

# Python for ArcGIS Introduction

---

## Description

Programming tools are now a standard feature within GIS software packages and allow GIS users to automate, speed up, and become more precise in their data management and analytic work. This workshop is designed for GIS users who have little to no experience with computer programming and will cover core programming concepts related to GIS using the Python programming language. The workshop will focus on guiding participants through hands-on exercises designed to provide the essential skills to programmatically manipulate data as part of a GIS workflow. This workshop is designed to be preparation for the following workshop on **Advanced Python for ArcGIS, but may be taken independently.**

## Specific Topics Include:

- Core Python programming concepts
- Introduction to ArcPy site package for ArcGIS
- Working with geospatial data using Python and ArcPy
- Simple data management and geoprocessing tasks

## Outline

---

Part I	Break	Part II
<b>I. Introduction</b>		
<b>II. Data and Software</b>		
<ul style="list-style-type: none"><li>• Downloads</li><li>• ArcGIS Pro</li><li>• ArcGIS Notebooks</li><li>• Python Window</li></ul>		
<b>III. Python Basics</b>		
<ul style="list-style-type: none"><li>• What is Python?</li><li>• Print Function</li><li>• Variables</li><li>• Basic Data Types<ul style="list-style-type: none"><li>▪ Strings</li><li>▪ Numbers</li><li>▪ Booleans</li><li>▪ Lists</li><li>▪ Tuples</li><li>▪ Sets</li><li>▪ Dictionaries</li></ul></li><li>• Data Type Conversion</li><li>• Simple Math with Python</li><li>• Python Syntax and Style</li><li>• Conditional Statements and Decision Making</li><li>• Loops</li><li>• Functions</li><li>• Classes and Methods</li></ul>		
		<b>IV. Calculate Fields Using Python</b>
		<b>V. Introduction to ArcPy</b>
		<ul style="list-style-type: none"><li>• What is ArcPy?</li><li>• <code>import</code> Statements</li><li>• ArcPy Help Documentation</li></ul>
		<b>VI. Using ArcPy</b>
		<ul style="list-style-type: none"><li>• Describing Data<ul style="list-style-type: none"><li>▪ System Paths vs. Catalog Paths</li><li>▪ Formatting Strings</li></ul></li><li>• Listing Data<ul style="list-style-type: none"><li>▪ Geoprocessing Environment Settings</li><li>▪ List Comprehensions</li></ul></li><li>• Geoprocessing Tools</li></ul>
		<b>VII. Conclusion</b>

## I. Introduction

---

## Instructors

### James Whitacre

**Chief, GIS Services Division | Pennsylvania Game Commission**

[jawhitacre@pa.gov](mailto:jawhitacre@pa.gov)

James Whitacre is Chief of the GIS Services Division of the Pennsylvania Game Commission where he leads, manages, and provides vision for the Agency's geospatial and mapping program. Formerly, he was the GIS Research Scientist for the Carnegie Museum of Natural History where he managed the GIS Lab at Powdermill Nature Reserve, the Museum's environmental research center, and supported museum staff and affiliated researchers with geospatial technologies and needs. Whitacre was also the GIS Manager from 2011 to 2014 at the Museum. Before returning to the Museum in 2018, Whitacre was the GIS Specialist for the Main Library at the University of Illinois at Urbana-Champaign where he provided GIS consultations for researchers and scholars, and taught GIS workshops to promote the use of GIS in research. Whitacre holds a Bachelor of Arts in Zoology from Ohio Wesleyan University and a Master of Science in Geography, concentrating on GIS and cartography, from Indiana University of Pennsylvania. Whitacre is past board member and Past President of Keystone GIS (formerly PaMAGIC).



### Emily Clees

**Geospatial Specialist II, Northcentral Region, GIS Services Division | Pennsylvania Game Commission**

[eclees@pa.gov](mailto:eclees@pa.gov)

Emily Clees is a Geospatial Specialist II for the Northcentral Region of the Pennsylvania Game Commission where she completes quality control checks on collected data, runs analysis for regional and state-wide projects, and creates tools to help visualize data collected in the Pennsylvania Game Commission. Prior to her role as a Geospatial Specialist, she received a Bachelor of Science in Environmental Resource Management with a minor in GIS and a Master of Science in Forest Resources from Penn State. Emily also received a GIS certificate from Penn State World Campus.



## Welcome to Python Bootcamp!

- That is what this will feel like!! And that is what is intended!
- Lots of information in a short amount of time...your brain will hurt!
- You may feel lost at first...practice and perseverance will help (I will try to go slow!)
- Have patience...it will take time for it all to settle into your head...and heart!



## Philosophy Over Practice

- I am not a traditional Python developer...my journey comes only through GIS
- I will be presenting a philosophy of how to approach Python in ArcGIS
- The focus will be on:
  - Foundational principles
  - Methods for how to systematically approach a project that would benefit from Python
- There will be some practice today, but it will be minimal
- Practice will be on YOU after the workshop!!! You will only get better if you dive in and start implementing Python in your everyday work.

# Why do I teach Python? Because of **Volvo** and **Seatbelts**!

## II. Data and Software

---

[Top](#)

### Downloads

[Top](#)

- Go to repo at <https://github.com/whitacrej/Python-For-ArcGIS>
- Click **Code** then click **Download Zip**
- **Extract** zip file to your Desktop or other well-known folder

### ArcGIS Pro

[Top](#)

- Recommended to use latest version 3.2+ or 2.9+
- Minimum version of 2.7+ required

See [ArcGIS Pro Documentation](#)

### ArcGIS Pro vs. ArcMap

- Stop using ArcMap! Seriously...it's time! ArcMap is currently in mature support and is being retired March 2026 (see [ArcMap Life Cycle](#))
- Oh, you HAVE to use it? Fine...
  - Nearly everything in this workshop still applies!!!
  - Except:
    - ArcMap does not use ArcGIS Notebooks...Use a code editor or the Python Window instead
    - ArcMap uses Python 2, which is being deprecated...just like ArcMap!
    - I'll point out a few other things that are different along the way...
- No, sorry, I don't have time to address every ArcMap concern...

### ArcGIS Notebooks

[Top](#)

- Built on top of [Jupyter Notebook](#)
- Included and integrated with ArcGIS Pro (starting at version 2.5)
- Optimized for ArcGIS Pro and Python 3
- Used to easily and quickly write and run code directly in ArcGIS Pro

### Python in ArcGIS Notebooks

- Can perform analysis and immediately view results in a geographic context, interact with the emerging data, document and automate your workflow, and save it for later use or share it
- ALL Python functionality in ArcGIS Pro is available through ArcGIS Notebooks
- Provides access to content in your map allowing for interactive workflows

### Markdown and HTML in ArcGIS Notebooks



- ArcGIS Notebooks utilize Markdown and HTML markup languages to format Markdown cells in a Notebook
- Markdown:
  - Lightweight markup language that you can use to add formatting elements to plaintext text documents
  - One of the world's most popular markup languages
  - Markdown syntax is added to text to indicate which words and phrases should look different
- HTML:

- Stands for Hyper Text Markup Language
- The standard markup language for creating Web pages
- Describes the structure of a Web page and tell the browser how to display the content
- This Notebook uses Markdown and HTML **EXTENSIVELY** to make it look the way it does!
- We will not go over Markdown in detail, but will a little bit

See:

