NR 426 – Programming for GIS I
Lab 1 – Python Basics

Warner College of Natural Resources | Colorado State University | Instructor: Elizabeth Tulanowski

This handout contains instructions for both parts of Lab 1 – Lab 1A and Lab 1B

Learning objectives for all of Lab 1:

1. Familiarize yourself with basic Python structure and syntax
2. Understand language constructs including:

`Variables`	`Data Types (strings, numbers, lists)`	`Loops`
`String formatting`	`Decision support statements (if)`	`Importing modules`
`Slicing`	`Working with lists`	`Math operations`
`print`	`concatenation`	`Running a script`
`comments`	`Working with Text files`	`Dictionaries`

3. Ability to write a basic script making use of simple strings, numbers, and lists
4. Describe the ways and places to write, save, and execute code
5. Navigate the Python help and documentation to find answers on your own

What to submit:

Use the **Lab 1A and Lab 1B template script files** to write code for each of the steps below. Submit the code from the in-class activities (demos) and the self-directed pieces to the appropriate assignment item in Canvas. The template script is set up to organize your code for each step, but add your own code and comments as needed.

Lab 1a: IDEs, Syntax, Strings, Numbers, Functions

Lab 1a In-class activities (done as a follow-along demo):

- Working with the help: How to find your own answers
  - Python Documentation
  - W3schools
  - geeksforgeeks

As a group, we will write the code to do the following tasks: *We'll do one at a time and discuss each before moving on.*

Use the Lab 1A template script from the N:\ drive Lesson 1 folder

1. Determine who has the longest name in the class. (Working with a function)
   - Just do your own name, then we'll compare
2. Create variables for these four objects: (Working with variables, math, and strings)
   i. Name
   ii. Year you moved here

iii. Current year
iv. City you live in

b. Use basic math to determine how old you were when you arrived in that city.
c. Create a formatted string with your above variables:
Sample Result: `Cam the Ram moved to Fort Collins in 2013, 13 years ago.`

- Review basics: comments, variables, math, string formatting, functions

Submit the above in your Lab 1a Python script

Lab 1a Self-directed activities (done on your own):

Continue using the Lab 1A template script file to write code that performs the following tasks:

Include comments as needed to explain what your code is doing.

3. Create two number variables and print their sum:
    a. Ie, create two variables, 5 and 4.
    b. Result: You'd want the number 9 to get printed
4. Convert a number of your choice from feet to meters using multiplication or division.

    Ie, Create a number variable that represents a value in feet. Convert to meters and print the result in a well formatted sentence

    Sample result: `5280 feet is equal to 1604 meters.`

5. Create a string variable and print with your own text to make a complete sentence
    a. Ie, create a variable "Park_Boundary.shp"
    b. Sample result: `The shapefile name is: Park_Boundary.shp`
6. Concatenate a number and a string variable using both str() and the f-string method
    a. Ie, Create a string variable for our course name and a number variable for the length of the class. Combine them into a sentence with a print statement using str()
    b. Sample result: `Our course Programming for GIS is 8 weeks long.`
    c. Repeat the task above, but now to build a print statement, but using an f-string

7. Work with index notation for a string
    a. Create a variable to represent a fictional dataset, for example: C:\Student\NR426\LandCover.shp (just make something up)
    b. Use index notation / slicing to parse out just the shapefile name without the file extension and return as a well formatted print statement
        i. Sample result: `The dataset name is: LandCover`
    c. Use index notation / slicing to parse out just the file extension and return as a print statement
        i. Sample result: `The file extension is .shp`
    d. Recall that -1 will let you index from the end
    e. Can you find a better approach to splitting the file name from the path? (Hint: look up os….)

8. Basic built-in functions (refer to the quick reference at the end of this lab handout, or search online)
    a. Use the appropriate function to determine and print out the *length* of 2 of your previously created variables, in a well formatted sentence.
        i. Sample result: `The file name has 21 characters`
    b. What *type* of object does the function you use above return? Use the correct function to return that information.
    c. Rounding
        i. Create a number variable with 4 decimals that represents a fictional area value for a polygon. (123.4567)
        ii. Write a print statement that dynamically reports the number, *rounded* to 2 decimal places (using the appropriate function), along with other text. Sample result:

            `The area of the open space is 123.46 acres`

9. Importing and using the os library

    The os library is quite useful for working with files. Look into the os library (Python doc) (GeeksforGeeks) samples and write the code to:

    a. Import the os library into the appropriate location in the script
    b. Create a variable representing a file path (ie, to your class folder)
    c. Create a second variable representing the name of a fictional dataset in that folder (like Mydata.shp- just make something up)
    d. Combine the path and file names into one complete string, **using the os library**
    Example:
    `Pathname = "C:\\Data\\NR426"`
    `Filename = "mydata.shp"`
    Result will a string variable that reads "C:\Data\NR426\mydata.shp"

