Introduction to Python for ArcGIS

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 attendees 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

Computing and Software Needs

- Your may use your own laptop, however computers are available in the lab
- ArcGIS Desktop 10.x (Standard or Advanced preferred) or ArcGIS Pro 1.4+
- Python IDE
  - IDLE (https://docs.python.org/2/library/idle.html) is also shipped with ArcGIS for Desktop, so it can be used.
  - PyScripter (http://sourceforge.net/projects/pyscripter/files/) (v. 2.6.0 **32-bit** version) is good for ArcMap script development *NOTE:* The zip files contain portable versions of PyScripter. No installation is needed. Just unzip the archive and start using PyScripter.
  - Spyder IDE (https://github.com/spyder-ide) is better for ArcGIS Pro development, however this requires special installaiton instuctions to make it work with ArcGIS, so do not install this prior to the workshop; I can deal with this on a case-by-case basis

Instructor

James Whitacre, GIS Specialist, University Library, University of Illinois at Urbana-Champaign

As the GIS Specialist in the Main Library at the University of Illinois at Urbana-Champaign, Mr. Whitacre's primary role is to provide GIS consultation and research assistance for faculty, staff, and students. Additionally, he teaches a myriad of GIS workshops for beginner to advanced users and helps manage the Library's GIS data and software assets. He is also a central resource for the GIS community on campus to promote the use of GIS in research. Mr. Whitacre holds a Master of Science in Geography and was previously the GIS Manager for the Carnegie Museum of Natural History.

I. Introductions and Outline

Outline

Data and Software Setup

What is Python?

Python Basics

- Print Statement
- Variables
- Basic Data Types
  - Strings
  - Numbers
  - Booleans
  - Lists
  - Tuples
  - Dictionaries
- Data Type Conversions
- Simple Math with Python
- Python Basic Syntax
- Conditional Statements
- Functions
- Loops

Calculate Fields Using Python

Break

Introduction to ArcPy

- What is ArcPy?
- Modules vs. Site Packages
- `import` Statements
- Utilizing the ArcPy ArcGIS Desktop Help Documentation

Using ArcPy in the ArcGIS Python Window

- The Python Window
- Describing Data with ArcPy
  - System Paths vs. Catalog paths
- Listing Data with ArcPy
  - Environmental Settings

    - List comprehensions
- Geoprocessing Tools with ArcPy

Conclusion

II. Data and Software Setup

Download Exercise Data

GitHub: https://github.com/whitacrej/Python-For-ArcGIS-2017 (https://github.com/whitacrej/Python-For-ArcGIS-2017)

BOX: https://uofi.box.com/PythonForArcGIS (https://uofi.box.com/PythonForArcGIS)

- Extract the data to a folder on the Desktop or other well-known folder
- For folks who can't download the files, we have it on a flash drive

ArcGIS vs. ArcMap vs. ArcGIS Pro

- ArcGIS, ArcGIS Desktop, or Desktop = Both ArcMap AND ArcGIS Pro
- ArcMap = ArcMap
- ArcGIS Pro or Pro = ArcGIS Pro

Integrated Development Environment (IDE)

- A software application that provides comprehensive facilities to computer programmers for software development (see https://en.wikipedia.org/wiki/Integrated_development_environment (https://en.wikipedia.org/wiki/Integrated_development_environment))
- Already using a Python IDE? If you feel comfortable in your own envirionment, please use it, but there is no guarantee that issues can be supported
- If all esle fails, you may need to use IDLE, which ships with ArcGIS Desktop and is installed automatically

Python IDE: IDLE

- Already installed as a part of ArcGIS Destkop Install
- Let's take a quick tour!
  - Two windows:
    - Python shell window (interactive interpreter)
    - Text Editor window (for scripts)

Python IDE: PyScripter

- Download PyScripter (http://sourceforge.net/projects/pyscripter/files/) (v. 2.6.0 **32-bit** version)

- Open PyScripter: Two Ways...
  - Install it like a normal program
    - Download **PyScripter-v2.6.0-Setup.exe**, run the install file, open program
    - Admin privileges needed
  - Run the portable version
    - Download **PyScripter-v2.6.0.zip**, extract, and run **PyScripter.exe**
    - Admin privileges *NOT* needed
- Let's take a quick tour!
  - Editor
  - Code explorer
  - Interpreter
  - Messages
  - Add line numbers: Tools > Options > Editor Options
  - Change the theme: View > Themes > Choose your favorite!!