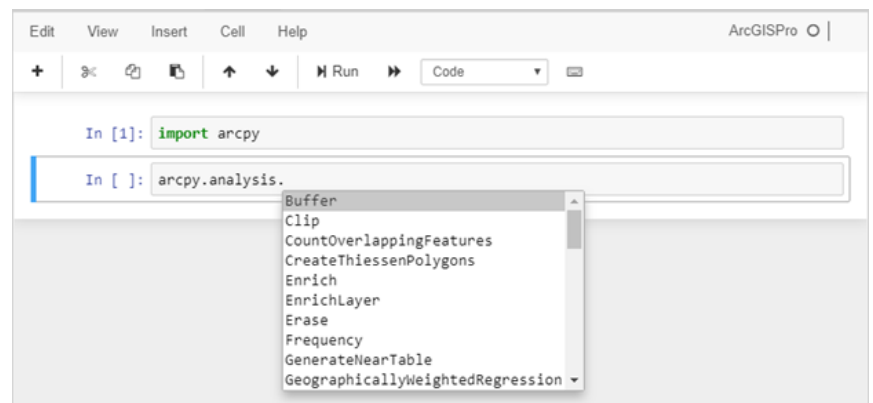
- [ArcGIS Notebooks Get Started](#)
- [The Markdown Guide](#)
- [W3 Schools HTML Tutorial](#)

## Create a new ArcGIS Notebook

- Click the **Analysis** tab, and click the **Python** dropdown, and click  **Python Notebook**
- In the **Catalog Pane**, **right-click on the folder** where you want to create it, click **New** (at the top), the click  **Notebook**

**We will be using ArcGIS Notebooks for nearly all our coding in the workshop!!!**

**I will go over tips and tricks as we go!**




## Python Window

Top

- Allows for running geoprocessing tools while also taking advantage of other Python modules and libraries
- Can be used for single-line code (e.g., for geoprocessing tools)
- Can be used for testing syntax and longer, more complex scripts that might be used for automating workflows
- Available in both ArcGIS Pro and ArcMap
- Replaces the Command Line from earlier releases of ArcGIS (pre-10.x)

See [Python Window](#)

## Open the Python Window

- Click the **View** tab, and click the  **Python Window** button
- Dock the Python Window to the bottom of ArcGIS Pro and Hide it

### Python Window Components

- Bottom section is the interactive Python prompt
  - This is where code is entered
- Top section is the transcript of the code ran and the output
- Code is generally executed one line at a time and displayed immediately. Exceptions include:
  - Multi-line constructs (such as `if` statements)
  - Hitting ENTER at the end of multi-line code will run it (you may need to hit ENTER a few times when you are ready to run the code)
  - Shift + Enter will allow multiple lines to be written
- Other features of the Python window
  - Autocompletion



- Conditional and Iteration execution
- Scripts can be saved and reused or opened with another code editor

## Following Along...

- The screen may be hard to see...so have everything open on your screen to follow along.
- Turn Line Numbers on:
  - On the Notebook, click **View** and click **Toggle Line Numbers**
  - This will help reference specific lines of code when I want to point out something, or if you have a question
- Toggle between applications and ArcGIS Pro Views
  - Use Alt + Tab keyboard shortcut to toggle between application windows
  - Use Ctrl + Tab to toggle between ArcGIS Pro Views (this also works in browsers to toggle between tabs!)

**A word on keyboard shortcuts...**  
**LEARN THEM AND USE THEM!!!**  
**I will point some out as we go along.**

## III. Python Basics

---

[Top](#)

### What is Python?

[Top](#)

- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics
- Good for scripting and for application development
- Simple, easy to learn syntax emphasizing readability (**Great for beginners!!!**)
- Has high-level built in data structures
- Supports modules, libraries, and packages, which encourages program modularity and code reuse
- Increases productivity due to no compilation step
- Debugging Python programs is easy; often the quickest way to debug a program is to add a few `print` statements to the code
- Open-source and freely distributed

See [Python Software Foundation: What is Python Executive Summary](#) for more information

See [Python in ArcGIS Pro](#)

### Some General Notes and Tips

- Learning a programming language is like like learning a new foreign language
  - There is grammar, or syntax
  - There is vocabulary, or tools, functions, methods, and modules
  - It's a new way of thinking
- People will refer to good code as being '*Pythonic*'
- Just like in ArcGIS, there is more than one way to do many things!
- Python and ArcGIS documentation sources are invaluable!!
  - You should reference them often and always have them ready when you are actively coding!!
  - Developing the skills to read and understand documentation is ESSENTIAL!!
  - Developing the skills to search for Python code examples is also ESSENTIAL!! Why write code when someone has already done for you!

**Disclaimer: The way I teach Python is specific to ArcGIS and covers the most important elements I have found to be helpful for beginners. Other Python instructors might emphasize other aspects more than I may.**

### Print Function

[Top](#)

- What is the `print()` function?

- A way to make your script talk back to you
- A way to see what a variable is
- It is used frequently!!
- How to use the print function

- Type `print()` \*
- Add the variable or string within the parentheses `()`

Using just `print` without `()` is improper syntax in Python 3. In Python 2 `print` without `()` is actually proper syntax. You should be aware of this if you are working with older code intended for Python 2.

```
In [ ]: # Click Run above to run the code
# Or hit Ctrl + Enter on the keyboard!

# This WILL work in Python 2 and Python 3!!!
print('Python is so cool!')
```

```
In [ ]: # This WILL NOT work in Python 3 (i.e., ArcGIS Pro)
print 'Python is so cool!'
```

```
In [ ]: # Printing multiple arguments
import os # We'll talk about this later!!
print('My name is', os.getlogin(), 'and I am', 30)
```

### Your turn!

1. Add a new cell below
2. Type a `print` statement!

## Variables

### Top

- What is a variable?
  - Reserved memory locations to store values for repeated use in the code
  - When you create a variable you reserve space in memory for the value
  - Stored as a specific data type (e.g., string, integer, floating point, list, dictionary, etc.)
  - Value and data type can be changed, or reassigned, very simply
- How to set, or declare, a variable
  - No explicit declaration needed
  - Type a descriptive word that represent what you want to store for use later
  - Type the equals sign: `=`
  - Type what the variable equals

```
In [ ]: # Create variables

food = 'cheese'

food_count = 6

print(food)
print(food_count)
```

### Cool tips...

```
In [ ]: # Create multiple variables
food, food_count = 'bread', 100

print(food)
print(food_count)
```

```
In [ ]: # Make multiple variables the same value
food1 = food2 = food3 = 'banana'

print(food1)
print(food2)
print(food3)
```

## Basic Data Types

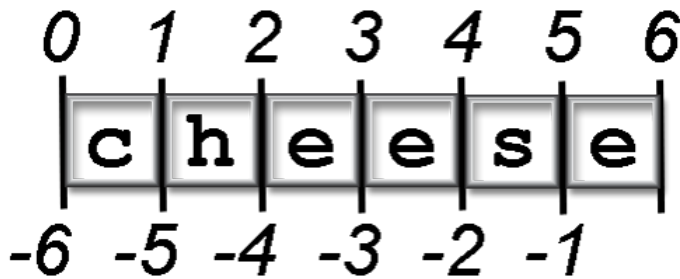
### Top

Data Type	Examples
String	"cheese" or 'Food Time'
Integer Number	68 or 23456 or 0
Float Number	345.67 or 28.1 or 98.0
Boolean	True or False
List	["apple", "orange"]
Tuple	("apple", "orange")
Dictionary	{"lat":39.799, "lon":-89.64}

