Python syntax and structure

Relevant Reading:

- Python Scripting for ArcGIS Pro: Ch. 4
- Python for ArcGIS Pro: Ch. 1

YouTube Playlist for this (and all) lessons:

https://www.youtube.com/playlist?list=PLDX1Ztgl1MQjvYdHP7iT6cPNpISTE6Hgy

NR426
Programming for GIS I
Lesson 1 (a, b, c)

Instructor:

Elizabeth Tulanowski | ESS Department

What does Lesson 1 cover?

- Three class sessions: Lessons 1a, 1b, 1c
- Navigating the help and learning to find your own answers
- Python language structure and syntax
- What are, and how to create, variables
- Looping and decision making
- Saving and running script tools

Lesson 1a – Part 1

- **Python help resources**
- **Basic Python rules**
- **Working with Python objects:**
  - **Numbers and Strings**

Part 2:

- Modules
- Functions
- Saving and executing code

Lesson 1a Videos:

Part 1

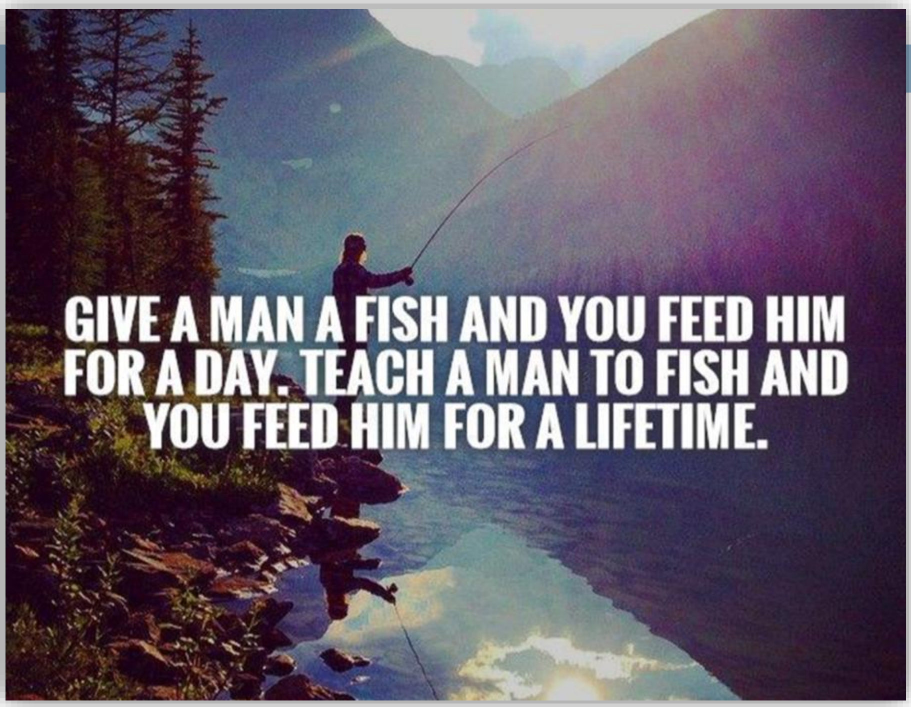
Part 2

Demo

PSA book Ch. 4.2, p. 84

Getting Help

- Python Beginners Guide
- Object-oriented programming concepts
- Getting Started
- Python 3 Tutorial
- Python 3 documentation
  - Python language reference – exact syntax
  - Python standard library – general descriptions of language components
- ArcGIS Pro Python help
- W3schools geeksforgeeks realpython
- Google, gis.stackexchange.com


GIVE A MAN A FISH AND YOU FEED HIM FOR A DAY. TEACH A MAN TO FISH AND YOU FEED HIM FOR A LIFETIME.

A quick look at simple code

```
#Import modules
import os, math

#Set up variables
path =  r"C:\Data\Colorado\LarimerCo"
shp = "FC_boundarynew.shp"
fieldname = "Area"
acres = 5

#Convert values from acres to sq miles
sqmi = acres / 640
print (sqmi)
print ("The area in sqmi is " + str(sqmi))
mi = math.sqrt(sqmi)
print ("Miles = " + str(mi))

# Parse out the pieces of the file name we need
print (shp[:5])
print(shp[-4:])
print (shp[:-4])

# Combine the path with the file name
fullpath = os.path.join(path, shp)
print (fullpath)

# Return the length of the string
print ("The length of the pathname is: " + str(len(path)))
len(fullpath)
```

Notice:

- **# Comments**
- **Case sensitivity**
- **Slashes**
- **Quotation marks**
- **Variables**
- **r before paths**

Python: Rules of the Road (overview)

# Comments

Case sensitivity

Slashes

Quotation marks

Variables

r before paths

PSA book
Ch. 4.27, p.145

```
# Comments are lines of code not executed – like notes


# Variables defined with =
# Put text, strings in quotation marks


#Set up variables
path =  r"C:\Data\Colorado\LarimerCo"
shp = "FC_boundarynew.shp"
fieldname = "Area"
acres = 5
```

Python: Rules of the Road (digging deeper)

- # Comments
- **Case sensitivity**
- Slashes
- Quotation marks
- Variables
- r before paths

PSA book
Ch. 4.27, p.145

- Case sensitivity
  - *What is case sensitive:* Python statements, functions, variables: `import, print, if,  for`
    - arcpy commands, functions, methods, tool names
    - Some describe properties (FeatureClass, Polygon) and SQL syntax are case sensitive
  - *What is NOT case sensitive:*

    Text inside of a string, ie, path names

Tip:
Pay attention
Be consistent

Python: Rules of the Road (digging deeper)

- # Comments
- Case sensitivity
- **Slashes**
- **Quotation marks**
- Variables
- r before paths
- PSA book
  Ch. 4.27, p.145
  Ch. 4.7, p. 93
  4.19, p. 127

- Slashes: / vs. \\
  - Use either 1-forward or 2-back slashes in path names, or "r" for raw:

```
path =  r"C:\Data\Colorado\LarimerCo"
```
← Recommended
```
path =  "C:\\Data\\Colorado\\LarimerCo"
```

  A single backslash is an escape character:
    - Usually followed by another character to make a newline, tab etc…
    - More info here
    - Also used for line continuation or in complicated strings like SQL clauses

- Quotation marks: “ “ **or ‘ ‘ ?**
  - You must open and close a string with the same “ or ‘
  - SQL query expressions have specific requirements for quotation marks

Variables and paths

- # Comments
- Case sensitivity
- Slashes
- Quotation marks
- **Variables**
- **r before paths**

PSA book Ch. 4.5, p. 89

- A placeholder that represents data or values (objects)
  - Input values, data paths, numbers, lists, files etc…
- *Dynamically typed.* Create using the =

  -
    ```
    shp = "FC_boundarynew.shp"
    acres = 5
    SqMi = .0078
    ```

Naming conventions *(from PSA book, 4.28)*

- Start with a character, avoid special characters (ie: * - ~ . Etc…).
- Use all lowercase, such as mycount. (follows Python style guide)
  - CamelCase – 1st letter of each word capitalized. *Ie, RoadsBuffer.shp* Common, more readable
- Underscores (_) can be used for readability
  - Ie: count_final or Roads_UTM
- Be descriptive yet concise
- Avoid slang terms or abbreviations

PSA book Ch. 4.4, p. 85

What can you do with numbers?

- Numbers: Goals for Lesson 1
  - Creating number variables
  - Simple math
  - Math module
  - Concatenating strings to numbers

DEMO