Type the code below into your IDE and run it....

- I will give an explanation of concepts, then have code examples that can be run in the IDE in the grey boxes.
- Please ***type*** the code as we go and try to ***avoid copy and pasting*** unless instructed otherwise. Typing the code will help you learn it better!
- When typing in the IDE, you will likely notice the auto complete functionality. Mastering this will help type code faster and with less mistakes**.** I will point out tips as we go.
- ***Congratulations***, this is your first code!!!

```
In [ ]: print('hello world')
```

III. What is Python?

- 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 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 (https://www.python.org/doc/essays/blurb/) for more information

Also see http://desktop.arcgis.com/en/arcmap/latest/analyze/python/what-is-python-.htm (http://desktop.arcgis.com/en/arcmap/latest/analyze/python/what-is-python-.htm)

Some General Notes

- 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'*
- You may feel lost at first...practice and perseverence will help (I will try to go slow!)
- Just like in ArcGIS, there is more than one way to do many things!

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.

IV. Python Basics

Print Statement

- What is print statement?
  - A way to make your script talk back to you
  - A way to see what a variable is

- How to use the print statement
  - Type `print` or `print()`*
  - Add the variable or string after `print` or within the parentheses `()`

*Using just `print` without `()` is proper syntax in Python 2.x, but getting in the habit of usng `print()` will prepare you for Python 3.x and ArcGIS Pro, which is the only way `print()` will work. We will use `print()` throughout this workshop

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

In [ ]:
```
print('Python is so cool!')
```

Variables

- 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 [ ]:
```
food = 'cheese'

food_count = 6

print(food)
print(food_count)
```

In [ ]:
```
# cool tips...

food, food_count = 'bread', 100

print(food)
print(food_count)

food1 = food2 = food3 = 'banana'

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

Basic Data Types

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

- 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 a 0)

In [ ]:
```
print('cheese ' + 'whiz') # Note the space...
print('cheese' * 3) # Note no space...
print('cheese'[1:4])
print('cheese'[:2])
print('cheese'[:-2])
print('cheese'[2:])
print('cheese'[-2:])
```

Numbers

- 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 precsision.

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

three = 3

print(5 + 7 - three)
```

Booleans

- 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. `{}`)

In [ ]:
```
print(True)
print(False)

a = True
b = False

print(a)
print(b)
```

In [ ]:
```
y = 0 # Run it once to evaluate, then change to 1 and run again; change to []
 and run again

if y:
    print(True)
else:
    print(False)
```

Lists

- What is a list?
  - Series of ordered items or objects
  - Compund 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

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

  - Iterate through it (use a loop)

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

  for fruit in fruitlist:
  # Do somthing...
      print(fruit)
```

  - Change list values

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

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

See https://www.tutorialspoint.com/python/python_lists.htm (https://www.tutorialspoint.com/python/python_lists.htm)

In [ ]:
```
fruitlist = ['apples', 'oranges', 'bananas']

print(fruitlist)

print(fruitlist[1])

print(fruitlist[1:3])

print(len(fruitlist))
```

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

In [ ]:
```
# Change the list...
fruitlist = ['apples', 'oranges', 'bananas']

fruitlist[1] = 'peaches'
fruitlist.append('cherries')
fruitlist.remove('apples')
del fruitlist[1]

print(fruitlist)
```

Tuples

- 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[1:3] # returns (oranges, bananas)
```

  - Iterate through it (use a loop)

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

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

Lists are generally more used than tuples in ArcGIS, and many times they can be used Interchangeably, so it is good to know about them in case they pop up.