## Strings

### Top

- What is a string?
  - Contiguous set of characters represented in quotation marks
  - Simply put, it is text values
  - Number characters are not treated like numbers (i.e., `1 != '1'`)
- What can you do with a string?
  - Concatenation (ex: `'cheese' + 'whiz'` will equal `'cheese whiz'`) *Note the space!!*
    - Plus (+) sign is the string concatenation operator (can only operate on all string values: `'cheese' + 1` is an error)
    - Asterisk (\*) is the repetition operator: `'cheese' * 3` is `'cheesecheesecheese'`
  - Slicing (e.g.: `'cheese'[1:4]` will equal `'hee'` as the index starts at 0)



```
In [ ]: # Test math with strings?? Yup! Concatenation!
print('cheese' + 'whiz') # Note the space...
print('cheese' * 3) # Note no space...

# Test some slicing operations
print('cheese'[1:4])
print('cheese'[:2])
print('cheese'[:-2])
print('cheese'[2:])
print('cheese'[-2:])
```

## Numbers

### Top

- Number datatypes store numeric values that act like numbers (e.g., for math operations)
- **Integer:** Number **without** decimal
- **Float:** Number **with** decimal
- What can you do with numbers?
  - Simple math - e.g: `5 + 7 - 3`
  - Math with variables - e.g.:

```
three = 3
5 + 7 - three
```

In ArcGIS, feature class attribute tables will have long and short integers and float and double precision numbers. In Python, integers are treated like long and float is treated like double precision.

```
In [ ]: # ALL integers
print(5 + 7 - 3)

# Float!
three = 3.3
print(type(three))

# Converts to a float...
print(5 + 7 - three)
```

## Booleans

### Top

- Boolean values are `True` or `False`
- Used for evaluating whether something is `True` or `False`
- The following will be evaluated as `False` :
  - `None`
  - `False`
  - Zero of any numeric type (i.e., `0`)
  - Any empty sequence (e.g., `''`, `()`, `[]`)
  - Any empty dictionary (e.g., `{}`)
- Use the `bool()` function to evaluate a variable's boolean value

```
In [ ]: # What boolean value will each variable evaluate to?

a = True # Note Title Case!
b = 'False'
c = ''
d = 0
e = 1 # Try negative 1!
f = [1]
g = []

print(bool(a))
print(bool(b))
print(bool(c))
print(bool(d))
print(bool(e))
print(bool(f))
print(bool(g))
```

## Lists

### Top

- What is a list?
  - Series of ordered items or objects
  - Compound data type
  - Enclosed by square brackets `[]` and items separated with commas `,`
  - Lists are mutable (i.e., items and number of items can be changed, replaced, added, or deleted)
- How are lists used?
  - Find one or a range of items

```
fruit_list = ['apples', 'oranges', 'bananas']
fruit_list[1] # returns oranges
fruit_list[1:2] # returns ['oranges']
fruit_list[1:3] # returns ['oranges', 'bananas']
```

\* Iterate through it (use a loop)

```
fruit_list = ['apples', 'oranges', 'bananas']

for fruit in fruit_list:
    # Do something...
    print(fruit)
```

\* Change list values



```
fruit_list[1] = 'peaches'
fruit_list.append('cherries')
fruit_list.remove('apples')

print(fruit_list) # returns ['peaches', 'bananas', 'cherries']
```

See [Tutorials Point: Python - Lists](#)

```
In [ ]: # Create a List
fruit_list = ['apples', 'oranges', 'bananas']
print(fruit_list)

# Test List slicing
print(fruit_list[1])

print(fruit_list[1:2])

print(fruit_list[1:3])

print(len(fruit_list))
```

```
In [ ]: # Iterate over the list...more on this later
for fruit in fruit_list:
    print(fruit)
```

```
In [ ]: # Change the List...
fruit_list = ['apples', 'oranges', 'bananas']

fruit_list[1] = 'peaches'
fruit_list.append('cherries') # This is called a 'Method'...more on that later!
fruit_list.remove('apples')
del fruit_list[1]

print(fruit_list)
```

## Tuples

### Top

- What is a tuple?
  - Similar to a list, but enclosed by parentheses `()`
  - Immutable (i.e., not changeable; read-only)
- How are tuples used?
  - Find one or a range of items

```
fruittuple = ('apples', 'oranges', 'bananas')
fruittuple[1] # returns oranges
fruittuple[1:2] # returns (oranges,)
fruittuple[1:3] # returns (oranges, bananas)
```

\* Iterate through it (use a loop)

```
fruittuple = ('apples', 'oranges', 'bananas'])

for fruit in fruittuple:
    # Do something...
    print(fruit)
```

**Tuples are important when you need to preserve data that you don't want to change. In ArcGIS, cursors always create tuples for this reason...more on that later.**

```
In [ ]: # Create a tuple
fruittuple = ('apples', 'oranges', 'bananas')
print(fruittuple)

# Test tuple slicing
print(fruittuple[1])
print(fruittuple[1:2]) # Is there a comma at the end? Why is that?
print(fruittuple[1:3])
```

```
In [ ]: # Iterate over the tuple...more on this later
fruittuple = ('apples', 'oranges', 'bananas')

for fruit in fruittuple:
    print(fruit)
```

```
In [ ]: # Try to assign a value to a index 1 of the tuple...oops!
fruittuple[1] = 'peaches' # Invalid syntax with tuple, this will produce an error, but it will work for a list
```

## Sets

### Top

- What is a set?
  - Unordered collection with no duplicate elements
  - Curly braces `{}` or the `set()` function can be used to create sets
  - Note: to create an empty set you have to use `set()`, not `{}`; the latter creates an empty dictionary, a data structure that we discuss in the next section.
- How are sets uses?
  - Basic uses include membership testing and eliminating duplicate entries

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'} # Returns unique list items (i.e., no duplicates: {'orange',
'banana', 'pear', 'apple'})

basket_list = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
basket = set(basket_list) # Converts a list to a set and returns unique list items: {'orange', 'banana', 'pear', 'apple'}

unique_chars = set('abracadabra') # Returns unique characters: {'a', 'r', 'b', 'c', 'd'} (Are the characters in order?)

* Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

a = set('abracadabra')
b = set('alacazam')

print(a) # Returns unique letters in a: {'a', 'r', 'b', 'c', 'd'}

print(a - b) # Returns letters in a but not in b (difference): {'r', 'd', 'b'}

print(a | b) # Returns letters in a or b or both (union): {'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

print(a & b) # Returns letters in both a and b (intersect): {'a', 'c'}

print(a ^ b) # Returns letters in a or b but not both (symmetric difference): {'r', 'd', 'b', 'm', 'z', 'l'}
```

**Note: Sets are good to know about, but we will not work with them in much detail in this workshop.**

```
In [ ]: # Create a set...this is uncommon
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)

# Convert a list to a set...this is VERY common!
basket_list = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
basket = set(basket_list)
print(basket)

# Test if a value is in the set
print('orange' in basket)
print('crabgrass' in basket)
```

```
In [ ]: # Perform set operations on unique letters from two words
a = set('abracadabra')
b = set('alacazam')

print(a)

# Difference
print(a - b)

# Union
print(a | b)

# Intersect
print(a & b)

# Symmetric difference
print(a ^ b)
```

## Dictionaries

### Top

- What is a dictionary?
  - An ordered set of `key:value` pairs enclosed by curly brackets `{}`
    - [As of version 3.7, dictionaries are ordered](#). They were unordered previously!
  - Indexed by **Keys** that must be unique values and are typically strings (i.e., in quotes), but can be other Python data types
  - Each key is a **Value** that can be any Python object
  - Sometimes referred to as 'associative memories' or 'associative arrays'
- How are dictionaries used?
  - Find the value that goes with a key

```
dict_ex = {'key': 'value', 'lat': 39.98, 'long': -89.65}
dict_ex['key'] # Returns value
dict_ex['lat'] # Returns 39.98
dict_ex['long'] # Returns -89.65
```

\* Get a list of keys