Lab 1B: Lists, Looping, Decision-making, Error handling

Lab 1B In-class activities (done as a follow-along demo with Lesson 1B):

Use the Lab 1b template script from the N:\ drive Lesson 1 folder

1. Determine longest name in a list of names
    - Create a list of students' names (choose 5-10 names)
    - Loop through each name, determine the length of each
    - Determine the max length: can be done with an if statement or max function

In a geospatial context, imagine it's a list of numbers that represent area, year, or coordinates and you're looking for the largest feature, or the newest feature. (we'll do this in Lesson 4)

2. Explore the random module
    - Use the list from the above step
    - Randomly select 3 people as 1st, 2nd, and 3rd place "winners" and print out the result with complete sentences

In a geospatial context, you may need to randomly select a certain number of features for sampling.

- Submit the above in your Lab 1B Python script

Lab 1B self-directed activities (done on your own after Lesson 1B):

Continue using the Lab 1B template script to complete all of the below tasks:

3. Create a list variable with the following park names:

```
parklist = ["Horsetooth", "Lory", "Maxwell", "RMNP"]
```

Write the code necessary to determine the results of the following, and provide your answer and a description of the operation as comments in the .py that you submit:

Try to determine the answer manually first, and then check your result by writing and running the code.

Sample result for a) :

```
parklist = ["Horsetooth", "Lory", "Maxwell", "RMNP"]
#  3a - len
print("a)")
print(len(parklist))
#This retrieves the length of the list and would return 4, for four items
```

a) len(mylist)	e) mylist.index("Lory")
b) mylist[2]	f) mylist.pop(1)
c) mylist[1:]	g) mylist.append("Greyrock")
d) mylist[1]	h) mylist.sort(reverse = True) For this one, you can't print the result directly, run the sort, and print the list in another line of code to see what it did.

The items in the list will change as you go, this is ok! The goal is to understand how each list method works.

4. A - Write code to define a list of items and print out each one, creating output similar to the example below. Add additional text at the top, like a title. Choose any items you want; state names are just an example. *(Hint: Create a list, start a loop, and print each item within the loop).* Sample result:

```
The Four Corners States are:
Colorado
New Mexico
Utah
Arizona
```

B – Now add the length (# of characters) of each string in the loop. The result would now look something like this:

```
The Four Corners States are:
Colorado - 8
New Mexico - 10
Utah - 4
Arizona - 7
```

C - Use the correct list method to alphabetize your list, and re-do the loop and length step above.

> ***Thought question:***
>
> What would happen if you put the print statement for "The Four Corners States are:" inside the *for* loop?
>
> Test it out to see!

D - Programmatically add a numbering scheme to the print statement so it reads like this:

```
The Four Corners States are:

1. Colorado - 8
2. New Mexico - 10
3. Utah - 4
4. Arizona – 7
```

> *Hint: Don't manually type in 1. , 2., etc… With a long list of unknown length, that wouldn't be an option. We looked at 2 ways to do this in Lesson 1b.*

Lab 1B: *More* in-class activities (done as a demo with Lesson 1C):

Continue using the Lab 1B template script to write the code for the demo and self-directed activities:

5. Set up an if statement with multiple conditions. Choose something fun or silly, or work-related, ie:
   - If the Slope is greater than 30, suitability is 1, else if the Slope is between 15 and 30, suitability is 2, else if the Slope is less than 15 the suitability is 3…
   - If the temp is above 80, I'll wear shorts; or if the temp is between 60 and 80 wear short sleeves; or else if the temp is between 40 and 60 wear long pants; or else if the temp is below 40 wear a hat…

Add more to the code: comments, useful print statements, creating and passing in variables.

- Use a variable to supply the value to evaluate so it's easy to update, then *Run and re-run your code to test each different condition.*
- Modify to accept user input using the input() statement.

6. Dealing with errors: Reviewing proper syntax, training the eye
   - Using the if statement code from the above demo activity

        - Break it. Put 4-5 mistakes in the code (ie: capitalization, indents, colons, typos)
        - Swap with a neighbor and find the errors and fix the code.
    - Watch this great video on common Python errors
7. Look at code that reads in a txt file or CSV of names, and performing the same actions to determine longest name
    - Create a list of 5 objects and write out each item to a new text file. Embellish with additional lines of text as desired. Don't need to submit this part.