In [ ]:
```
fruittuple = ('apples', 'oranges', 'bananas')

print(fruittuple)
print(fruittuple[1])
print(fruittuple[1:3])
```

In [ ]:
```
fruittuple = ('apples', 'oranges', 'bananas')

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

In [ ]:
```
fruittuple[1] = 'peaches' # Invalid syntax with tuple, this will produce an er
ror, but it will work for a list
```

Dictionaries

- What is a dictionary?
  - An *unordered* set of `key:value` pairs enclosed by curly brackets `{}`
  - **Keys** must be unique values and are typically numbers or strings in quotes, but can be other Python data types
  - **Values** can be any Python object and do not require quotes
- How are dictionaries used?
  - Find the value that goes with a key

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

  - Get a list of keys

```
dicttest.keys() # returns ['key', 'lat', 'long']
```

  - Get a list of values

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

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

In [ ]:
```
dicttest = {'key':'value', 'lat':39.98, 'long': -89.65}

print(dicttest['key'])
print(dicttest['lat'])
print(dicttest['long'])

print(dicttest.keys())

print(dicttest.values())
```

Data Type Conversion

- 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.")

# Change x to be str(x)
```

In [ ]:
```
x = 99
s = "99"
l = [s, x]
t = (s, x)

print(type(x))
print(type(s))
print(type(l))
print(tuple(l))
print(type(t))
print(list(t))
```

Simple Math with Python

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: [http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/calculate-field-examples.htm (http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/calculate-field-examples.htm)](http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/calculate-field-examples.htm)

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 [ ]:
```
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 Basic Syntax

- 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:

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

  # 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!
```

- Variable name capitalization matters!!

```
Cat != cat
```

- Indentation matters!! AND...
- Colons matter!!

```
if x
  print(x)

  # Will not work, but this will:
  if x:
  ....print(x)

  # typically 2 or 4 spaces (represented by '.')
  # 4 spaces are prefered (using Tab in the IDE should do this automaticall
y)
```

- 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 strip denotation with the same type of quote structure
  - Single quotes (') are the easiest to type (no Shift!!), so I default to them usually

```
# 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...
  'C:\\data\\things' # I prefer this one when working in ArcGIS Pro...I will
 tell you why later
  'C:/data/things'
  r'C:\data\things' # I prefer this one when working in ArcMap...I will tell
 you why later
```

- Commenting is great!!
  - Use it to help document and explain your code...we will do this throughout the 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 commnets as mush 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 i
    n your code
```

See the Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/) for more information

Conditional Statements and Decision Making

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

Source: https://www.tutorialspoint.com/python/python_decision_making.htm (https://www.tutorialspoint.com/python/python_decision_making.htm)

`if` Statements

- `if` statement - one decision/option

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

      # 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! Just unindent to show the end of the section.

Comparison Operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(1 == 2) is NOT true
!=	If values of two operands are not equal, then condition becomes true.	(1 != 2) is true
<>	If values of two operands are not equal, then condition becomes true. *This is the same as != operator, but is deprecated and not valid in Python 3.*	(1 <> 2) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(1 > 2) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(1 < 2) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(1 >= 2) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(1 <= 2) is true.
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true
not	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.	not(a && b) is false
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y
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: https://www.tutorialspoint.com/python/python_basic_operators.htm (https://www.tutorialspoint.com/python/python_basic_operators.htm)

In [ ]:
```
a = 21
b = 10
c = 0

if a == b:
    print("Line 1 - a is equal to b")
else:
    print("Line 1 - a is not equal to b")

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

# 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")

if a < b:
    print("Line 4 - a is less than b")
else:
    print("Line 4 - a is not less than b")

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;
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")

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")
```

Functions

- Block of organized, reusable code
- Performs a single, related action
- Good when a funcion 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

Defnining Funcitons

- 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 Funtion syntax

    def functionname( parameters ):
        """function_docstring""" # This is optional, note triple-quotes
        function_suite
        return [expression]
```

Source: https://www.tutorialspoint.com/python/python_functions.htm (https://www.tutorialspoint.com/python/python_functions.htm)

Script Source: https://www.codecademy.com/learn/python (https://www.codecademy.com/learn/python)

In [ ]:
```
def tax(bill):
    """Calculates 8% tax of restaurant bill."""
    tax = bill * 0.08
    print('Tax: ${:0.2f}'.format(tax))
    return tax

def tip(bill):
    """Calculates 15% tip of restaurant bill."""
    tip = bill * 0.15
    print('Tip: ${:0.2f}'.format(tip))
    return tip

def totalbill(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
print('Bill: ${:0.2f}'.format(meal_cost))

meal_tax = tax(meal_cost)
meal_tip = tip(meal_cost)
meal_total = totalbill(meal_cost, meal_tax, meal_tip)
```