```
dict_ex.keys() # Returns dict_keys(['key', 'lat', 'long'])
list(dict_ex) # Returns ['key', 'lat', 'long']
```

\* Get a list of values

```
dict_ex.values() # returns ['value', 39.98, -89.65]
```

**Note: Dictionaries are good to know about, but we will not work with them in much detail in this workshop.**

```
In [ ]: # Create a dictionary
dict_ex = {'key': 'value', 'lat': 39.98, 'long': -89.65}

# Get dictionary values by key
print(dict_ex['key'])
print(dict_ex['lat'])
print(dict_ex['long'])

# Get dictionary keys
print(dict_ex.keys())
print(list(dict_ex))

# Get all dictionary values
print(dict_ex.values())
```

## Data Type Conversion

### Top

- Converting between data types is common
- There are built-in functions to deal with this
- Example: You want to concatenate a numerical value into a string
- There are also functions to determine what the data type is of a variable

```
In [ ]: # Run the following code...What happens?

x = 99

print("There are " + x + " files.")
```

```
In [ ]: # Change x to be str(x)

print("There are " + str(x) + " files.")
```

```
In [ ]: # Create different data type variables
x = 99
s = "99"
l = [s, x]
t = (s, x)

# Convert variable
print(type(x))
print([str(x)])

print(type(s))
print([int(s)])

print(type(l))
```

```
print(tuple(t))

print(type(t))
print(list(t))
```

# Simple Math with Python

[Top](#)

## Python Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a - b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
// Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

## More Examples

Operator	Explanation	Example	Result
x + y	x plus y	1.5 + 2.5	4.0
x - y	x minus y	3.3 - 2.2	1.1
x * y	x times y	2.0 * 2.2	4.4
x / y	x divided by y	4.0 / 1.25	3.2
x // y	x divided by y (floor division)	4.0 // 1.25	3.0
x % y	x modulo y	8 % 3	2
-x	negative expression of x	x = 5; -x	-5
+x	x is unchanged	x = 5; +x	5
x ** y	x raised to the power of y	2 ** 3	8

Source: [Calculate Field Python examples](#)

When performing field calculations with a Python expression, Python math rules are in effect. For example, dividing two integer values will always produce an integer output (3 / 2 = 1). To get decimal output:

- One of the numbers in the operation must be a decimal value: 3.0/2 = 1.5
- Use the float function to explicitly convert the value to a float:

```
float(3)/2 = 1.5

# or

3/float(2) = 1.5
```

```
In [ ]: # Perform Python math
print(1.5 + 2.5)

print(3.3 - 2.2)

print(2.0 * 2.2)

print(4.0 / 1.25)

print(4.0 // 1.2)

print(8 % 3)

x = 5
print(-x)
print(+x)
```

```
print(2 ** 3)
```

# Python Syntax and Style

[Top](#)

## Python Syntax Rules

- Variables cannot start with a number or have a space in it

```
1line = 5 # This will not work...
```

```
    a line = 5 # Neither will this...
```

- Here is a list of common **\*reserved words**; *do NOT* use these as variable names:
  - Basically, if it turns a color when you are done typing it, don't use it as your variable's name!
  - There are likely many more reserved words!

and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try

- Variable name capitalization matters!!

```
Cat != cat
```

- Colons and indentation matters!!
  - Typically 2 or 4 spaces (represented by '.')
  - 4 spaces are preferred (using Tab in the IDE should do this automatically)

```
# This will not work...
```

```
if x
print(x)
```

```
# But this will (periods are shown to emphasize the indentation):
```

```
if x:
...print(x)
```

- Quotations are a bit tricky, but very cool
  - Single ( ' '), double ( " ") and triple-single and triple-double ( ''' or """) quotes can be used to denote strings
  - Make sure to end the string denotation with the same type of quote structure
  - Single quotes ( ' ') are the easiest to type (no Shift!!), so I default to them usually

```
# Quotation Examples
```

```
word1 = 'Dog'
word2 = "'Dog'"
word3 = '"Dog"'
print(word1, word2, word3)
# The three words above will print: Dog, 'Dog', and "Dog"
```

```
words1 = 'That's the dog's toy' # Is a syntax error
words2 = "That's the dog's toy" # Prints: That's the dog's toy
```

```
more_words = ""She said, "Good dog!" And the dog's tail wagged.""
# Prints: She said, "Good dog!" And the dog's tail wagged.
```

- Backslashes can be confusing...

```
# These are all the same thing...
```

```
r'C:\data\things' # I generally prefer this method
'C:\\data\\things' # I use this method sometimes too
'C:/data/things'
```

- Commenting is great!!
  - Use it to help document and explain your code...we will do this throughout the workshop and in exercises!
  - Comments are not run in code; they are ignored
  - Blank lines are also ignored

```
# This is a block comment
```

```
## So is this
```

```
### The blank line below this one will be ignored
```

```
""" This is good for multi-line block comments
Notice that this line is still a comment
Use block comments as much as you need, but not too much
```

```
Don't forget to close the multi-line comment"""
```

```
s = 'something' # This is an in-line comment...use these sparingly in your code
```

## Python Code Styling

### What is 'Code Style'?

- How code is written when more than one possible syntax method is available (e.g., 4-space indentation vs. 2-space indentation)
- Code style is determined by conventions and standards
- While code may be syntactically correct, it may not be stylistically correct

### Why is Code Style Important?

- Code is read much more often than it is written, therefore, it is important **HOW** we write code so that it can be more human-readable
- Code style should be consistent and follow industry best practices
- Helps to balance human-readability with machine-readability

See:

- [PEP 8 -- Style Guide for Python Code](#)
- [Google Python Style Guide](#)

### Naming Conventions

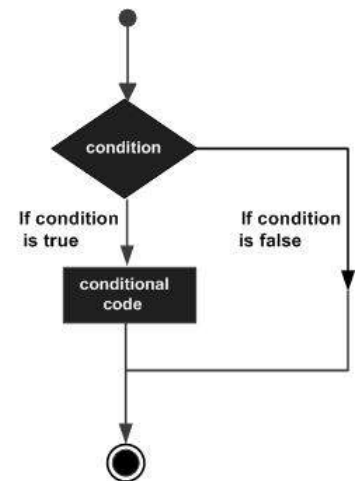
- This is more important than you may realize!!!
- Familiarize yourself with the different types in the [PEP 8 -- Style Guide for Python Code!!](#)
- Note that all lowercase and underscores are used OFTEN!!! Prescribed style for variable and function names...Why?
  - Easier and faster to type!! No shift keystroke required...
  - Easier and faster to read
    - Think about when you read UPPERCASE vs. lowercase
    - We read lowercase letters WAY more than we read UPPERCASE letters, therefore the brain identifies words better in lowercase
- Start using similar naming conventions in your everyday work with file and folder names!!

## Conditional Statements and Decision Making

[Top](#)

- Many times, we need code to make decisions
- Some decisions are easy, while others are complex
- Decisions are made by evaluating whether something is `True` or `False`

Source: [Tutorials Point: Python - Decision Making](#)



### `if` Statements

- `if` statement - one decision/option

```
# If 'x' is True
if x == 100: # Note the colon (:) and the double equals sign (==)

    # Do something or many things
    x = x + 1 # Note the indentation
    print(x)
```

- `if ... else` statement - two decisions/options

```
# If 'x' is True
if x == 100:
    # Do something or many things
    x = x + 1
    print(x)

# If 'x' is NOT True
else:
    # Do something else
    x = x - 1
    print(x)
```

- `if ... elif ... else` statement - many decisions/options

```
if x == 100:
    x = x + 1
    print(x)

