Pre-Workshop Python Basics

Description

Python programming tools are now a standard feature within GIS software packages and allow GIS users to automate, speed up, and become more robust in their data management and analytic work. This **Pre-Workshop Module** is designed for GIS users who do not have much experience with Python programming to help them get up to speed on the basic Python concepts that will be covered in the Python for ArcGIS Workshop. This module should take about 1-2 hours to complete.

NOTE: Basic Python programming will not be covered in the Python for ArcGIS Workshop, so participants are expected to know some basic Python concepts such as syntax, variables, data types (strings, numbers, Booleans, lists, tuples, dictionaries, etc.) conditional statements, functions, and loops. These concepts are all covered below

Instructor

James Whitacre

GIS Research Scientist, Carnegie Museum of Natural History, Powdermill Nature Reserve

whitacrej@canregiemnh.org

James Whitacre is the GIS Research Scientist for the Carnegie Museum of Natural History where he manages the GIS Lab at Powdermill Nature Reserve, the Museum's environmental research center, and supports museum staff and affiliated researchers with geospatial technologies and needs. This is Whitacre's second appointment at the Museum as he was formerly the GIS Manager from 2011 to 2014. 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.

Outline

[I. Data and Software Setup](#)

[II. What is Python?](#)

[III. 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](#)

I. Data and Software Setup

Computing and Software Needs

1. ArcGIS Pro 2.5.x+ or ArcGIS Desktop 10.4.x+ (Standard or Advanced preferred)

- **ArcGIS Notebooks (https://pro.arcgis.com/en/pro-app/arcpy/get-started/pro-notebooks.htm)** will be used for practicing writing code
  - Included with ArcGIS Pro 2.5+
  - Optimal for ArcGIS and Python 3
  - Easily run code directly in ArcGIS Pro

2. Python Code Editor or IDE

The workshop is designed to use any script editor or integrated development environment (IDE) and can be completed using any combination of the software included in the list below. The items in bold are the recommended applications for the workshop in order.

- **Jupyter Notebook (https://jupyter.org/)**
  - Installed with ArcGIS Pro 2.2+
  - Optimal for Python 3
  - Easily run code in the application

- **Visual Studio Code (https://code.visualstudio.com/)**
  - Great all purpose code and text editor...an essential application for coding in numerous languages!
  - See Getting Started with Python in VS Code (https://code.visualstudio.com/docs/python/python-tutorial) for Python setup
  - Requires some application savvy-ness to unleash full potential
  - Not very easy to run code in the application

- **Notepad++ (https://notepad-plus-plus.org/)**
  - Great all purpose text editor
  - Highlights Python syntax so can be used to develop Python scripts
  - Not very easy to run code in the application

- IDLE (https://docs.python.org/2/library/idle.html)
  - Installed with ArcGIS Desktop
  - Can be used to run and edit scripts
  - A little clunky for Python development...but is a standard software worth knowing

- PyScripter (http://sourceforge.net/projects/pyscripter/files/)
  - Good for Python 2 script development for ArcMap
  - Can be used to run and edit scripts
  - Download **v. 3.6.1 32-bit** version for ArcMap without 64-bit Background Geoprocessing
  - Download **v. 3.6.1 64-bit** version for ArcMap with 64-bit Background Geoprocessing
  - *NOTE: The zip files contain portable versions of PyScripter. No installation is needed. Just unzip the archive and start using PyScripter.*

- Spyder (https://www.spyder-ide.org/)

- Better for scientific Python 3 and ArcGIS Pro development
- Requires special installaiton through ArcGIS Pro and/or Anaconda that may be tricky
- Easily run code in the application, but may require some setup

- Pycharm (https://www.jetbrains.com/pycharm/)
  - Good for all around Python 3 development
  - Requires a normal installation from a free download
  - Easily run code in the application, but may require some setup

- There are many other script editors and IDEs! Always good to experiment with others

Download Exercise Data

From GitHub

- Go to repo at **https://github.com/whitacrej/Python-for-ArcGIS-ILGISA-2020 (https://github.com/whitacrej/Python-for-ArcGIS-ILGISA-2020)**
- Click on the **Code** dropdown
- Click **Download Zip**
- **Extract** zip file to desktop or well-known folder