Submit the above as part of your Lab 1B script.

Lab 1B: *More* self-directed activities (done on your own, based on Lesson 1c slides):

Add the code to your Lab 1B script to perform the following tasks:

Include good comments to explain what your code is doing.

8. Manipulating strings to prepare file names:
   Write the code to parse and format a list of made-up shapefile names to represent what it will need to be called when imported into a geodatabase. The base name is the same, but you need to (1) add the folder path, (2) remove the .shp extension, and (3) capitalize the file name.
   Use these general steps:
   a. Create a list of fictional shapefile names, for example: roads.shp, streams.shp, boundary.shp, peaks.shp
   b. Create a string variable to represent the folder path to the fictional geodatabase
   c. Add additional text as desired and use comments to explain the code
   d. Loop through and manipulate the list items to provide a result that:
      a. Programmatically format the filename with a capital letter (Find a string method for this)
      b. Remove the .shp file extension, since it's not needed in a GDB. (Multiple options for this)
      c. Combines the file name to the GDB path

      Sample result:

      ```
      The output feature class names once imported will be:
      C:\Data\parkdata.gdb\Roads
      C:\Data\parkdata.gdb\Streams
      C:\Data\parkdata.gdb\Boundary
      C:\Data\parkdata.gdb\Peaks
      ```

9. Sometimes string sequences within longer ID values or descriptions can be used to identify features, or even generalize or reclassify data.
   For example, take these NLCD land cover categories: Deciduous Forest, Evergreen Forest, and Mixed Forest, Shrub/Scrub, Pasture/Hay, Open Water.

   Your task:
   - Create a string variable for a land cover type of your choice (Here's the full list of land cover types)

- write the code to determine if it contains the word “Forest” in the land cover name.
- Return a message to the user stating if it’s a forest type or not.
- In a comment, state how/where you found the function needed to accomplish the task

Try a couple of different land cover types to make sure your logic and code work as expected:
Sample results for two land cover types:

```
The shrub/scrub type is not a forest type
The Mixed Forest type is a forest type
```

In more of a geospatial context, you may be looping through land cover values in a table, and returning “forest” into another field in the table, to generalize the data. In this exercise, we’re just looking at single land cover values, not through a table.
Or more generally, you may need to find out if a file name or attribute value contains a certain word or code, to find a feature, or identify errors in the data

10. Working with filenames and paths and os

a. Create a string variable for your Lesson 1 folder (or any folder of your choice)
b. Write the code to determine if a file or folder **exists** or not, by creating a string variable for a file/path name and using the appropriate **os module** method and an if statement.
   (1) You may need to look up the right method to use!
   (2) Change your variable value to test your logic
c. Return the result in a well formatted sentence.

Quick references for basic Python commands and functions:

A quick reference to some basic Python string commands:

Adapted from Python Scripting for ArcGIS by Paul Zandbergen, Exercise 1		
Type:	**Result**	**Notes:**
`print ("Hello World")`		The code prints Hello World to the next line. This is called a string, as in a string of characters. Strings are values, just as numbers are.
`print ('Let's go!')`		The code returns a syntax error because Python does not know how to distinguish the quotation marks that mark the beginning and end of the string from the quotation marks that are part of the string — in this case, in the word "Let's." The solution is to mix the type of quotation marks used, with single and double quotation marks.
`print ("Let's go!")`		The code successfully prints Let's go!
`z = "Alphabet Soup"` `print (z[7])`	T	The code returns the letter t, the seventh letter in the string where the letter A is located at index number 0. This system of numbering a sequence of characters in a string is called indexing and can be used to fetch any element within the string. The index number of the first element is 0.
`print (z[0])`	A	The code returns the letter A.
`print (z[-1])`	P	The code returns the letter p. The index number of the last element depends on the length of the string itself. Instead of determining the length, negative index numbers can be used to count from the end backward.
`print (z[0:8])`	Alphabet	To fetch more than one element, use multiple index numbers. This is known as slicing. z[0:8] returns the characters with index numbers from 0 up to, but not including, 8, and therefore the result is Alphabet.
`name = "Geography"` `name.find ("graph")`	3	The result is 3, the index of the letter g. In this example, find is a method that you can use on any string.
`string.upper()` `string.lower()`		Force all letters to uppercase, lower, etc… Documentation on string operations

`x="Cam The Ram"`
`y = x.split(" ")`
`print (y)`
	["Cam", "the", Ram"]	Create a string
		Split the string every where there is a space
		Returns a list of each piece parsed from the original

A quick reference for working with Numbers in Python:

Type this:	**Result?**	**Notes:**
`12 + 17`	29	Basic calculator functions work just as you would expect: `+ - * /`
`10 / 3`	3	Division: If the inputs are both integers, the result is, by default, also an integer. This results in rounding. If you want ordinary division, the solution is to use real numbers or floats — that is, numbers with decimals. If either one of the inputs in a division is a float, the result will also be a float.
`10.0 / 3.0`	3.3333333333333335	The very last number does not make sense because it has reached the limit of the number of decimal places Python uses.

