwise, the buffer distance will be invalid (e.g., 2miles)

7) **Line 6**   The value of the output is variable not being modified. This would result in only one output file. Need to use the iterating variable in its name.

8) **Line 7**   The toolbox alias is missing. The problem description specifies that the script should be calling the Buffer tool in the Analysis toolbox, so you need to add the alias, _analysis to the tool call.

9) **Line 9**   This line should be indented the same level as line 8, otherwise lines 4-8 will keep looping infinitely.

10) **Line 9**   Count should be incremented by 1, not 2. count = count + 1