```
# Lesson 1A Part 1 Numbers Demo

# Add
print (5 + 7)

# Make integer variable
elev = 5280
# Make float variable
conv = .3048
conv2 = 3.28

# Do basic math
print (elev * conv) # multiply elev by conv
print (elev / conv2 )  # divide elev by conv2

# Will this work?
print (“Elevation " + elev + " in meters: " + elev*conv)

# Round to 1 decimal place
round(conv2, 1)
```

Learn more about numbers in this video

Strings example

- Strings: Goals for Lesson 1
  - Creating string variables
  - Basic manipulation and formatting (Help)
  - Concatenating and slicing strings
  - Getting length
  - Split

DEMO

More on strings:
PSA book Ch. 4.12-13 and this video

```
# Strings demo
City = "Fort Collins"
State = "colorado"
county = "LARIMER"

print (City + ", " + State)                            → Fort Collins, colorado
location = City.title() + ", " + State.title()
print (location)                                       → Fort Collins, Colorado

print (county.lower())                                 → larimer
print (county.title())                                 → Larimer

# Basic string operations
#length - number of characters
print (len(county))                                    → 7

# indexing - get the first two characters
print (State[0:2])                                     → co
print (State[0:2].upper())  # Make it all caps         → CO

#split - split a string at a delimiter
Citywords = City.split(" ")
first = Citywords[0]
second = Citywords[1]
print (first)                                          → Fort
print (second)                                         → Collins
```

Concatenating

Concatenate = combining

Concatenate text to text or text to numbers using one of a few methods

- "text" + "text" → string *concatenation*

  `"roads" + "_clp"  → "roads_clp"`
- number + number → *addition*

  `2 + 5 → 7`
- "text" + number → *error*

  ```
  "roads" + 2 →
    TypeError: cannot concatenate
  'str' and 'int' objects
  ```
- "text" + str(number) → concatenating a string and a number

  `"roads_" + str(2)  → "roads_2"`
- **Alternative: f-strings**

  See Method 2 on the right

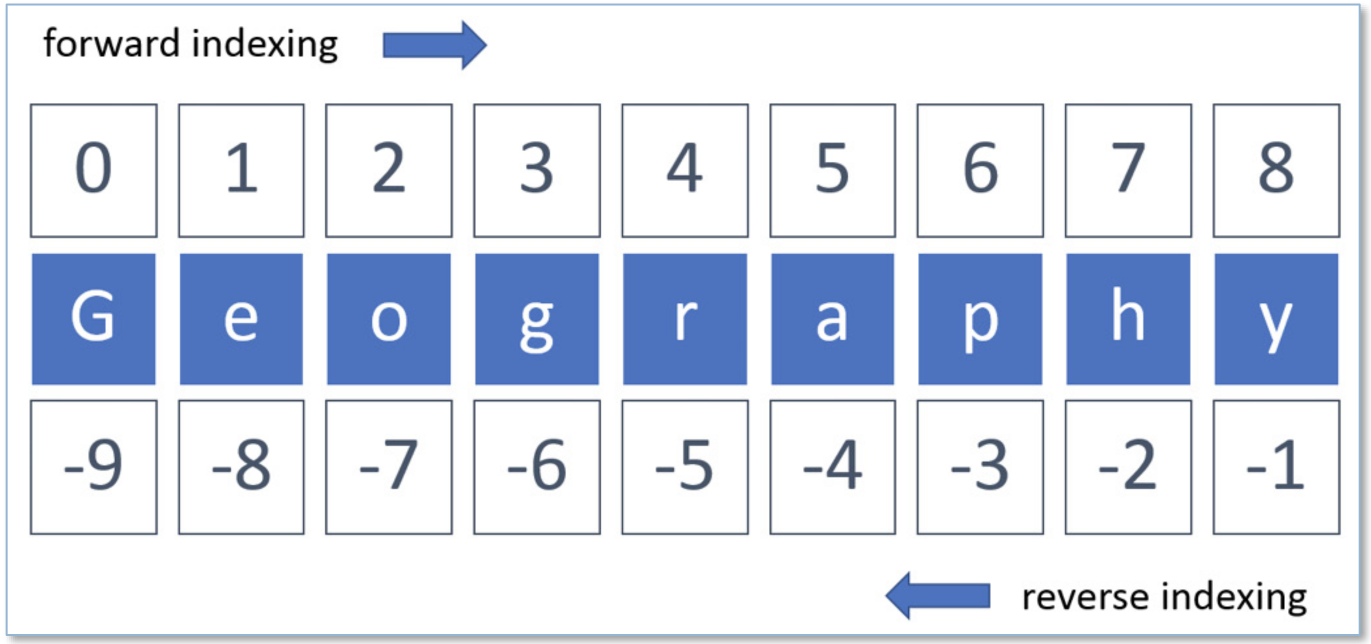
```
folder = r"C:\Data\Python"
path =  "roads.shp"
length = len(path)

# Method 1:  concatenate with +
msg = "The full path is " + folder + "\\" + path + " and has " /
         + str(length) + " characters"
print (msg)
#or use os to join the folder to the path
import os
msg2 = "The full path is " + os.path.join(folder,path) + " and has "/
         + str(length) + " characters"
print (msg2)

# Method 2:  f-string
fullpath = os.path.join(folder, path)
msg4 = f"The full path is {fullpath} and has {length} characters"
print ("\nMethod #3:")
print (msg4)


# Method 3:  .format and {}
msg3 = "The full path is {0} and has {1} characters".format(os.path.join/
         (folder, path), length)
print (msg3)

#Method 4:  % as placeholder
msg5 = "The full path is %s and has %i characters"%(fullpath, length)
print ("\nMethod #4:")
print (msg4)
```

Indexing/slicing for strings *(also lists and tuples)*

- Access characters or ranges of characters with index numbers
  - Also works for items in a list or tuple
- 0-based from the left | -1 based from the right
- Ranges use a colon “Up to but not including the number after the colon “