From Pathable

- Go to **https://ilgisa.pathable.co/meetings/virtual/GgicDzbmPNjyrJKxz (https://ilgisa.pathable.co/meetings/virtual/GgicDzbmPNjyrJKxz)**
- Click on **Files**
- Click on **Python for ArcGIS ILGISA 2020 GitHub Reppo**
- **Extract** zip file to desktop or well-known folder

Opening and Accessing Exercise Data and Scripts

- All of the GIS data and ArcGIS Notebooks can be opened in ArcGIS Pro 2.5+
- Python Scripts will need to be opened in a script editor or IDE

Note about ArcMap

- ArcMap should be usable for all exercises, however it has not been tested
- Jupyter Notebooks cannot be used; the ArcMap Python Window and script editors will need to be used to write and test code

ArcGIS vs. ArcMap vs. ArcGIS Pro

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

Script Editor vs. Integrated Development Environment (IDE)

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))
- Allows for running and debugging code

Script Editor

- Software that only allows textual code editing
- Does not integrate running and debugging code on the base software; may be included as an extension

IDE: Jupyter Notebook

Jupyter

- **Actually, it is Project Jupyter (https://jupyter.org/)**

Project Jupyter is a non-profit, open-source project, born out of the IPython Project (https://ipython.org/) in 2014 as it evolved to support interactive data science and scientific computing across all programming languages. Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license (https://opensource.org/licenses/BSD-3-Clause).

- **IPython (https://ipython.org/) is...**

IPython provides a rich architecture for interactive computing with:

- A powerful interactive shell.
- A kernel for Jupyter.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into your own projects.
- Easy to use, high performance tools for parallel computing.

- **So, it has it roots in Python**
- **But can be used with other programming languages**

Jupyter Notebook
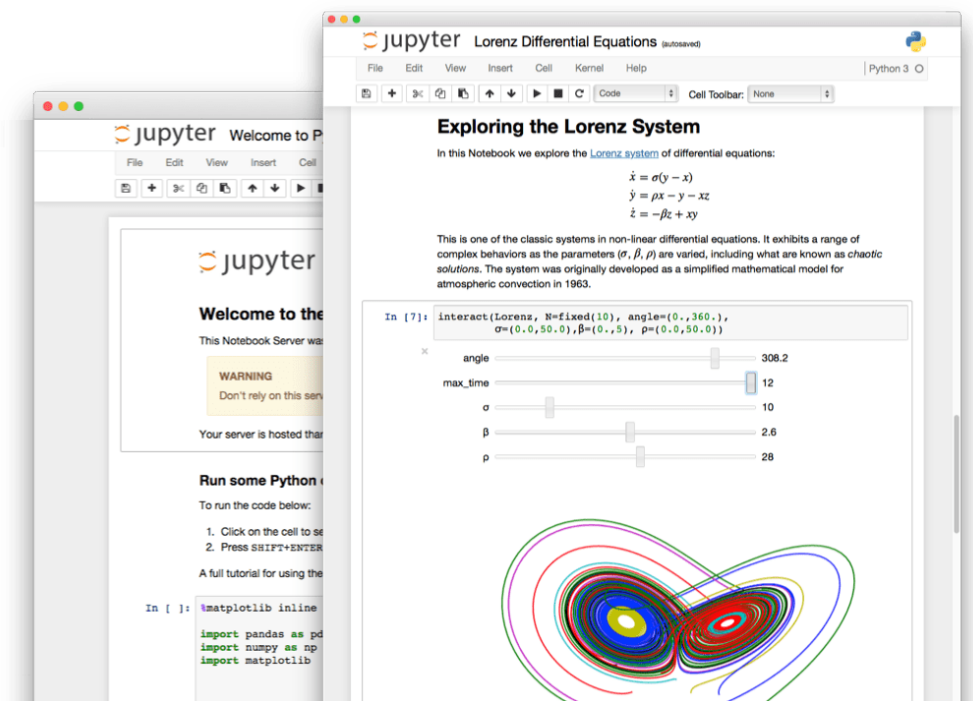
- **Web-based application (i.e. browser-based) to help capture entire computational workflows and processes**
  - Development
  - Documentation
  - Code execution
  - Visualizing and communicationg results
- **Main Features**
  - Acts as an interactive IDE, including syntax highlighting, indentation, code completion
  - Executes code directly in the browser environment and maintains results until cleared or when code is run again
  - Utilizes Markdown markup language to provide additional code commentary or for presentations