# If 'x' is NOT True, try 'y'
elif y == 100:
    y = x + y
    print(y)

# If 'y' is NOT True, try 'z'
elif z == 100:
    z = x ** y
    print(z)

# If everything is NOT True
else:
    w = y % z
    print(w)
```

- `if` statements can be nested

```
if x == 100:
    x = x + 1
    print(x)

    # If 'x' is True, AND 'y' is True
    if y == 100: # ****Notice the second indent!!****
        y = x + y
        print(y)

    # If 'x' is True, but 'y' is NOT True
    else:
        z = x ** y
        print(z)

# If 'x' is NOT True
else:
    w = y % z
    print(w)
```

**Note:** No `end if` is required like many other coding languages! Just unindent to show the end of the section.

## Comparison Operators

Operator	Description	Example
<code>==</code>	If the values of two operands are equal, then the condition becomes true.	<code>(1 == 2)</code> is NOT true
<code>!=</code>	If values of two operands are not equal, then condition becomes true.	<code>(1 != 2)</code> is true
<code>&lt;&gt;</code>	If values of two operands are not equal, then condition becomes true. <i>This is the same as != operator, but is deprecated and not valid in Python 3.</i>	<code>(1 &lt;&gt; 2)</code> is true
<code>&gt;</code>	If the value of left operand is greater than the value of right operand, then condition becomes true.	<code>(1 &gt; 2)</code> is not true.
<code>&lt;</code>	If the value of left operand is less than the value of right operand, then condition becomes true.	<code>(1 &lt; 2)</code> is true.
<code>&gt;=</code>	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	<code>(1 &gt;= 2)</code> is not true.
<code>&lt;=</code>	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	<code>(1 &lt;= 2)</code> is true.
<code>and</code>	Called Logical AND operator. If both the operands are true then then condition becomes true.	<code>(a and b)</code> is true
<code>or</code>	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	<code>(a or b)</code> is true
<code>not</code>	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	<code>not(a &amp;&amp; b)</code> is false
<code>in</code>	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	<code>x in y</code> , here <code>in</code> results in a 1 if <code>x</code> is a member of sequence <code>y</code>

Operator	Description	Example
not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y

Adapted from: [Tutorials Point: Python - Basic Operators](#)

```
In [ ]: # Test conditional statements
a = 21
b = 10
c = 0

# Equals operator...note double = for comparison
if a == b:
    print("Line 1 - a is equal to b")
else:
    print("Line 1 - a is not equal to b")

# Not operator
if a != b:
    print("Line 2 - a is not equal to b")
else:
    print("Line 2 - a is equal to b")
```

```
In [ ]: # This is novalid in Python 3:
if a <> b:
    print("Line 3 - a is not equal to b")
else:
    print("Line 3 - a is equal to b")
```

```
In [ ]: # Less than
if a < b:
    print("Line 4 - a is less than b")
else:
    print("Line 4 - a is not less than b")

# Greater than
if a > b:
    print("Line 5 - a is greater than b")
else:
    print("Line 5 - a is not greater than b")
```

```
In [ ]: a = 5;
b = 20;

# Less than or equal to
if a <= b:
    print("Line 6 - a is either less than or equal to b")
else:
    print("Line 6 - a is neither less than nor equal to b")

# Greater than or equal to
if b >= a:
    print("Line 7 - b is either greater than or equal to b")
else:
    print("Line 7 - b is neither greater than nor equal to b")
```

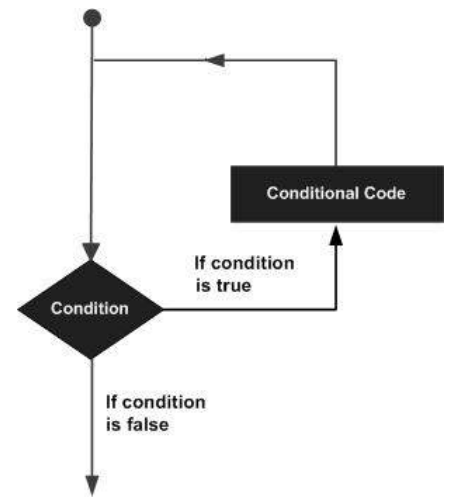
## Loops

### Top

- Code is generally executed consecutively
- Loops allow for a block of code to be executed several times
- Two basic types: `for` and `while` loops

See [Tutorials Point: Python - Loops](#)





## for Loops

- Work on ordered lists and other sequences
- Repeats a block of code for each element of the list
- When the end of the list is reached, the loop ends

```
for item in list_of_items:
    # Do some code...
```

```
In [ ]: # Test a Loop on a List
a_list = ['a', 'b', 'c', 'd']

for item in a_list:
    print(item)
```

## while Loops

- Executes the code block while a given condition is true
- Requires an exit condition, otherwise it could be an infinite loop (this is bad!!)

```
i = 0 # This is called a sentry variable
while i <= 10:
    print(i)
    i += 1 # Increment the sentry variable to ensure the exit condition
i += 1 is another way to increment a numeric value by a constance value (in this case 1).
```

See [Tutorials Point: Python Assignment Operators Example](#) for other similar operations. Python is full of these types of tricks!

```
In [ ]: i = 0

while i <= 10:
    print(i)
    i += 1
```

## Functions

### Top

- Block of organized, reusable code
- Performs a single, related action
- Good when a function needs to be reused a lot
- Many built-in functions (like `print()` and `str()`)
- Users can create their own function, commonly referred to as *user-defined* functions

## Defining Functions

- Function blocks begin with the keyword `def` followed by the function name and parentheses `()`.
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

# Basic Function syntax

```
def function_name( parameters ):
    """function_docstring""" # This is optional, note triple-quotes
    # Function code
    return # expression
```

Source: [Tutorials Point: Python - Functions](#)

```
In [ ]: def tax_calc(bill, percent):
        """Calculates percent tax of restaurant bill."""
        tax = bill * percent
        print('Tax: ${:0.2f}'.format(tax))
        return tax

def tip_calc(bill, percent):
    """Calculates percent tip of restaurant bill."""
    tip = bill * percent
    print('Tip: ${:0.2f}'.format(tip))
    return tip

def total_bill(bill, tax, tip):
    """Calculates the total restaurant bill."""
    total = bill + tax + tip
    print('Total: ${:0.2f}'.format(total))
    return total

# Change the meal_cost value a few times to see how the results change
meal_cost = 100
tax_perc = 0.08
tip_perc = 0.15
print('Bill: ${:0.2f}'.format(meal_cost))

meal_tax = tax_calc(meal_cost, tax_perc)
meal_tip = tip_calc(meal_cost, tip_perc)
meal_total = total_bill(meal_cost, meal_tax, meal_tip)
```

## Classes and Methods

[Top](#)

- Python is an Object-Oriented Language
- Objects are a defined data structure that contain data and code, and that behave in a specific way
- Class is:
  - A defined prototype for an object that defines a set of attributes that characterize any object of the class
  - The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
  - Can have multiple instances in a script
- Method is a special kind of function defined in the Class

**We will work with classes A LOT!! But, we will not create classes.**

Source: [Tutorials Point: Python - Object Oriented](#)

```
In [ ]: # Create a List
fruit_list = ['apples', 'oranges', 'bananas']