Loops

- 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 https://www.tutorialspoint.com/python/python_loops.htm (https://www.tutorialspoint.com/python/python_loops.htm)

`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 [ ]:
```
a_list = ['a', 'b', 'c', 'd']

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

`while` Loops

- Executes the code block while a gived 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 https://www.tutorialspoint.com/python/assignment_operators_example.htm (https://www.tutorialspoint.com/python/assignment_operators_example.htm) for other similar operations. Python is full of these types of tricks!

In [ ]:
```
i = 0

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

V. Calculate Fields Using Python

- **A great place to practice Python in ArcGIS is by calculating fields**
- Python and VB Script can be used to calculate fields
- VB Script is the default, so Python needs to be set as the parser
- Simple expressions use the Expression box
- More complex calculations can use the Pre-Logic Script Code (window) or Code Block (tool) boxes
- **Note: Python can be difficult to debug and check for errors when calculating fields**

Using Python in the Field Calculator Window

1. Open the *Illinois.mxd* ArcMap document *or* Illinois.aprx ArcGIS Project File

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. Add a field with the following parameters:

Name: AreaTest
Type: Double

4. Right-click on the new field header and select Field Calculator

5. Double click any field to add it to the expression box

Make sure the VB Script radio button is selected

6. Select the Python radio button and add the same field

What is different?
Clear the expression box when done

7. Click on the 'About calculating fields' link at the bottom left

Notice the different examples
Find the 'Code samples-geometry' section
These expressions can be used in lieu of the 'Calculate Geometry' window accessed t
hrough the attribute table
These expressions can also be used in the Calculate Field tool

8. Enter the following code in the expression box

Note: In ArcGIS Pro, Python becomes the default coding language for calculating fields and the above steps do not apply

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 vales compared to the AREA field?


# Second, try square miles

!Shape.area@SQUAREMILES!

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


# Now try

!Shape.geodesicArea@SQUAREMILES!

# Which method is more acurate? 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!
```

Field Calculator (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. Delete the 'AreaTest' field

Click 'Yes' on the warning

2. Add two fields with the following parameters:

Name: DensitySqmi
Type: Double
Alias: Density Per SqMi

Name: DiffIncome
Type: Double
Alias: Avergae vs. Median Income

3. Calculate the field using Python

Clear the expression box before each calculation
Make sure the Python radio button is selected
Enter the code 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. """

# DensityPerSqmi calculation

!ACSTOTPOP! / !AREA!

############################################################

# DiffIncome calculation

!ACSAVGHINC! - !ACSMEDHINC!
```

Field Calculator Window (continued)

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

1. Add two fields with the following parameters:

```
Field 1:
    Name: NameLong
    Type: Text
    Alias: Long Name
    Length: 100

Field 2:
    Name: NameShort
    Type: Text
    Alias: Short Name
    Length: 50

Field 1:
    Name: NameArea
    Type: Text
    Alias: Name (with Area)
    Length: 100
```

2. Calculate the fields using Python

```
Clear the expression box before each calculation
Make sure the Python radio button is selected
Enter the code 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. """

# NameLong calculation

!NAME! + ', ' + !STATE_NAME!

############################################################

# NameShort calculation

!NAME!.replace(' County', '')

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

############################################################

# NameArea calculation

!NAME! + ' (Area: ' + !AREA! + ' SqMi)'

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


# NameArea can also be calculated this way

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

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

Field Calculator Window (continued)

Now we would like to classify the median income levels.

1. Add a field with the following parameters:

Name: IncomeLevel
Type: Text
Alias: Income Level
Length: 50

2. Calculate the fields using Python

Clear the expression box
Make sure the Python radio button is selected
Check the 'Show Codeblock' box
Enter the code below

3. When you have completed the calculation in the Field Calculator window, open the result from the Results window

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. """

# When calulating a field, a function usually is used in the
# Pre-Logic Script Code:

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


# Expression:

income_level(!ACSMEDHINC!, 35000, 60000)
```

Field Calculator Window (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 Desktop help for Calculate Field examples (http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/calculate-field-examples.htm#ESRI_SECTION1_C9E46547D3B246C1B89A36B0C84F7BA8). This page is a great resource.

1. Add a field with the following parameters:

```
Name: VertexCount
Type: Long Integer
Alias: Number of Vertices
```

2. Calculate the fields using Python

```
Clear the expression box
Make sure the Python radio button is selected
Make sure to check the 'Show Codeblock' box
```

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."""