For further resources:
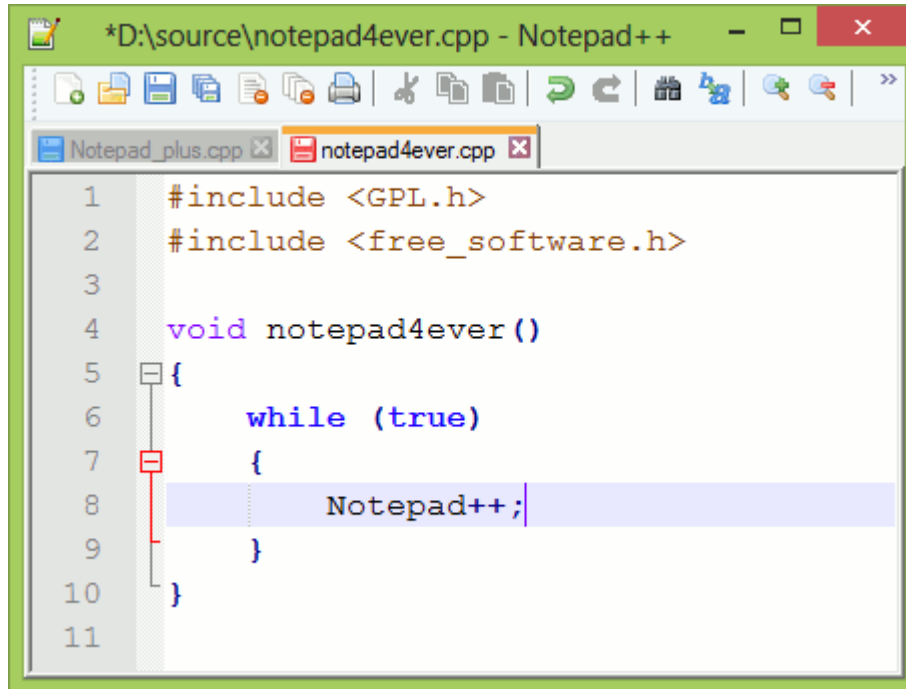
Project Jupyter: https://jupyter.org/index.html (https://jupyter.org/index.html)

Jupyter Notebook Documentation: https://jupyter-notebook.readthedocs.io/en/stable/index.html (https://jupyter-notebook.readthedocs.io/en/stable/index.html)

Code Editor: Notepad++

- Free (as in "free speech" and also as in "free beer") source code editor and Notepad
- Available for Windows
- Comes with built-in support for JavaScript, TypeScript and Node.js
- Rich ecosystem of extensions for other languages (such as C++, C#, Java, **Python**, PHP, Go) and runtimes (such as .NET and Unity).
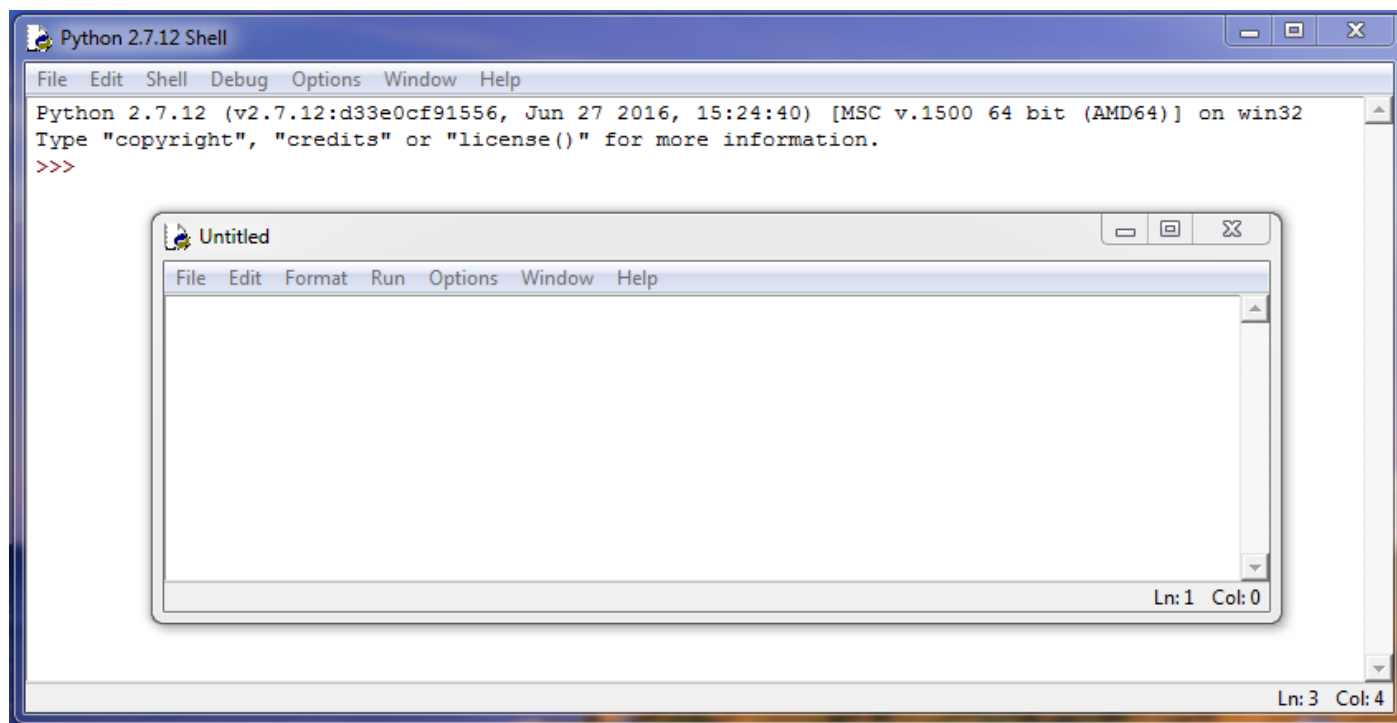
Code Editor: Visual Studio Code (or VS Code for short)