```
>>> text = "Geography"
>>> text[5]
'a'
>>> text[-4]
'a’
>>> text[0:3]
'Geo’
>>> text[:3]
'Geo’
```

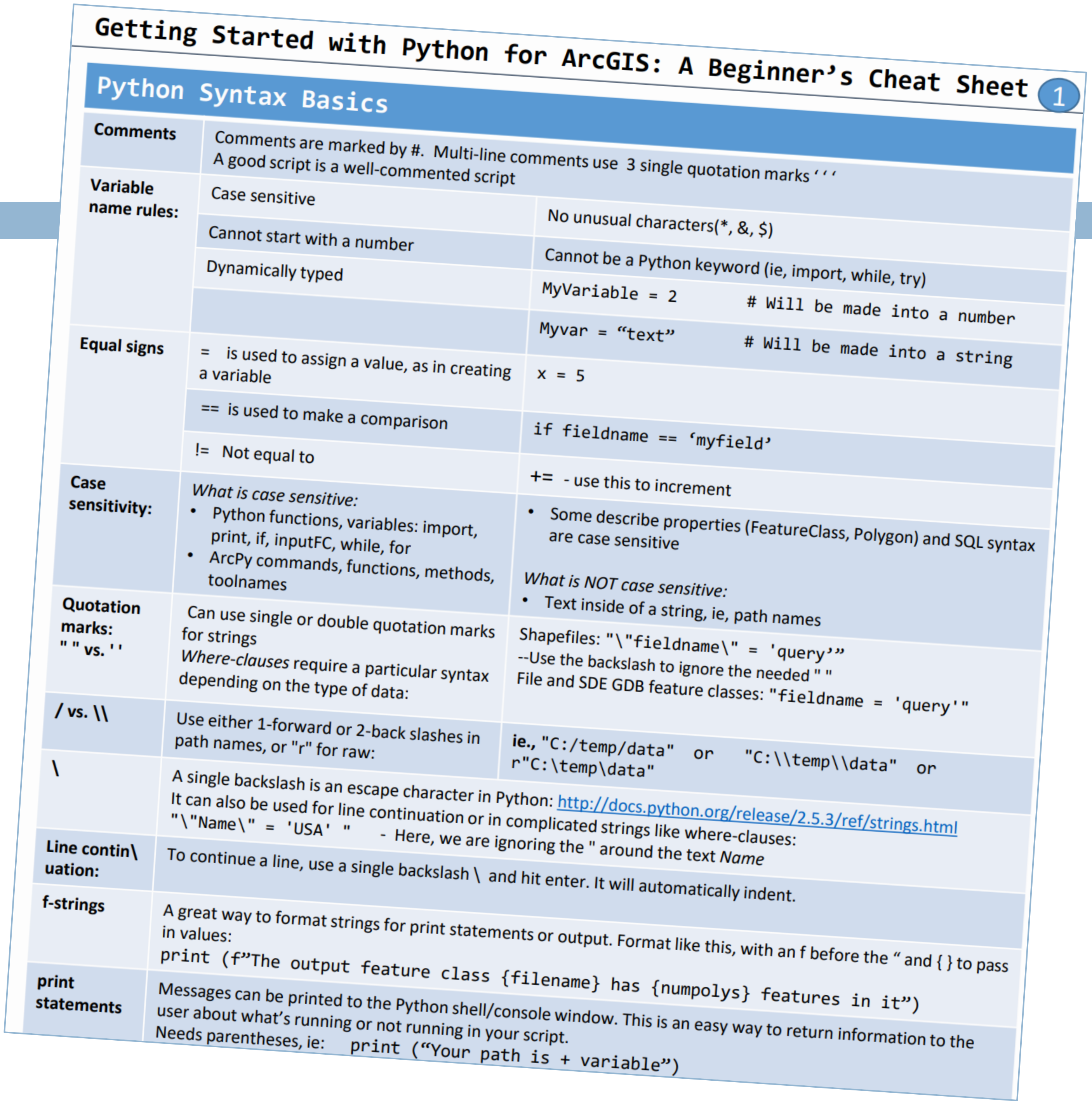
```
>>> text[-1]
‘y’
>>> text[0:-1]
'Geograph’
>>> NewName = text[:3]
>>> print (NewName)
Geo
```



So many rules!

- Refer to the

[Python for ArcGIS: A Beginner's Cheat sheet]



Getting Started with Python for ArcGIS: A Beginner's Cheat Sheet 1

Python Syntax Basics

Comments	Comments are marked by #. Multi-line comments use 3 single quotation marks ''' A good script is a well-commented script	
Variable name rules:	Case sensitive	No unusual characters(*, &, $)
	Cannot start with a number	Cannot be a Python keyword (ie, import, while, try)
	Dynamically typed	`MyVariable = 2        # Will be made into a number`
		`Myvar = "text"        # Will be made into a string`
Equal signs	= is used to assign a value, as in creating a variable	`x = 5`
	== is used to make a comparison	`if fieldname == 'myfield'`
	!= Not equal to	+= - use this to increment
Case sensitivity:	*What is case sensitive:* • Python functions, variables: import, print, if, inputFC, while, for • ArcPy commands, functions, methods, toolnames	• Some describe properties (FeatureClass, Polygon) and SQL syntax are case sensitive *What is NOT case sensitive:* • Text inside of a string, ie, path names
Quotation marks: " " vs. ' '	Can use single or double quotation marks for strings *Where-clauses* require a particular syntax depending on the type of data:	Shapefiles: `"\"fieldname\" = 'query'"` --Use the backslash to ignore the needed " " File and SDE GDB feature classes: `"fieldname = 'query'"`
**/ vs. **	Use either 1-forward or 2-back slashes in path names, or "r" for raw:	**ie.,** `"C:/temp/data"` or `"C:\\temp\\data"` or `r"C:\temp\data"`
\	A single backslash is an escape character in Python: http://docs.python.org/release/2.5.3/ref/strings.html It can also be used for line continuation or in complicated strings like where-clauses: `"\"Name\" = 'USA' "` - Here, we are ignoring the " around the text *Name*	
Line contin\ uation:	To continue a line, use a single backslash \ and hit enter. It will automatically indent.	
f-strings	A great way to format strings for print statements or output. Format like this, with an f before the " and { } to pass in values: `print (f"The output feature class {filename} has {numpolys} features in it")`	
print statements	Messages can be printed to the Python shell/console window. This is an easy way to return information to the user about what's running or not running in your script. Needs parentheses, ie: `print ("Your path is + variable")`	

NR426

Programming for GIS I

Lesson 1 (a, b, c)

Instructor:

Elizabeth Tulanowski | ESS Department

Python syntax and structure

Relevant Reading:

- Python Scripting for ArcGIS Pro: Ch. 4
- Python for ArcGIS Pro: Ch. 1

YouTube Playlist for this (and all) lessons:

https://www.youtube.com/playlist?list=PLDX1ZtgI1MQjvYdHP7iT6cPNpISTE6Hgy

Lesson 1a – Part 2

- Some programming basics
  - Objects, Methods, Functions, Modules
- Python help resources
- Commenting code
- Working with Python objects:
  - Numbers and Strings
- **Modules**
- **Functions**
- **Saving and executing code**

PSA book Ch. 4.20, p.130

Import

- Python is modular
- Import only those "modules" that contain the functionality you need in your script.
- Some typical modules you may find necessary or useful:
  - arcpy, os, sys, traceback, math, datetime and time, zipfile, pandas
    - Open source geospatial libraries: geopandas, rasterio, earthpy, pysheds........
  - Example:

```
import os, sys
from arcpy import sa
```

- For help/learning more: Python module index

Learn more in this video

PSA book Ch. 4.10, p. 99

Built-in functions

- Python functions that do not require anything to be imported http://docs.python.org/3/library/functions.html
- Useful examples:
  - `len()` – returns the length, the numbers of characters in a string
  - `str()` – converts numbers to strings so they can be concatenated
  - `int()` – converts floating point numbers to integer
  - `round()` – rounds values
  - `print()` – prints text to the console window
  - `max()` – returns largest item in an iterable (list) (also `min`)
  - `type()` – returns the type of the object

    (ie, int, string, list, etc…)

Function vs. Method

- Functions and methods are both tasks that perform operations
- A method is tied to an object and can call arguments
  - Object.method()
- A function has arguments which can declare the object to use
  - Function(object)

Would return:

```
name = "cam the Ram"

print (type(name))   # Function      →   <class 'str’>

print (len(name))   # Function       →   11

print (name.upper())   # Method      →   CAM THE RAM

print (name.title())   # Method      →   Cam The Ram
```

Saving and executing code

- Standalone scripts
  - .py files run from Python IDE (or a command prompt), not from ArcGIS
  - Code follows best practices for header, variables, comments, body, readability
- Command line
  - Run code one line at a time (small tasks)
- Script tools in ArcGIS
  - Scripts can be written with user parameters to be run from ArcToolbox as a dialog box
  - Script must be added as a tool to a custom toolbox and contain the proper code to accept user inputs
- Notebooks (ie, Jupyter Notebooks)

What did you just learn?

1. Name 2 places to write and execute Python code
2. Name another place you can write, but couldn’t execute, Python code
3. Name 4 types of Python objects
4. Name and describe 4 built-in functions

What did you just learn?

6. Why do we care about slashes? / \ \\
7. What does it mean for variables to be *dynamically typed*?
8. Explain the use of quotation marks in Python.
9. Explain the difference with using + for numbers vs. strings

What did you just learn?

Name the function that converted `inStr` to each `var#`

`inStr = "MiNnEaPoLiS"`	
`var1 = "MINNEAPOLIS"`	`upper()`
`var2 = "Minneapolis"`	`capitalize()`
`var3 = "minneapolis"`	`lower()`
`var4 = "MiNnE"`	`split("a")[0]  or inStr[0:5]`

`var5 = "Minnesota"`	`lower().replace("apolis","sota").capitalize()`
	`Or inStr[0:5] + "sota"`

Lesson 1A: Putting it together

- Review options for IDEs
  - Full descriptions and walk-throughs in PSA Exercise 2, posted to Canvas
- Follow-along demo
  - Writing a simple script to:
    - Create variables for string and numbers
      - Create variables to represent folder paths
    - Format strings
      - Use index notation to parse certain characters
    - Do basic math
    - Import common modules
    - Implement common functions like len, str, print, type

Watch this
Demo Video

Start Lab 1a: Python basics

- Lab 1 has 2 parts:
  - Lab 1a: based on Lesson 1a, Python basics, syntax, variables, functions, finding help
  - Lab 1b: based on Lesson 1 b-c, Lists, looping, if, input
  - See Canvas for due dates
- Use knowledge from today's lecture + demo + resources
  - Submit your code for each part + the demo

Need more guidance?

Supplement with Python Scripting for ArcGIS Pro book, Exercise 4

Lesson 1b

- Working with lists and dictionaries
- Using classes and objects
- Looping statements (for, while)

- Continue Lab 1

Watch the
Lesson 1b videos:
Lecture/Slides
Demo

PSA book Ch. 4.8, 4.15

Lists

- A list is an ordered collection of objects stored in memory and referenced by a variable name. Typically the items are the same type.
  - 2 ways to create:

```
myList1 = []    #Creates an empty list object
mylist2 = [1, 2, 3, 4]  or    myList3 = [“a”, “b”, “c”]
```

- Lists are “mutable” or changeable, and can have values added and removed, sorted.
- Lists are indexed (Just like strings)
  - Use index notation (slicing) to access individual items or ranges of items

```
myList3[1]    → “b”
myList2[0:2]    → [1,2]
```

Tuples: another type of collection
Indicated by ( ) and Immutable
More info on tuples

Learn more about lists in this video

More on lists

- You can loop through the items in a list using a *for* loop

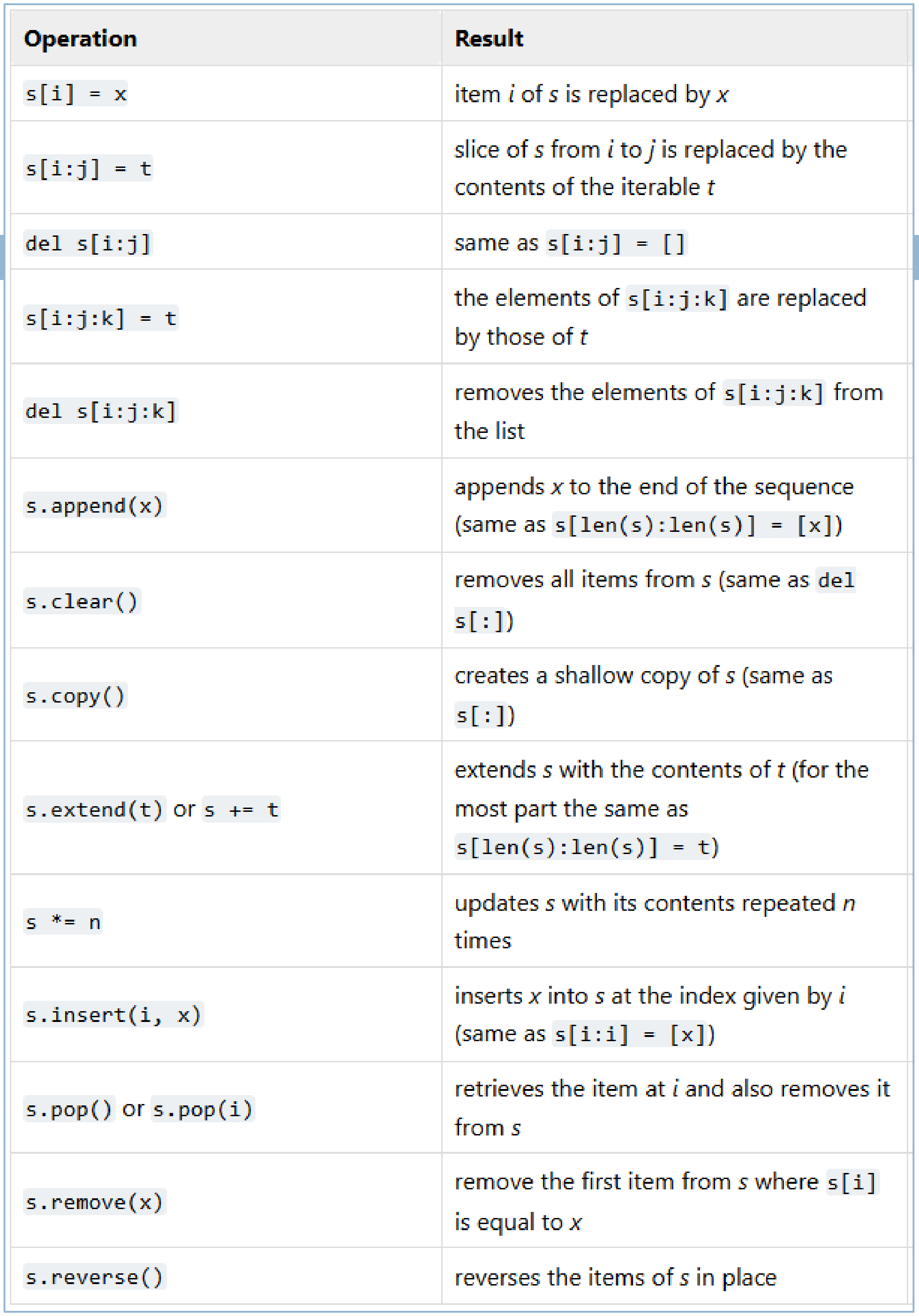
```
for x in myList3:
       print (x)
```

- Lists can be altered:

```
myList3.append[“d”]
print (myList3)  → [“a” “b” “c” “d”]
```

PSA book Ch. 4.15

- More info on lists
  - Sections on Common sequence operations and sequence types –
  - .count ; .sort ; methods in graphic



Operation	Result
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as `s[i:j] = []`
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)
`s.clear()`	removes all items from *s* (same as `del s[:]`)
`s.copy()`	creates a shallow copy of *s* (same as `s[:]`)
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)
`s *= n`	updates *s* with its contents repeated *n* times
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)
`s.pop()` or `s.pop(i)`	retrieves the item at *i* and also removes it from *s*
`s.remove(x)`	remove the first item from *s* where `s[i]` is equal to *x*
`s.reverse()`	reverses the items of *s* in place

PSA book Ch. 4.22, p.135

Indentation

- Critical to Python - defines blocks of code
  - for and while loops, if, try-except, def, other “compound statements”

```
for x in myList:
    print (x)
print (“done with the list”)
```

- Be consistent with the amount of indentation
  - Usually a tab

Loops (for)

PSA book Ch. 4.23, p.136

Learn more about loops in this video

`for`

- Used for looping through lists.
- Performs operation(s) for each item in the list, and automatically iterates to the next item until the list is done.

```
for x in parkList:
     print (x)
```

Numbering a list with a counter: (don't do it brute-force!)

```
Count = 1
for x in parkList:
     print (f"{Count}. {x}")
     Count += 1  # same as Count = Count + 1
```

Another way to increment is with **.enumerate()**

List comprehension

- A more compact way to iterate with lists
  - Regular **for** loop:

```
for <item> in <list>:
  <expression>
```

```
squares = []
for x in range(6):
    squares.append(x * x)
print(squares)
```

  - List comprehension:

```
[<expression> for <item> in <list>]
```

```
squares = [x * x for x in range(6)]
print(squares)
```

Another example, with spatial data:

```
fclist = ["roads", "streams", "parks", "buildings"]
paths = [f"C:\\data.gdb\\{fc}" for fc in fclist]
print(*paths, sep="\n")
```

```
C:\data.gdb\roads
C:\data.gdb\streams
C:\data.gdb\parks
C:\data.gdb\buildings
```

PSA book Ch. 4.23, p.136

Loops (while)

`while`

- Use for looping as long as a condition is met.
- While something is true, while something exists, while there's a row in the table to look at. Sometimes uses the next() method to iterate to the next item.

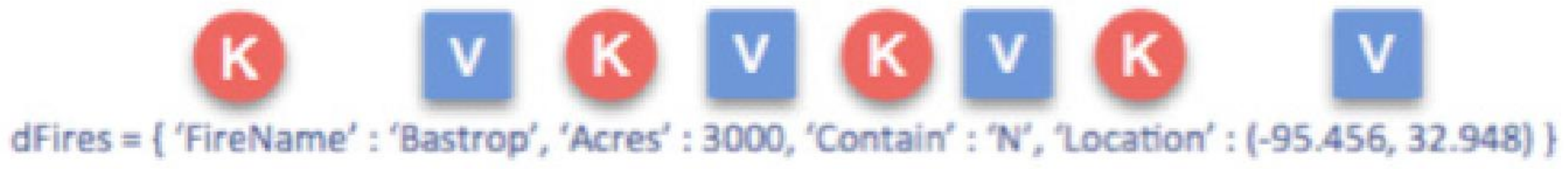
```
num = 1
while num < 5:
    print (num)
    num = num + 1    # or num += 1
print ("We made it to 5")
```

- What would happen if you don't have the `num = num + 1`?
- A break statement can be used to exit out of a while loop in the middle.
- More info on `while` from geeksforgeeks

PSA book Ch. 4.17, p. 122

Dictionaries

- Unordered collection of objects
- Stored and retrieved as key-value pairs
- *Think*: State:Capital or X:40.5



```
dFires = { 'FireName' : 'Bastrop', 'Acres' : 3000, 'Contain' : 'N', 'Location' : (-95.456, 32.948) }
```

- Instantiated with { }
- Referenced by the key
- Additional resources:
  - From Real Python or geeksforgeeks
  - This video

Putting it together

- Follow-along demo
- Writing a simple script to:
  - Build off Lesson 1a skills
  - Create a list of names, modify it
  - Sequentially number and print out a list of items
  - Use .split() to create a list from a delimited string
    - Practice with indexing and slicing

Watch this

Demo Video

Lists vs. Dictionaries

Answers at the end of the section

	List	Dictionary
Ordered collection		
Access by		
How to initialize		
How to determine number of items		
How to retrieve values		
How to add data to		

Will this work?

- 1.

```
Filename = “landcov.shp”
Basename = filename[:-4]
```

- 2.

```
area = 34.3
print “The total area of water is + area”
```

- 3.

```
word = “Thisisareallylongstring”
print word.len()
```

What did you just learn?

1. What are some key similarities between lists and tuples?
2. What are some key similarities between lists and dictionaries?
3. Describe the two main looping structures in Python
4. Which approaches can be used to break up long lines of code for better readability?
5. Why do we add comments to our code?

Lists vs. Dictionaries *(Answers)*

	List	Dictionary
Ordered collection	Yes	No
Access by	Index number	Key `dict.key` or dict.get(key)
How to initialize	[ ]	{ }
How to determine number of items	len	len
How to retrieve values	print(list) or better, use for loop	Get list of key-value pairs with dict.items() Use in for loop
How to add data to	.append(), .extend(), .insert()	`dict[newkey] = ‘newvalue’`

Continue Lab 1: Python basics

- Lab 1b today
  - Lists, loops, dictionaries
- Use knowledge from today's lecture + demo + resources

Lesson 1c

- Decision-making with if
- Error handling (Briefly)
- Working with files: text files, zip files
- Review and recap

- Finish Lab 1

Watch the
Lesson 1c videos:
Lecture/Slides
Demo
Lesson Review/Conclusion

PSA book Ch. 4.21, p.132

Decision Support Statements (if)

`if`

Used for decision-making. If a criteria is met, do something. Otherwise (else) do something else