# Pre-Logic Script Code:
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


# Expression:

VertexCount(!Shape!)
```

Future Challenge: Try using Python with map labels in ArcGIS

Break

VI. Introduction to ArcPy

What is ArcPy?

> ArcPy is a site package that builds on (and is a successor to) the successful arcgisscripting module. Its goal is to create the cornerstone for 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 within Python is the fact 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 http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy/what-is-arcpy-.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy/what-is-arcpy-.htm)

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 include runnable code
- Modules are imported into a script using the **`import`** statement
- Many times you need to download and install new modules

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

- 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

```
import arcpy
  import os
  import sys

  # Best practice is to stack each import statement as above, but this is no
t wrong:
  import arcpy, os, sys

  import arcpy as a # This can sometimes make things simpler; you can use a.
whatever instead of arcpy.whatever
  from arcpy import da # Do this if you only want a specific part of the sit
e package/module
  from arcpy.sa import * # This also works, but should be used sparingly
```

Importing ArcPy

- To work with ArcPy, you must `import` it
- Once imported, you can now use all the modules, functions, tools, and classes

See http://desktop.arcgis.com/en/arcmap/latest/analyze/python/importing-arcpy.htm (http://desktop.arcgis.com/en/arcmap/latest/analyze/python/importing-arcpy.htm) for more information

In [ ]:
```
# Always 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
```

Utilizing the ArcPy ArcGIS Desktop Help Documentation

Desktop vs. Python Window vs. Online Documentation

- Desktop help can be opened through the Help menu or from the tool interface
- Python window help and syntax panel on the right (more on this later)
- ArcGIS Desktop Online Documentation: http://desktop.arcgis.com/en/documentation/ (http://desktop.arcgis.com/en/documentation/)

ArcPy Help

- http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy/what-is-arcpy-.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy/what-is-arcpy-.htm)
- Contains help on functions and classes not listed in the tool documentation
- Note the additional modules

Understanding Geoprocessing Tool Help with Python and ArcPy

- Includes tool syntax and sample codes for the Python window and stand-alone scripts
- Includes detailed explanations of each parameter

VII. Using ArcPy in the ArcGIS Python Window

Another great place to practice Python in the Python Window in ArcGIS

1. Download exercise data

GitHub: https://github.com/whitacrej/Python-For-ArcGIS-2017 (https://github.com/whitacrej/Python-For-ArcGIS-2017)

BOX: https://uofi.box.com/PythonForArcGIS (https://uofi.box.com/PythonForArcGIS)

- You should have already done this...but just is case you didn't

2. Open the 'Illinois.mxd' ArcMap Document *or* Illinois.aprx ArcGIS Project File

3. Open the Python Window in ArcGIS

- Click the Python widow button and dock it
- Allows for running geoprocessing tools while also taking advantage of other Python modules and librarires
- Can be used for single-line code (e.g. Used instead of ArcToolbox to access tools)
- Can be used for testing syntax and longer, more complex scripts that might be used for automating workflows
- Replaces the Command Line from earlier releases of ArcGIS (pre-10.x)

The Python Window (ArcMap)

- Left side is the interactive Python interpreter panel
  - This is where code is entered
  - Note the three greater-than symbols (>>>)
- Right side is the help and syntax panel
- Code is generally executed one line at a time and displayed immediately. Exceptions include:
  - Multi-line constucts (such as `if` statements)
  - Pressing SHIFT or CTRL + ENTER at the end of a line of code (you may need to hit ENTER a few times when you are ready to run the code)
- Other Advanatages of the Python window
  - Autocompletion
  - Conditional and Iteration executuion
  - Scripts can be saved and reused or opened with another Python IDE

See http://desktop.arcgis.com/en/arcmap/latest/analyze/executing-tools/what-is-the-python-window-.htm (http://desktop.arcgis.com/en/arcmap/latest/analyze/executing-tools/what-is-the-python-window-.htm) for more info

Describing Data with ArcPy

- 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 valdating parameters
- Many property groups and some properties exist for only some types

See ArcGIS Documentation: http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy-functions/describe.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy-functions/describe.htm)

System Paths vs. Catalog paths

- System paths are recognized by the Windows operating system
  - Files in folders
  - Ex. Shapefiles and rasters
- Catalog paths are only recognized by ArcGIS
  - Used for feature classes and other data in geodatabases
  - Geodatabases could be file (.gdb), personal (.mdb) or enterprise (.sde)
  - Contain 2 parts:
    - Workspace - could be the geodatabase root or a feture dataset
    - Base name - the feature class, raster, or other file types that can be saved in geodatabases
  - Requires the programmer to be aware of the context inwhich the path is being used

Type, do not copy, the following code into the Python window

Note: The following code is intended to only be run in the Python window of ArcGIS

In [ ]:
```
# Step 1:
import arcpy