		Ordinary integers cannot be larger than 2147483647 or smaller than -2147483647. However, if you want larger values, you can use a long integer, commonly referred to as "long."
12345678901	12345678901L	The letter L at the end indicates that Python converted the input value to a long integer.
2 ** 5	32	Exponentiation, or power, operator (**). Can also be accomplished with the pow method.

More about variables:

Type:	Result	Notes:
x = 12 print (x)	12	The value of 12 is prints to the next line. The code line x = 12 is called an assignment. The value of 12 is assigned to the variable x. Another way of putting this is to say that the variable x is bound to the value of 12. X is dynamically typed as an integer, you don't need to declare it explicitly
x = 12 y = x / 4 print (y0	3	Variables can be used directly in mathematical or string operations
X = "text" Y = 'text' Z = '''Use three to span multiple lines'''		4 ways to make a string: 1. Single quotation marks 2. Double quotation marks 3. Three single quotation marks (can span multiple lines) 4. Three double quotation marks (can span multiple lines)
Name = "Cam the Ram" print (name)	Error: name not defined	Variable names are case sensitive The variable was called Name (uppercase), but the print statement asked for name (lowercase).
path = "c:/data/part1/final" pathlist = path.split("/") lastpath = pathlist[-1] print (lastpath)	Final	In the first line of code, the path is assigned as a string to the variable path. In the second line of code, the string is split into four strings, which are assigned to the list variable pathlist. And in the third line of code, the last string in the list with index -1 is assigned to the string variable lastpath.

A quick reference for working with functions and modules:

Type this:	Result	Notes

len("Cam the Ram")	11	Returns the length of an object
		String = # characters
		List = # items in the list

d = pow (2, 3) print (d)		Instead of using the exponentiation operator (**), you can use the pow function. You supply the function with parameters, or arguments (in this case, 2 and 3), and it returns a value.
print (dir(__builtins__))		View the complete list of built-in functions Note the double underscores around the word __builtins__
help(<function name>		Access help /documentation for a specific function
type(<variable or object> Type(40)	int	Returns the type for an object: int, str, etc…
range(10, 21, 2)	[10, 12, 14, 16, 18, 20]	The function returns a list of integers from 10 to 20, with an increment of 2. However, the only required parameter is the endpoint (stop).

`import math`		Gain access to modules using import, and then use the functions from that module by writing <module>.<function>.
`print (dir(math))`		You can obtain a list of all the functions in the math module using the dir statement.
`print (math.floor.__doc__)`		The result is a printout of the same syntax from Help but now directly within the Python console window.

`import random`
`j = random.uniform(0, 100)`
`print (j)`
	The result is a float from 0 to 100.	Imports the random module, and generates a variable j which is a random floating point number between 0 and 100

`import random`
`p = random.randint(1, 6)`
`print (p)`
		Generates a random number between 1 and 6

A quick reference on list methods

Adapted from Python Scripting for ArcGIS by Paul Zandbergen, Exercise 1

Type this:	**Result**	**Notes:**
`w = ["Apple", "Banana", "Grape", "Kiwi"]` `print (w)`	This prints the contents of the list.	Printing a list object directly does not format it very well. It's fine for development, but not for a final script or messaging to a user
`print (w[0])`	Apple	This returns Apple because the index number of the first element in the list is 0.
`print (w[-1])`	Kiwi	You can use negative numbers for index positions on the right side (the end) of the list.
`print (w[1:-1])`	['Banana','Grape']	Returns the elements from index number 2 up to, but not including, -1, and therefore the result is ['Cantaloupe', 'Durian'].
`w.append("Orange")` `print (w)`	["Apple", "Banana", "Grape", "Kiwi", "Orange"]	Modify lists with various list methods: https://docs.python.org/3/tutorial/datastructures.html
`x = 1` `x = x + 1`	X now equals 2	This trick can be used within a loop to increment a number, like a counter, or to create sequential numbers

Also check out: https://docs.python.org/3.6/library/stdtypes.html#sequence-types-list-tuple-range

Rev. 1/10/26