# Check data type
print(type(fruit_list)) # What do you notice about the output?
```

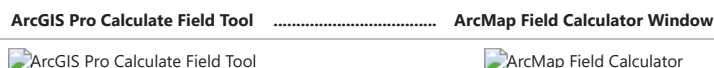
```
In [ ]: # Test the List append method
fruit_list.append('cherries')
print(fruit_list)
```

Now we are ready to put our Python knowledge into practice!

## Break

## IV. Calculate Fields Using Python

- **A great place to practice Python in ArcGIS is by calculating fields**
- ArcGIS Pro:
  - Python 3, Arcade, and SQL can be used to calculate fields; Python 3 is the default
  - Must use the Calculate Field geoprocessing tool; Field Calculator Window is removed
- ArcMap:
  - Python and VB Script can be used to calculate fields; VB Script is the default, so Python needs to be set as the parser
  - Can use the Calculate Field geoprocessing tool or Field Calculator Window
- Simple expressions use the Expression parameter
- More complex calculations can use the Code Block parameter (tool) or the Pre-Logic Script Code block (window)
- **Note: Python can be difficult to debug and check for errors when calculating fields**



### Calculating Fields Using Python

#### 1. Open the *Illinois* Map

#### 2. Open the Attribute Table for the 'Illinois Counties' Layer

- We see that there is a field named 'AREA', but we don't know the units.
- Note the other fields

#### 3. Open the Calculate Field tool in the Geoprocessing Pane

#### 4. Enter the following parameters into the Calculate Field tool

- Input Table: **Illinois Counties** (This parameter will stay the same for this entire exercise)
- Field Name: **Area\_Test**
- Field Type: **Double**

#### 5. See the [Calculate Field Python examples](#) ArcGIS Pro help documentation

- This page is a great resource
- Notice the different examples
- Find the 'Code samples-geometry' section
- These expressions can be used in lieu of the **Calculate Geometry Attributes** tool

#### 6. Enter the following code in the expression box and click Run

#### 7. Delete the 'AreaTest' field when finished

```
In [ ]: """ Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# First try square meters
!Shape.area@squaremeters!

# How close are the values compared to the AREA field?

# Second, try square miles
```

```
!Shape.area@squaremiles!

# Are these values closer to the AREA field? Do you think we can get better?

# Now try

!Shape.geodesicArea@squaremiles!

# Which method is more accurate? Why do you think that is?

# One last trick, type:

None

# This will make all the values NULL in the field...it is good to know about!
```

## Calculating Fields Using Python (continued)

Now we know the units, but we want to calculate the population density. We also want to know the difference between the median and average household income.

### 1. Calculate two fields with the following parameters:

- Field Name: **Density\_Sqmi**
- Field Type: **Double**
- Field Name: **Diff\_Income**
- Field Type: **Double**

### 2. Calculate the fields using the Python expressions below

```
In [ ]: """ Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# Density_Sqmi calculation

!ACSTOTPOP! / !AREA!

#####

# Diff_Income calculation

!ACSAVGHINC! - !ACSMEDHINC!
```

## Calculating Fields Using Python (continued)

Now we would like to have three more fields for a short name, a long name, and a name with the area.

### 1. Calculate three fields with the following parameters:

- Field Name: **Name\_Long**
- Field Type: **Text**
- Field Name: **Name\_Short**
- Field Type: **Text**
- Field Name: **Name\_Area**
- Field Type: **Text**

### 2. Calculate the fields using the Python expressions below

```
In [ ]: """ Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# Name_Long =

!NAME! + ', ' + !STATE_NAME!

#####
```

```

# Name_Short =
!NAME!.replace(' County', '')

# Note the first parameter...what do you notice?

#####

# Name_Area =
!NAME! + ' (Area: ' + !AREA! + ' SqMi)'

# Did it work? Why or why not? Hint: Check the ArcMap Results window if you can't figure it out
# Change the expression to be correct...

# Name_Area can also be calculated this way...

# Name_Area =
' '.join([!NAME!, ' (Area:', str(!AREA!), ' SqMi)'])

# What do you notice that is different with the elements being joined? I'm looking for 2 specific things...
# Hint: Don't space out on me...and...0, 1, 2, 3

# Or this way...

# Name_Area =
'{0} (Area: {1} SqMi)'.format(!NAME!, !AREA!)

# Or even better, this way...

# Name_Area =
f'!NAME! (Area: !AREA! SqMi)'

# Note the last two methods are the best practice...and that you do not need to convert the AREA to a string!
# More on this later...

```

## Calculating Fields Using Python (continued)

Now we would like to classify the median income levels.

### 1. Add a field with the following parameters:

- Field Name: **Income\_Level**
- Field Type: **Text**

### 2. Calculate the fields using the Python expression and code block below

### 3. When you have completed the calculation, open the result from the Geoprocessing History pane

- How does this look different?

```

In [ ]: """ Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# Expression:
# Income_Level =
income_level(!ACSMEDHINC!, 35000, 60000)

# Code Block:
def income_level(median_income, low_income, mid_income):
    """Classifies median income levels given low income and middle income values"""
    if median_income < low_income:
        return 'Low Income'
    elif low_income <= median_income < mid_income:
        return 'Middle Income'
    else:
        return 'High Income'

```

## Calculating Fields Using Python (continued)

Now we would like to know how many vertices are in each feature...just for fun!

We will use an example from the ArcGIS Help Documentation: [Calculate Field Python examples](#)

### 1. Calculate a field with the following parameters:

- Field Name: **VertexCount**

- Field Type: **Long Integer**

## 2. Calculate the fields using the Python expression and code block below

**Copy and Paste** the code below; it is for demonstrating that while loops can be used in the Field Calculator

```
In [ ]: """ Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool."""

# Expression:
# VertexCount =
VertexCount(!Shape!)

# Code Block:
def VertexCount(feat):
    partnum = 0

    # Count the number of points in the current multipart feature
    partcount = feat.partCount
    pntcount = 0

    # Enter while loop for each part in the feature (if a singlepart
    # feature this will occur only once)
    while partnum < partcount:
        part = feat.getPart(partnum)
        pnt = part.next()

        # Enter while loop for each vertex
        while pnt:
            pntcount += 1
            pnt = part.next()

        # If pnt is null, either the part is finished or there
        # is an interior ring
        if not pnt:
            pnt = part.next()
        partnum += 1
    return pntcount
```

## Future Challenge: Try using Python with map labels in ArcGIS

# V. Introduction to ArcPy

[Top](#)

## What is ArcPy?

[Top](#)

ArcPy is a Python site package that provides a useful and productive way to perform geographic data analysis, data conversion, data management, and map automation with Python.

This package provides a rich and native Python experience offering code completion (type a keyword and a dot to get a pop-up list of properties and methods supported by that keyword; select one to insert it) and reference documentation for each function, module, and class.

The additional power of using ArcPy is that Python is a general-purpose programming language. It is interpreted and dynamically typed and is suited for interactive work and quick prototyping of one-off programs known as scripts while being powerful enough to write large applications in. ArcGIS applications written with ArcPy benefit from the development of additional modules in numerous niches of Python by GIS professionals and programmers from many different disciplines.

From [ArcGIS Pro Help Documentation - What is ArcPy?](#)

## Python in ArcGIS

- First introduced at ArcGIS Desktop 9.0 with Python 2 and `arcgisscripting` site package
- ArcGIS Desktop 10 introduced `arcpy` site package
- ArcGIS Pro uses Python 3 `arcpy` site package
- ArcGIS Pro integrates [Conda](#) to manage Python packages and modules via the [Python Package Manager](#)
- ArcPy is the primary desktop Python site package designed exclusively for ArcGIS Desktop

- [ArcGIS API for Python](#) is designed for ArcGIS Online and ArcGIS Enterprise
  - We will not go over this API in this workshop

## What is a Python Module?

- Extensions that can be imported and used in Python scripts to expand the built-in capabilities
- Contain pre-written code to help out with specialized tasks and defines functions, classes and variables for those tasks
- Also allows you to logically organize your Python code
- Single file consisting of Python code and also includes executable code
- Modules are imported into a script using the `import` statement
- Many times you need to download and install new modules; but ArcGIS Pro includes many of the most popular ones
- Also called libraries

## What is a Python Site Package?

- Like a module, but contains a collections of modules, functions, and classes
- Site packages and modules both add functionality to Python

## `import` Statements

### Top

- We aren't able to use everything Python has to offer without importing modules
- Only need to import a module once in a script
- Customary to import everything at the beginning of your script