file_name = r'C:\Data\CSV\JeopardyContestants_LatLon.csv'
print(arcpy.Exists(file_name))

# What kind of path is this? System or Catalog?
# What is the result?


# Try again, this time, but type the up arrow to reload the last code
# Replace the '...' with the location where you placed your downloaded data
# I also have a cool trick...drag, drop, and roll!

file_name = r'C:\...\Data\CSV\JeopardyContestants_LatLon.csv'
print(arcpy.Exists(file_name))

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


print(arcpy.Describe(file_name).dataType)

# What type of data is this?
```

In [ ]:
```
# Step 2:

# Replace the '...' with the location where you placed your downloaded data

feature_class = r'C:\...\Data\Illinois.gdb\Illinois_Counties'
# What kind of path is this?

print(arcpy.Exists(feature_class))

print(arcpy.Describe(feature_class).dataType)
# What type of data is this?
```

In [ ]:
```
# Step 3:

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 statements?

Lets look at the help:

- http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy-functions/describe-object-properties.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy-functions/describe-object-properties.htm)
- http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy-functions/layer-properties.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/arcpy-functions/layer-properties.htm)

In [ ]:
```
# Step 4:

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.GetCount_management(layer_name)

# Here is a new way to format strings with inline variable substitution

print('{} is a {} stored at {} with a file name of {} with the {} coordinate s
ystem and contains {} 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
```

Formatting Strings

- Formatting strings pretty slick
- Can save space and time

*See https://pyformat.info/ (https://pyformat.info/) and https://docs.python.org/2/library/string.html#custom-string-formatting (https://docs.python.org/2/library/string.html#custom-string-formatting)

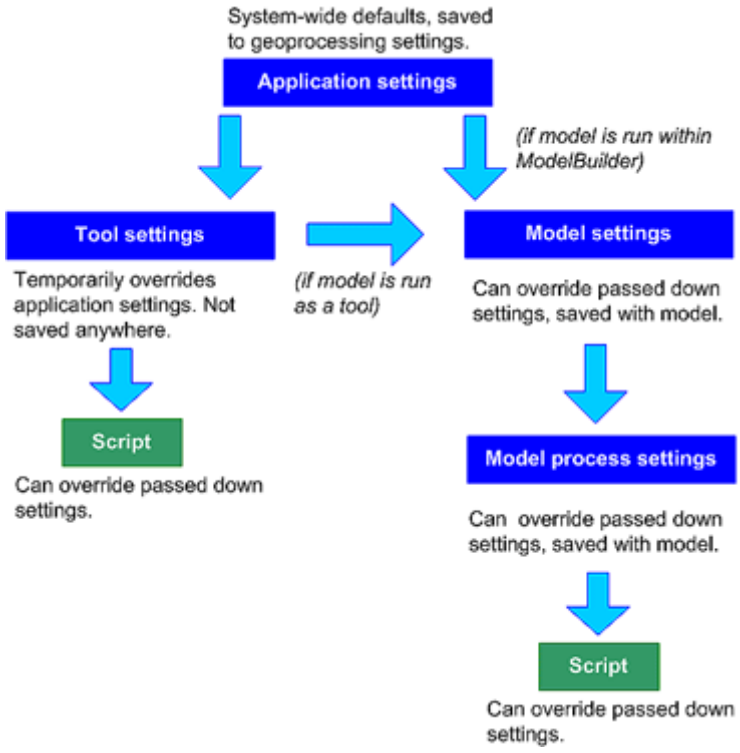
Listing Data with ArcPy

- Helps with batch processing (primary reason for developing scripts)
- Lists are good for interating 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, we must utilize the environmental settings class