```
if fc == “roads”:
    arcpy.Buffer_analysis(fc, fc+”_clp”, “1 mile”)
else:
    print “That’s not the feature class we want to buffer”
```

Learn more in this video

PSA book Ch. 7

Simple error handling

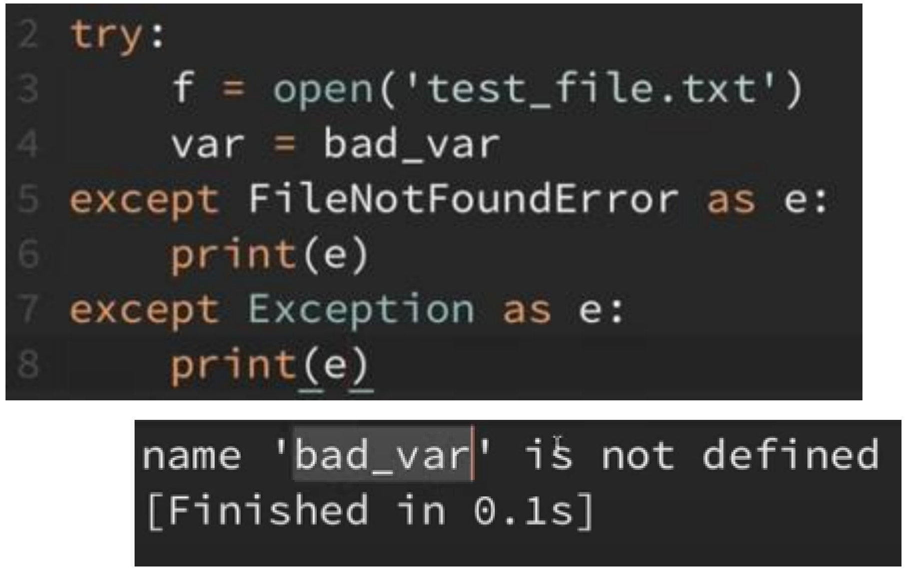
The try – except block

- One of a few ways you can anticipate and handle errors in a script.
- *Try to do something, and if anything goes wrong, raise an exception.*
- Allows the script to continue running, and even think it executed successfully, even if a step failed.

```
try:
        arcpy.Buffer_analysis (input, output, 2 miles)
except:
        print arcpy.GetMessages()
```

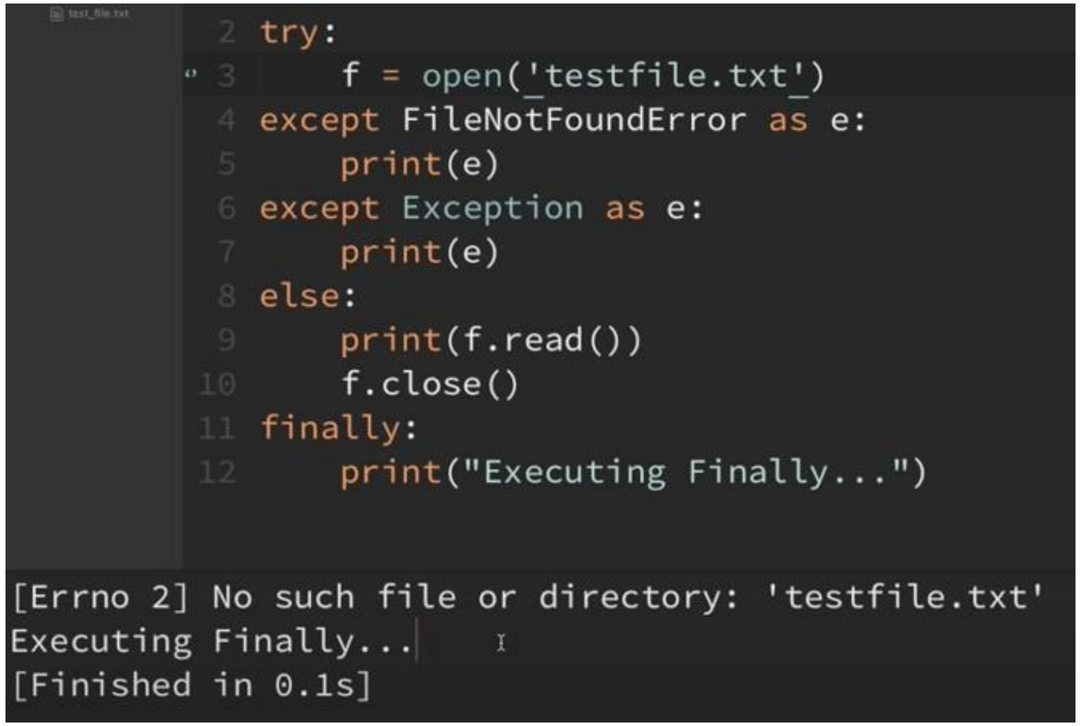
Learn more in this video

A little more to the try-except



```
try:
    f = open('test_file.txt')
    var = bad_var
except FileNotFoundError as e:
    print(e)
except Exception as e:
    print(e)
```

```
name 'bad_var' is not defined
[Finished in 0.1s]
```

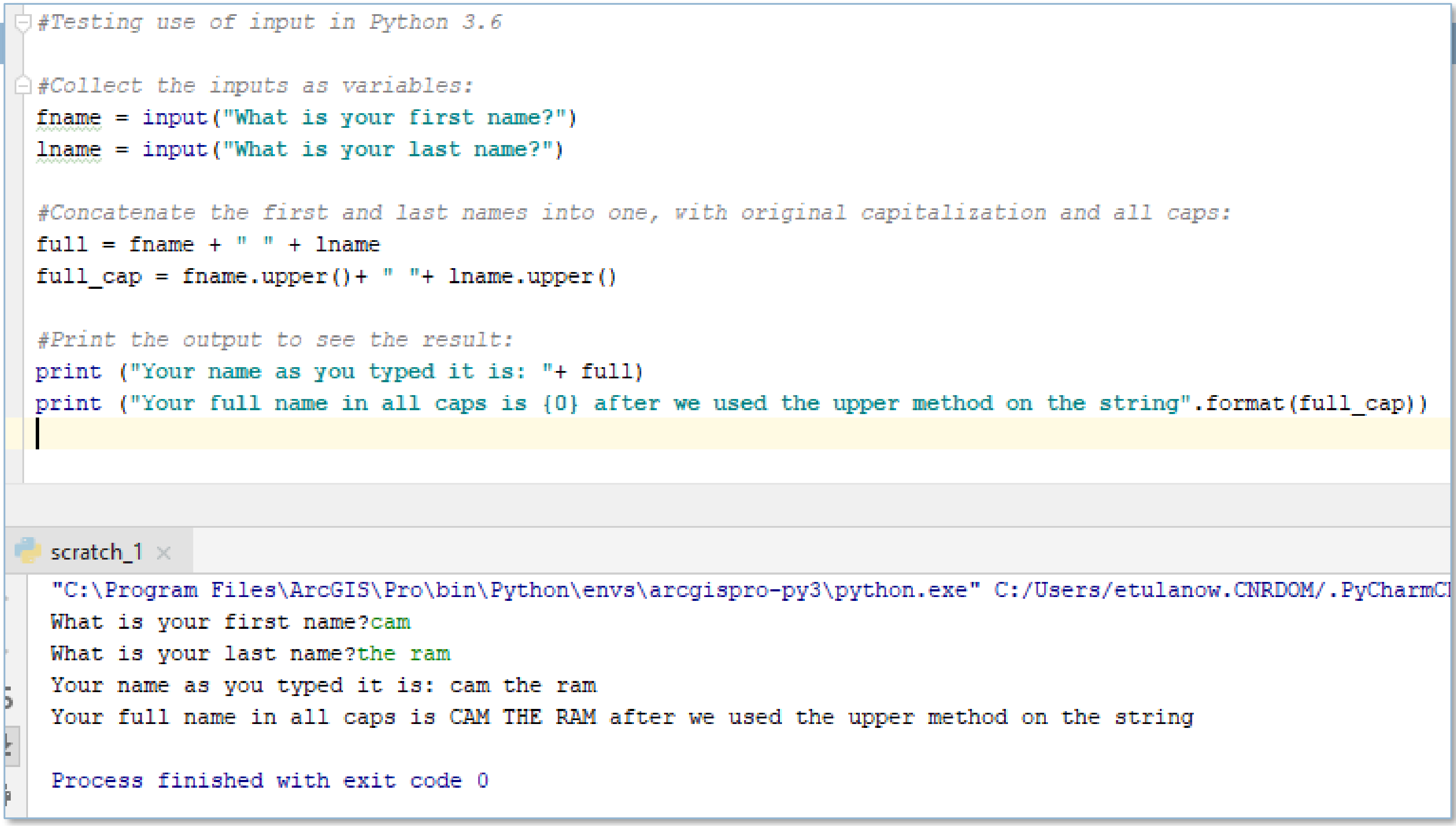