- Lightweight but powerful desktop source code editor
- Available for Windows, macOS and Linux
- Comes with built-in support for JavaScript, TypeScript and Node.js
- Rich ecosystem of extensions for other languages (such as C++, C#, Java, **Python**, PHP, Go) and runtimes (such as .NET and Unity).
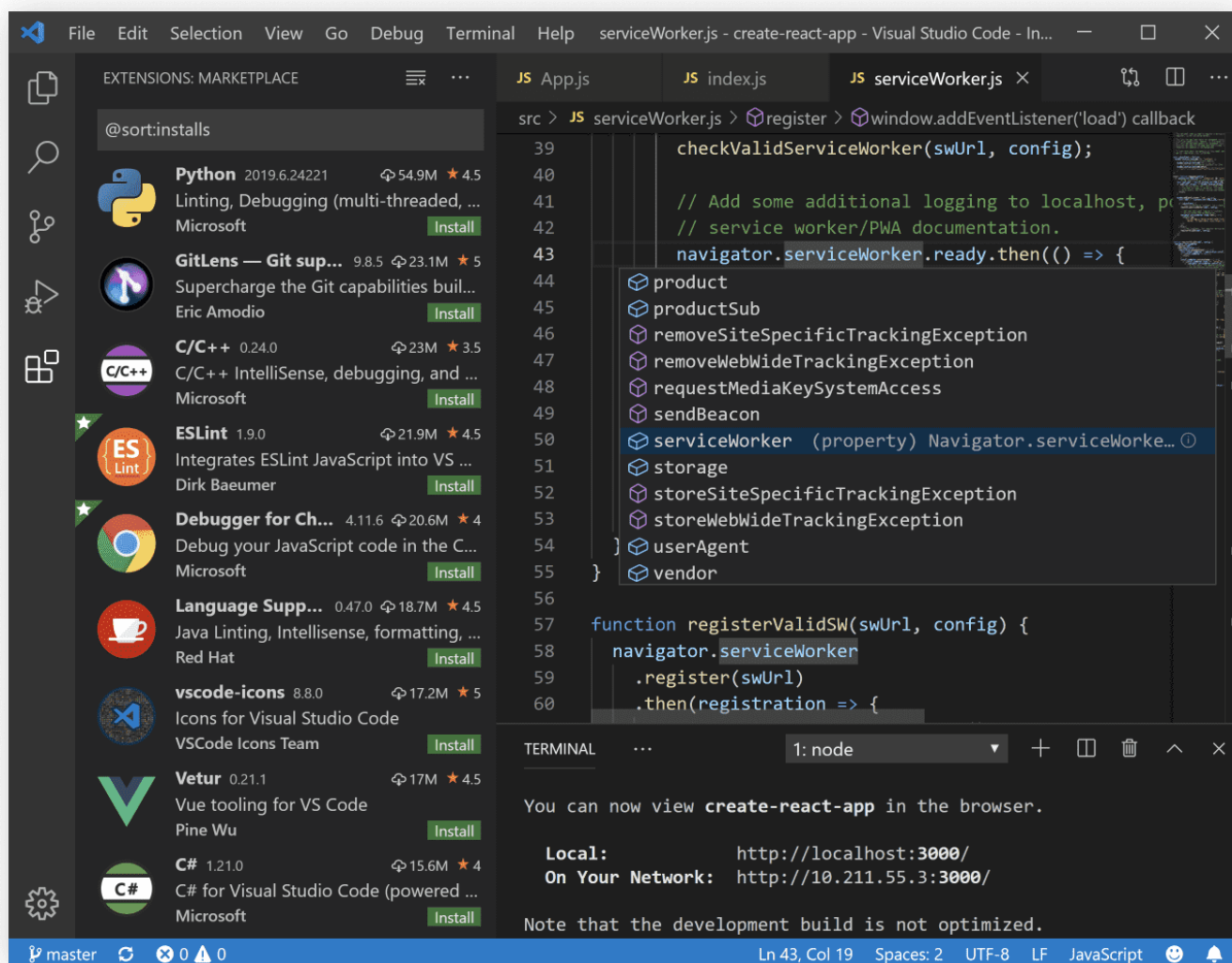
IDE: IDLE