```
# Best practice is to stack each import statement
import arcpy
import os
import sys

# But this is still correct syntax (Remember the Style Guide!!)
import arcpy, os, sys

# This can sometimes make things simpler; you can use a.whatever instead of arcpy.whatever
import arcpy as a

# Do this if you only want a specific part of the site package/module
from arcpy import da

# This also works, but should be used sparingly
from arcpy.sa import *
```

## Importing ArcPy

- To work with ArcPy outside of ArcGIS Pro, you must `import` it
- Once imported, you can now use all the modules, functions, classes, methods, and geoprocessing tools

See [Importing ArcPy](#) for more information

```
In [ ]: # Recommended to import the ArcPy site package and other modules first in your script
# This may take a while if you are working outside of ArcGIS

import arcpy
```

## ArcPy Help Documentation

### Top

### Python and ArcPy Help

- [ArcGIS Python Help](#)
- Contains help on functions and classes not listed in the tool documentation
- Note the additional modules

## Python and ArcPy Geoprocessing Tool Help

- Each geoprocessing tool includes syntax and sample code for the Python window and stand-alone scripts
- Includes detailed explanations of each parameter
- See [Calculate Field](#) tool help as an example

## VI. Using ArcPy

---

[Top](#)

### Describing Data

[Top](#)

- Describing data allows to learn more about the data we are working with
- Describe functions are useful when scripts may be dependent on the type of data being used
- Good for controlling script flow and validating parameters
- Many property groups and some properties exist for only some types

See [ArcPy Function - Describe](#)

### System Paths vs. Catalog Paths

[Top](#)

- System paths are recognized by the Windows operating system
  - Files in folders
  - e.g., Each Shapefile component (.shp, .shx, .dbf, .prj), TIFF Raster components (.tif, .prj), Word Documents (.docx), Compressed files (.zip)
- Catalog paths are only recognized by ArcGIS
  - Used for feature classes and other data in geodatabases
    - e.g., Geodatabases (.gdb) not recognized as a folder, Shapefiles (.shp) recognized as one 'file', TIFF Raster (.tif) recognized as a raster file
  - Contain 2 parts:
    - Workspace - could be the geodatabase root or a feature dataset
    - Base name - the feature class, raster, or other file types that can be saved in geodatabases
- As an ArcGIS user and Python programmer, it is necessary to be aware of the context for how the path is being used

```
In [ ]: # Step 0: Get the ArcGIS Project Home (i.e. Default) folder path
# Also, think about relative vs. absolute paths!!

# Run this code in the Notebook and copy the path from output
# Tip: You can use the ArcGIS Notebook 'Find and Replace' to replace all '{...}' with the aprx home folder
# Tip: Copy Path in Catalog Pane!!! Always copy paths if you can!!!

import arcpy

aprx = arcpy.mp.ArcGISProject('CURRENT')

aprx_folder = aprx.homeFolder

print(aprx_folder)
```

The following code is intended to only be run in the Python Window of ArcGIS Pro

Type, do not copy (except for file paths), the code into the Python Window for the following exercises...This is so you can practice typing code.

Unhide the Python Window!!

```
In [ ]: # Exercise 1: Check if a file exists and what data type it is

file_path = r'C:\Python-for-ArcGIS\CSV\JeopardyContestants_LatLon.csv' # What kind of path is this? System or Catalog?
print(arcpy.Exists(file_path))

# What is the result?

# Try again, but this time type the up arrow to cycle through the last few lines of code

# Replace the '{...}' with the ArcGIS Pro Project Home folder Location
```



```

file_path = r'...\CSV\JeopardyContestants_LatLon.csv'
print(arcpy.Exists(file_path))

# Does it exist? Keep trying until you get the path correct and the result returns True

print(arcpy.Describe(file_path).dataType)

# What type of data is this?

```

```

In [ ]: # Exercise 2: Check if a feature class exists what data type it is

# Replace the '{...}' with the ArcGIS Pro Project Home folder Location

feature_class = r'...\PythonForArcGIS.gdb\Illinois_Counties' # What kind of path is this? System or Catalog?

print(arcpy.Exists(feature_class))

print(arcpy.Describe(feature_class).dataType)

# What type of data is this?

```

```

In [ ]: # Exercise 3: Check if a layer exists and data type it is

layer_name = 'Illinois Counties' # What kind of path is this? System or Catalog?

print(arcpy.Exists(layer_name))

lyr_desc = arcpy.Describe(layer_name)

# Note how this Describe function and variable look different than before...
# We can have describe objects become variables to be used later

print(lyr_desc.dataType)

# What type of data is this?

# For Layers, there is a Describe property called 'dataElement' that accesses
# the 'dataType' of what the Layer is referencing

print(lyr_desc.dataElement.dataType)

# What type of data is this?

```

What is the difference between the last two `print()` results?

Lets look at the ArcPy Documentaion:

- [Describe object properties](#)
- [Layer properties](#)

```

In [ ]: # Exercise 4: Get more Describe information about a Layer

lyr_datatype = lyr_desc.dataElement.dataType

lyr_path = lyr_desc.path

lyr_basename = lyr_desc.baseName

lyr_spatialref = lyr_desc.spatialReference.name

lyr_count = arcpy.management.GetCount(layer_name)

# You may see the code for the Get Count tool written as below too.
# This is the old style...but it is correct
lyr_count = arcpy.GetCount_management(layer_name)

# Here is a new way to format strings with inline variable substitution
print('{0} is a {1} stored at {2} with a file name of {3} with the {4} coordinate system and contains {5} features.'\
      .format(layer_name, lyr_datatype, lyr_path, lyr_basename, lyr_spatialref, lyr_count))

# The slash does not need to be typed, it is there for formatting purposes

# Here is another way to quickly format strings...this is my preferred method! Less code!!
print(f'{layer_name} is a {lyr_datatype} stored at {lyr_path} with a file name of {lyr_basename}\
      with the {lyr_spatialref} coordinate system and contains {lyr_count} features.')