```
try:
    f = open('testfile.txt')
except FileNotFoundError as e:
    print(e)
except Exception as e:
    print(e)
else:
    print(f.read())
    f.close()
finally:
    print("Executing Finally...")
```

```
[Errno 2] No such file or directory: 'testfile.txt'
Executing Finally...
[Finished in 0.1s]
```

Simple user input

- Use input()
- Accepts user input, sets to a variable



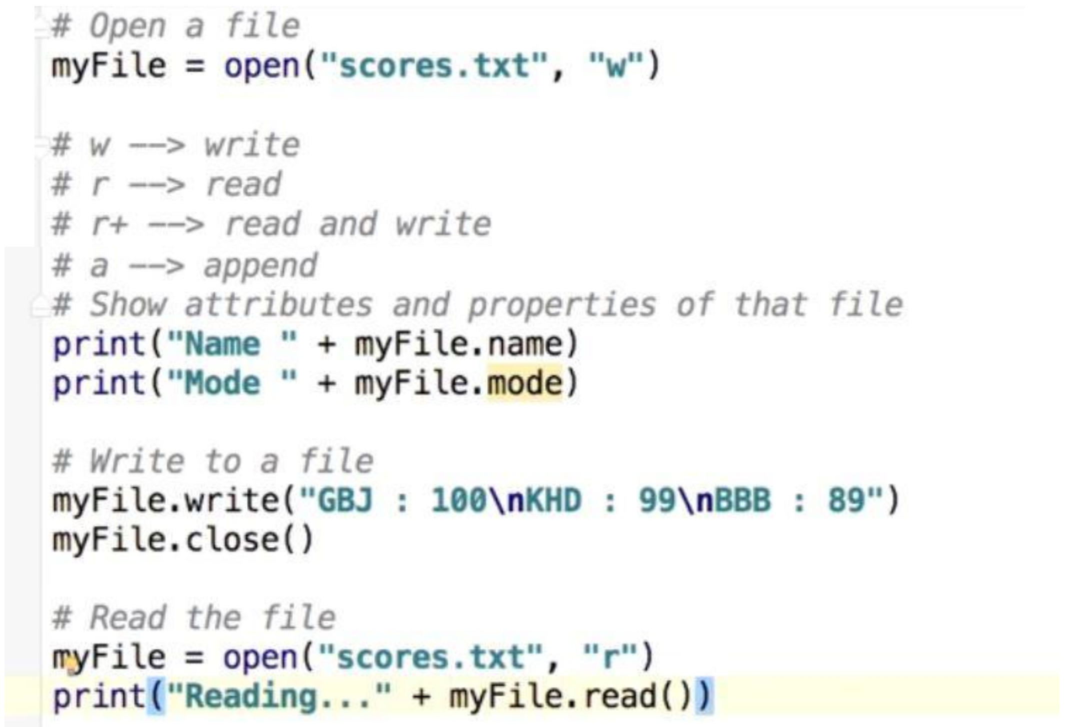
```
#Testing use of input in Python 3.6

#Collect the inputs as variables:
fname = input("What is your first name?")
lname = input("What is your last name?")

#Concatenate the first and last names into one, with original capitalization and all caps:
full = fname + " " + lname
full_cap = fname.upper()+ " "+ lname.upper()

#Print the output to see the result:
print ("Your name as you typed it is: "+ full)
print ("Your full name in all caps is {0} after we used the upper method on the string".format(full_cap))
```

scratch_1

```
"C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\python.exe" C:/Users/etulanow.CNRDOM/.PyCharmC
What is your first name?cam
What is your last name?the ram
Your name as you typed it is: cam the ram
Your full name in all caps is CAM THE RAM after we used the upper method on the string

Process finished with exit code 0
```

Working with files

- Text files and CSVs can be created, opened, written to



```
# Open a file
myFile = open("scores.txt", "w")

# w --> write
# r --> read
# r+ --> read and write
# a --> append
# Show attributes and properties of that file
print("Name " + myFile.name)
print("Mode " + myFile.mode)

# Write to a file
myFile.write("GBJ : 100\nKHD : 99\nBBB : 89")
myFile.close()

# Read the file
myFile = open("scores.txt", "r")
print("Reading..." + myFile.read())
```

```
    with open('Wildfires.txt','r') as f:   #open the file
        lstFires = f.readlines() #read the file into a list
        cur = arcpy.InsertCursor("FireIncidents")