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

```
Python 2.7.12 Shell

File  Edit  Shell  Debug  Options  Window  Help
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:24:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
Untitled

File  Edit  Format  Run  Options  Window  Help



                                                                    Ln: 1  Col: 0
                                                                    Ln: 3  Col: 4
```

Typing and running code in Notebooks or the IDE

- I will give an explanation of concepts, then have code examples that can be run in a notebook or IDE in the grey boxes.
- Use a blank notebook or your IDE to *TYPE* the code as we go
    - Typing the code will help you learn it better!
    - Try to *avoid copy and pasting* unless instructed otherwise
- When typing, you will likely notice the auto complete functionality. Mastering this will help type code faster and with less mistakes.

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

Congratulations, this is your first code!!!

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

ArcGIS Pro Python and ArcPy Documentation

- Python in ArcGIS Pro: [https://pro.arcgis.com/en/pro-app/arcpy/get-started/installing-python-for-arcgis-pro.htm* (https://pro.arcgis.com/en/pro-app/arcpy/get-started/installing-python-for-arcgis-pro.htm*)](https://pro.arcgis.com/en/pro-app/arcpy/get-started/installing-python-for-arcgis-pro.htm*)
- Learning and understanding the ARcGIS Documentation on Python is very important!!!

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.

III. 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()` *
  - Add the variable or string within the parentheses `()`

*# Note: print() will work in both Python 2.x and Python 3.x, but using just `print` without `()` is acceptable syntax in Python 2.x therefore the syntax above is preferred so that it is acceptable in both Python versions. We will use `print()` throughout this workshop

```
In [ ]:  # This WILL work in Python 2.x and Python 3.x!!!
         print('Python is so cool!')
```

```
In [ ]:  # This WON'T work in Python 3.x (i.e. ArcGIS Pro)
         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)
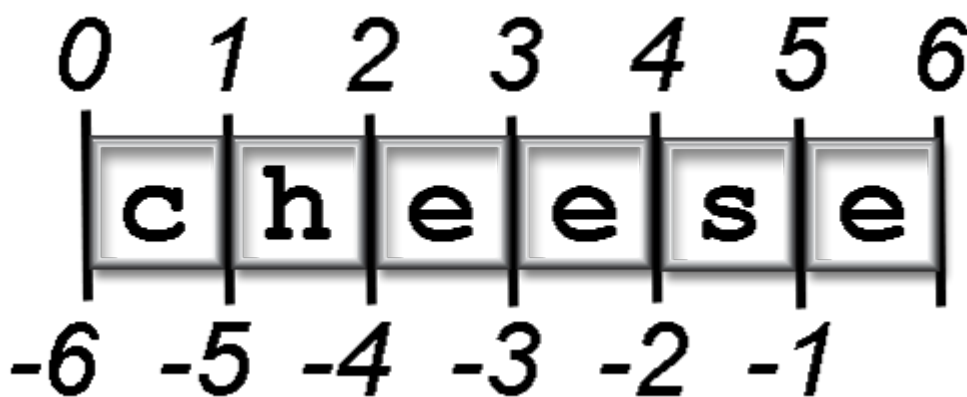