# The slash does not need to be typed, it is there for formatting purposes

```

# Tool organization in ArcPy

Geoprocessing tools are organized in two different ways. All tools are available as functions on ArcPy but are also available in modules matching the toolbox alias name. Although most of the examples in the help show tools organized as functions available from ArcPy, both approaches are equally valid. Which approach you use will come down to a matter of personal preference and coding habits. In the following example, the Get Count tool is shown using both approaches.

From [Using tools in Python - Tool organization in ArcPy](#)

## Formatting Strings

[Top](#)

- Formatting strings pretty slick
- Can save space and time

See [PyFormat](#) and [Python 3's f-Strings: An Improved String Formatting Syntax \(Guide\)](#)

## Listing Data

[Top](#)

- Helps with batch processing (primary reason for developing scripts)
- Lists are good for integrating processes using loops (another primary reason for developing scripts)
- Easy inventory of data (e.g., list of fields in a table; feature classes in a geodatabase)
- In order to list data with some ArcPy List functions, the Geoprocessing Environment Settings class Workspace property needs to be set

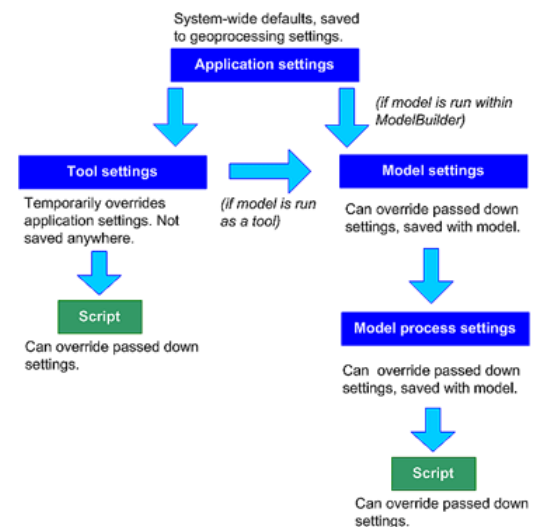
## Geoprocessing Environment Settings

[Top](#)

- Additional parameters that influence how tools run and/or their results
- Corresponds to the 'Environments' found in geoprocessing tool dialog
- Important for controlling certain aspects of geoprocessing tool outputs
- Available in Python through the `arcpy.env` class properties that must be set in the code
- Follows a hierarchy within ArcGIS for how geoprocessing environment settings are implemented (See image)
  - Geoprocessing environment settings set in scripts **override** all other environment settings

See:

- [What is a geoprocessing environment setting?](#)
- [Using environment settings in Python](#)
- [arcpy.env Class](#)



The following code is intended to only be run in the Python Window of ArcGIS Pro

Type, do not copy (except for file paths), the code into the Python Window for the following exercises

```
In [ ]: # Exercise 1: List Files and Feature Classes

# First, we must set the environmental settings

# Replace the '{...}' with the ArcGIS Pro Project Home folder Location
# Tip: Copy as Path!!

arcpy.env.workspace = r'{...}\CSV'

print(arcpy.ListFiles())

# How many files are Listed?
```

```

arcpy.env.workspace = r'{...}\PythonForArcGIS.gdb'

print(arcpy.ListFiles())

print(arcpy.ListFeatureClasses())

# What is the difference between the two List functions?

```

```

In [ ]: # Exercise 2: List fields

# Note that the environment setting is not needed!

print(arcpy.ListFields('Illinois Counties'))

# What is the result? Can you understand it?
# What is actually being output in this case?
# (Hint: What type/class of object is being displayed?)

fields = arcpy.ListFields('Illinois Counties')
print(type(fields[0]))

# We can print each field name...
for field in fields:
    print(field.name) # '.name' is property of the Field class

# Or the field aliases...
for field in fields:
    print(field.aliasName) # '.aliasName' is another property of the Field class

```

```

In [ ]: # Exercise 3: Create a list of field names

# We can also create a list of the field names using a 'list comprehension'...they are VERY powerful

field_list = [field.name for field in arcpy.ListFields('Illinois Counties')]

print(field_list)

# How does this result look differently than above in Step 2?

test_field = 'OBJECTID'

if test_field in field_list:
    print('{} exists'.format(test_field))
else:
    print('{} does not exist'.format(test_field))

# Repeat the above code using a test field that you know doesn't exist to test the 'else:' statement code
# Does it work?

```

## List Comprehensions

### Top

- Used for creating Python lists concisely and quickly
- Consists of brackets containing an expression followed by a `for` clause
- Optionally can have one or more `if` or `for` clauses
- Result will be a new list resulting from evaluating the expression in the context of the `for` and `if` clauses
- Always returns a list as a result

See:

- [ArcPy ListFields](#)
- [Leran Python: List Comprenensions](#)
- [Python For Beginners: List Comprehensions in Python](#)

**\*Once you understand these, you are on your way to becoming a Python ninja!\***

## Geoprocessing Tools

### Top

- ArcPy has access to all geoprocessing tools in ArcGIS Pro in addition to many non-tool functions
- Geoprocessing tools have a fixed set of parameters that provide the tool with the information required for execution
- Tools usually have input parameters that define the dataset or datasets that are typically used to generate new output data

## Tool Parameters

- Parameters have several important properties:
  - Each parameter expects a specific data type or data types, such as feature class, integer, string, or raster
  - A parameter expects either an input or output value
  - A parameter either requires a value or is optional
  - Each tool parameter has a unique name
- When a tool is used in Python, its parameter values must be correctly set so it can execute when the script is run
- Once a valid set of parameter values is provided, the tool is ready to be executed
- Parameters are specified as either strings or objects.

See [Using tools in Python](#)

The following code is intended to only be run in the Python Window of ArcGIS Pro

Type, do not copy (except for file paths), the code into the Python Window for the following exercises

```
In [ ]: # Exercise 1: Select Layer by Attribute
import arcpy

layer_name = 'Illinois Counties'

expression = 'ACSMEDHINC <= 40000'

arcpy.management.SelectLayerByAttribute(layer_name, 'NEW_SELECTION', expression)

# Now clear the selection
arcpy.management.SelectLayerByAttribute(layer_name, 'CLEAR_SELECTION')

# Note the difference in how the code is written...See note below
```

```
In [ ]: # Exercise 2: Run some geoprocessing tools

# Ensure that a default geodatabase is set

# Replace the '{...}' with the ArcGIS Pro Project Home folder Location
# Don't forget: Copy as Path!!

arcpy.env.workspace = r'{...}\PythonForArcGIS.gdb' # What does setting the environment here do?

# Create a variable for dissolve fields
dissolve_fields = ['STATE_NAME', 'ST_ABBREV']

# New drag, drop, and roll trick...using tools from the toolbox!

arcpy.management.Dissolve(layer_name, 'Illinois', dissolve_fields, '', 'SINGLE_PART', 'DISSOLVE_LINES')

arcpy.management.PolygonToLine('Illinois', 'Illinois_Boundary')

# You may want to rearrange your layers in the TOC
```

```
In [ ]: # Step 3: Run a series of Geoprocessing tool

# Replace the '{...}' with the ArcGIS Pro Project Home folder Location

csv_file = r'{...}\CSV\JeopardyContestants_LatLon.csv'

arcpy.management.CopyRows(csv_file, 'JeopardyContestants_Table')

arcpy.management.MakeXYEventLayer('JeopardyContestants_Table', 'lon', 'lat', 'Jeopardy Contestants')

# Note the quotes!!
arcpy.analysis.Select('Jeopardy Contestants', 'JeopardyContestants',
                      '"lat" IS NOT NULL OR "lon" IS NOT NULL')

# Note the indent!!
arcpy.analysis.Buffer('JeopardyContestants', 'JeopardyContestants_Buffer',
                      '5 Miles', 'FULL', 'ROUND', 'ALL', '', 'GEODESIC')
```

## Your turn:

### 1: Write code to clip the buffer to the Illinois state layer

Be careful with which tool you choose...there are few tools named 'Clip...'

### 2: Write code to intersect the new buffer to the Illinois Counties layer

You might need to check the tool's help documentation...

### 3: Save your code as a Python file (.py) in a new 'Scripts' folder

Open the .py file and view it in a text editor (like Notepad++)

You may need to clean up the script for any mistakes or extraneous code (look back at your errors...and delete!)

### 4. Load the Python file back into the Python window

Clear the Python transcript in ArcGIS

Try running the whole script at once

This is one way you can create, save, and reuse a script for use again!

## VII. Conclusion

---

[Top](#)

Python is simple to use and easy to learn!

The only way to get better is to practice and challenge yourself

Utilize Python and ArcGIS help documentation and forums

There is so much more to learn!!

```
In [ ]: print('Have a great lunch. Ta ta for now!')
```