        for fire in lstFires: #loop through each line
            if 'Latitude' in fire: #skip the header
              continue
        vals = fire.split(",") #split the values based on comma
        latitude = float(vals[0]) #get latitude
        longitude = float(vals[1]) #get longitude
        confid = int(vals[2]) #get confidence value
        #create new Point and set values
        pnt = arcpy.Point(longitude,latitude)
        feat = cur.newRow()
        feat.shape = pnt
        feat.setValue("CONFIDENCEVALUE", confid)
        cur.insertRow(feat) #insert the row into featureclass
except:
    print(arcpy.GetMessages()) #print out any errors
finally:
    del cur
    f.close()
```

From Programming Python for ArcGIS Pro book

❶ Open in write or append mode
❷ Write line by line or paragraphs
❸ Close file to save it

PSA book Ch. 8.7 – 8.8, p. 274

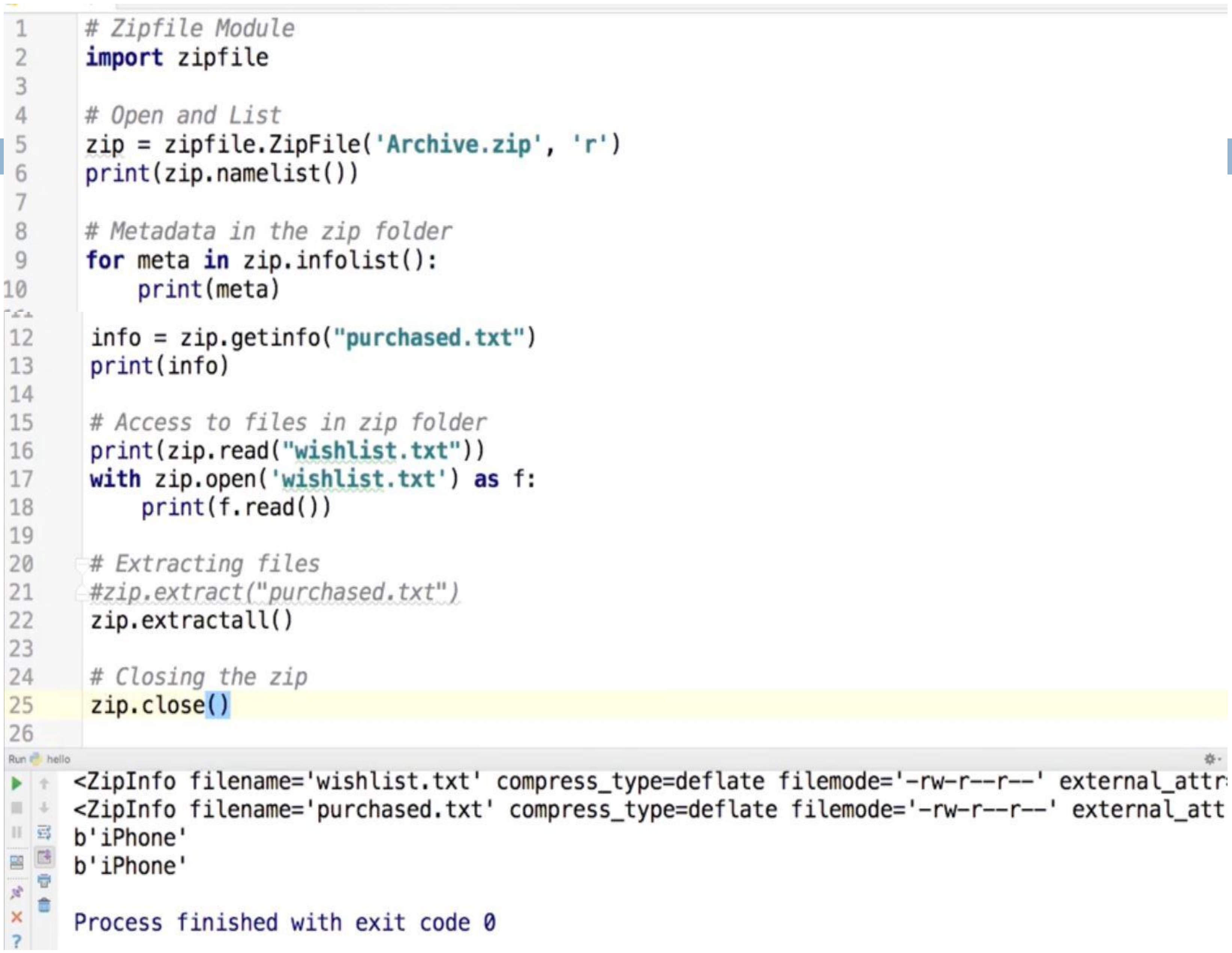
Working with files

- Text files and CSVs can be created, opened, written to
  - Uses for text files
    - Log process steps and track time
    - Create reports
    - Join tables
    - Create address labels after geocoding
  - Useful methods
    - `file()` creates new file or opens existing file
    - `read()` read one line at a time, implement in a loop
    - `readlines()` reads all line at once, then loop through object to access each line
    - `write()` creates one line
    - `writelines()` creates a paragraph
    - `close()` releases the file from memory
  - Newline character `\n`

Learn more in this video

Zip files?

Zipfile info



```
# Zipfile Module
import zipfile

# Open and List
zip = zipfile.ZipFile('Archive.zip', 'r')
print(zip.namelist())

# Metadata in the zip folder
for meta in zip.infolist():
    print(meta)

info = zip.getinfo("purchased.txt")
print(info)

# Access to files in zip folder
print(zip.read("wishlist.txt"))
with zip.open('wishlist.txt') as f:
    print(f.read())

# Extracting files
#zip.extract("purchased.txt")
zip.extractall()

# Closing the zip
zip.close()
```

Run hello

```
<ZipInfo filename='wishlist.txt' compress_type=deflate filemode='-rw-r--r--' external_attr
<ZipInfo filename='purchased.txt' compress_type=deflate filemode='-rw-r--r--' external_att
b'iPhone'
b'iPhone'

Process finished with exit code 0
```

Putting it together

- Follow-along demo
- Writing a simple script to:
  - Build off Lesson 1a and 1b skills
    - Create variables, create and loop through list
  - Evaluate conditions with an if statement
    - Incorporate string functions into an if
    - Allow user inputs with input() and use in the if
  - Highlight common errors
    - Implement the above code with a try-except statement

Watch the
Demo Video

Review and Lesson conclusion

Lesson 1 – Python Basics

Complete the code:

```
# You have several items you want to list out
# Let's complete the code

mtnList =  __ "Massive", "Elbert", "Longs", "Quandary"__
 ___  mtn in mtnList_  # What type of loop?
                       # What do we need at the end of the line?
    print _ __ _       # Don't forget indentation
                       # What gets printed?
```

```
# SOLUTION
mtnList =  ["Massive", "Elbert", "Longs", "Quandary"]
 for mtn in mtnList:   # What type of loop?
                       # What do we need at the end of the line?
    print (mtn)        # Don't forget indentation
                       # What gets printed?
```

What did you just learn?

1. What Python statement evaluates a conditional?
2. Describe the purpose and difference between:

   =   ==   !=   +=

3. Name the 3 modes for opening text files
4. What statement can be used to redirect the code if there’s an error?
5. Do extra spaces or empty lines in your code matter?

Lesson 1 objectives

- Familiarize yourself with basic Python structure and syntax
- Understand language constructs including:

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

- Ability to write a basic script making use of simple strings, numbers, looping, and if
- Describe the ways and places to write, save, and execute code
- Navigate the Python help and documentation to find answers on your own

Lesson 1 conclusion

- Watch lecture videos
- Readings: Python Scripting for ArcGIS Pro Chapter 4
- What's due
  - All of Lab 1and Quiz 1a & 1b
  - By start of the next lecture
- Supplement:
  - PSA Exercise 4, posted to Canvas
  - Esri web course Python for Everyone
  - Python Programming Beginner Tutorial YouTube playlist by Corey Schafer

For next week:

Lesson 2: Geoprocessing with arcpy

- Watch Lesson 2 videos
- Read PSA book, Chapter 5
- Look through Geoprocessing and Python section of ArcGIS Pro help
- Optional- Esri web course: Python Scripting for Geoprocessing Workflows

- ***NEW to ArcGIS Pro?*** Check these out before Lesson 2:
  - PSA Book Chapter 3
  - Get Started with ArcGIS Pro web course
  - Video playlist on ArcGIS Pro
  - Know how to open a project, add data, look at attributes