```
In [ ]:  print('cheese ' + 'whiz') # Note the space...
         print('cheese' * 3) # Note no space...
         print('cheese'[1:4])
         print('cheese'[:2])
         print('cheese'[:-2])
         print('cheese'[2:])
         print('cheese'[-2:])
```

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 = 3.3

         print(type(three))

         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. `{}`)
- Use the `bool()` function to evaluate a variable's boolean value

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

a = True
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

- 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

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

  - Iterate through it (use a loop)

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

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

  - Change list values

```python
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)

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

        print(fruitlist)

        print(fruitlist[1])

        print(fruitlist[1:3])

        print(len(fruitlist))
```

```python
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)
        ```

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 [ ]: 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

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

x = 99

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

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

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

In [ ]:
```python
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)

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, pass,
yield, break, except, import,
    print, class, exec, in, raise, continue, finally, is, return, def, for, lambd
a, 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
```

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

```
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 automatically)

- 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

```python
# 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...

```python
# These are all the same thing...
  'C:\\data\\things' # I prefer this one when working in ArcGIS Pro...I will te
ll you why later
  'C:/data/things'
  r'C:\data\things' # I prefer this one when working in ArcMap...I will tell yo
u 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

```python
# 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 in your c
ode
```
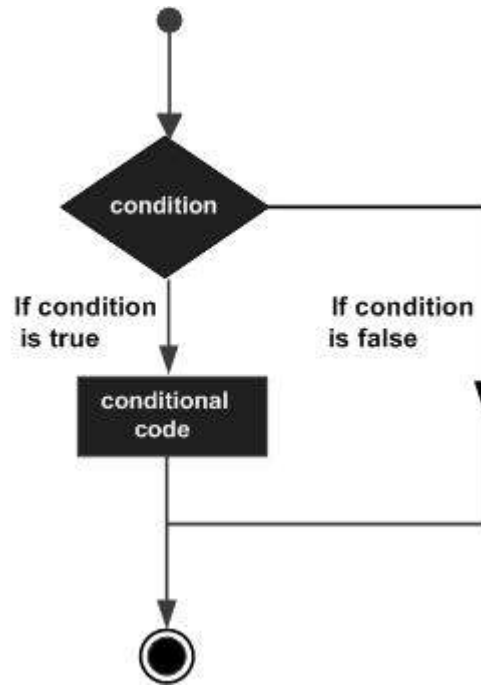
See the [PEP 8 -- Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)](https://www.python.org/dev/peps/pep-0008/) for more information

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 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

```python
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
==	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")
```

```
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 [ ]:  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 (e.g. `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.

```python
# Basic Funtion syntax

def functionname( parameters ):
    """function_docstring""" # This is optional, note triple-quotes
    # Function code
    return #expression
```
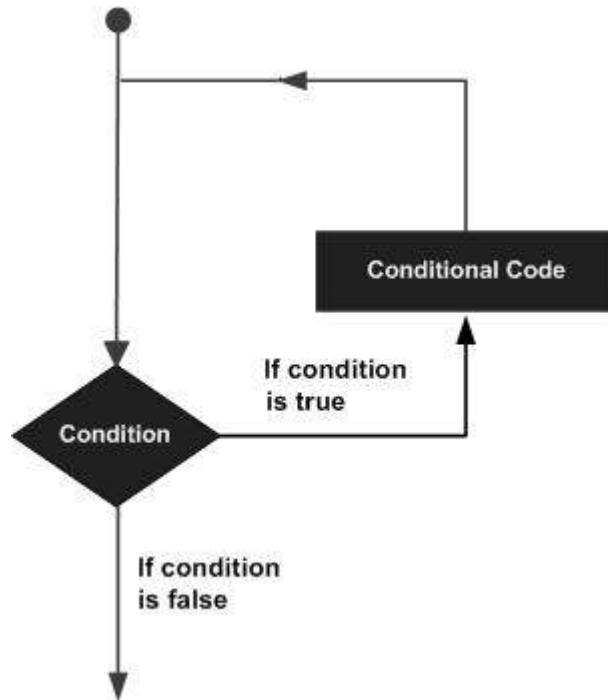
Source: https://www.tutorialspoint.com/python/python_functions.htm (https://www.tutorialspoint.com/python/python_functions.htm)

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

In [ ]:
```python
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
```

```
In [ ]:  print('Yay for Python!!')
```

Congratulations, you're on your way to using Python!!!