Environmental Settings

http://desktop.arcgis.com/en/desktop/latest/tools/environments/what-is-a-geoprocessing-environment.htm (http://desktop.arcgis.com/en/desktop/latest/tools/environments/what-is-a-geoprocessing-environment.htm)

- Hidden parameters that influence how tools run
- Fundamental in controlling the geoprocessing workflow
- Exposed as properties and set using the **env** class
- Corresponds to the 'Environments...' found in tools
- There is a hierarchy with how ArcGIS deals with environmental settings



See http://desktop.arcgis.com/en/desktop/latest/tools/environments/environment-levels-and-hierarchy.htm (http://desktop.arcgis.com/en/desktop/latest/tools/environments/environment-levels-and-hierarchy.htm)

Type, do not copy, the following code into the Python window

Clear the Python window

Note: The following code is intended to only be run in the Python window of ArcGIS

In [ ]:
```
# Step 1:

# First, we must set the environmental settings
# Replace the '...' with the location where you placed your downloaded data
# Don't forget...drag, drop, and roll!
arcpy.env.workspace = r'C:\...\Data\CSV'

print(arcpy.ListFiles())

# How many files are listed?


arcpy.env.workspace = r'C:\...\Data\Illinois.gdb'

print(arcpy.ListFiles())

print(arcpy.ListFeatureClasses())

# What is the difference between the two list functions?
```

In [ ]:
```
# Step 2:

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

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

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

# We can print each field name...
for field in fields:
    print(field.name)

# Or the field aliases...
for field in fields:
    print(field.aliasName)
```

In [ ]:
```
# Step 3:
# We can also create a list of the field names; this is called a 'list compreh
ension' and 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?

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

- Used for creating list concisely and quickly
- Consists of brackets containing an expression followed by a for clause
- Optinally 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 result list

See:

- http://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-functions/listfields.htm (http://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-functions/listfields.htm)
- http://www.learnpython.org/en/List_Comprehensions (http://www.learnpython.org/en/List_Comprehensions)
- http://www.pythonforbeginners.com/basics/list-comprehensions-in-python (http://www.pythonforbeginners.com/basics/list-comprehensions-in-python)

Geoprocessing Tools with ArcPy

- ArcPy gives access to all tools in ArcToolbox in addition to many non-tool functions
- Working in the Python window allows for testing ideas and seeing how things work
- Tested scripts can also be saved for reuse in the Python window or in an IDE

Type, do not copy, the following code into the Python window

Clear the Python window

Note: The following code is intended to only be run in the Python window of ArcGIS

In [ ]:
```
# Step 1:
import arcpy

layer_name = 'Illinois Counties'

expression = 'ACSMEDHINC <= 40000'

arcpy.SelectLayerByAttribute_management(layer_name, 'NEW_SELECTION', expressio
n)

# Now clear the selection
arcpy.SelectLayerByAttribute_management(layer_name, 'CLEAR_SELECTION')
```

In [ ]:
```
# Step 2:

# Ensure that a default geodatabase is set
arcpy.env.workspace = r'C:\...\Data\Illinois.gdb'

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

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

arcpy.Dissolve_management(layer_name, 'Illinois', dissolve_fields, '', 'SINGLE
_PART', 'DISSOLVE_LINES')

arcpy.PolygonToLine_management('Illinois', 'Illinois_Boundary')

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

In [ ]:
```
# Step 3:

# Replace the '...' with the location where you placed your downloaded data

csv_file = r'C:\...\Data\CSV\JeopardyContestants_LatLon.csv'

arcpy.CopyRows_management(csv_file, 'JeopardyContestants_Table')

arcpy.MakeXYEventLayer_management('JeopardyContestants_Table', 'lon', 'lat',
'Jeopardy Contestants')

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

arcpy.Buffer_analysis('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 work as a text file (.txt) in the 'Scripts' folder (ArcMap only)

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

4: Save your work as a Python file (.py) in the 'Scripts' folder

After saving, open the .py file in PyScipter or other IDE
You may need to clean up the script for any mistakes or extraneous code (look back
 at your errors...and delete!)

5. Load the Python file back into the Python window

Clear the Python window in ArcGIS
Try running the whole script at once
This is one way you can create, save, and reuse a script for use again!