What is Python

- Python is a free, open-source, object-oriented programming language
  - Stong community support
  - 1000s packages you may download and reference (language processing, facial recongition etc.)

    Examples:
  - scipy, numpy, pandas, mpi4py for science
  - pyQT for GUI interface devlopment
  - APIs - LinkedIn, CraigsList, Google, BeautifulSoup
  - TensorFlow for Machine Learning
  - nltk for Natural Lanaguage Processing and Analysis
  - py2FaceR for Facial Recognition

- Objects can have properties and methods:
  - Properties – Name, Spatial Reference, Extent, etc.
  - Method – Something the object can do
- Scripting allows you to automate time-consuming and complex process so you can work more efficiently.

How do I get it

Python

Python of "free" but is curated by several organizations (some for profit)

https://anaconda.org/ (https://anaconda.org/)
https://www.enthought.com/product/canopy/ (https://www.enthought.com/product/canopy/)
https://www.python.org/ (https://www.python.org/)

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. Currently the repository contains over 130,000 packages.

https://pypi.python.org/pypi (https://pypi.python.org/pypi)

Make No Commitments

http://pythonfiddle.com/ (http://pythonfiddle.com/)

Some Python Miscellany

- Python is interpreted
- Python is case-sensitive
- Python uses indentation to define code blocks
- '#' is used to designate comment code (and '""" ...'""" for multi-line comments)
- '*' *is a wildcard character in strings (useful for finding or identifying specific files: "I:\users\wshakes\plays*henry*.doc")*
- = vs ==
  - = is used for assignment.
  - == is used to test for equivalency

Data Types

- Numbers
  - Integer/Long Integer, Float
- Strings
  - Text
- Lists
  - Ordered list of numbers, strings, other lists, or combinations of data types.
- Dictionaries
  - Keyed collection of numbers, strings, other lists, or combinations of data types.
- Tuples
  - Similar to list above but immutable (useful for indexing)

Numbers

- Assign number values to variables using =
- Convert between integer and floating point using int() and float() functions.
- Numbers can be converted into strings using the str() function
- Python cannot concatenate numbers and strings

In [32]:
```
number = 14
print(number)
```

```
14
```

In [33]:
```
float(number)
```

Out[33]:
```
14.0
```

In [34]:
```
int(number)
```

Out[34]:
```
14
```

In [35]:
```
str(number)
```

Out[35]:
```
'14'
```

In [36]:
```
year = 2018
s = "number = {0}".format(year)
print(s)
```

```
number = 2018
```

Op	Arithmetic Operators	Op	Comparison Operators
+	Addition	==	Is Equal To
-	Subtraction	!=	Does Not Equal
*	Multiplication	<>	Does Not Equal
/	Division	>	Greater Than
%	Modulus	<	Less Than
**	Exponent	>=	Greater Than or Equal To
//	Floor Division	<=	Less Than or Equal To

Strings

- Strings are defined by single or double quotes ("Hello World")
- Strings are a collection of characters
- Individial characters may be accessed via and index
- Backslashes are escape characters in Python. Use “r” to define strings that contain backslashes
- Strings are concatenated using the '+" operator

In [37]:
```
example = "Hi"
example = 'Hello world'
print(example)
```

```
Hello world
```

In [38]:
```
print(example[1])
```

```
e
```

In [39]:
```
filePath = "C:\\users\\chuck\\Tools"
print(filePath)
filePath = r"C:\users\chuck\Tools"
print(filePath)
```

```
C:\users\chuck\Tools
C:\users\chuck\Tools
```

In [40]:
```
example = "Hi " + 'everyone'
print(example)
```

```
Hi everyone
```

String Method	Description	Examples: **>>> txt = r"I:\GIS\clayton.shp"**
.startswith(prefix)	Returns True if string starts with prefix.	>>> txt.startswith("I:\GIS") True
.endswith(suffix)	Returns True if string ends with suffix. *Useful for finding filetypes!	>>> txt.endswith(".shp") True
.isalnum(string)	Returns True if all characters in string are alphanumeric.	>>> txt.isalnum() False
.replace(old, new)	Returns a copy of the string with all occurrences of old replaced by new.	>>> txt.replace("GIS", "GDB") 'I:\GDB\clayton.shp'
.split(sep)	Returns a list of the words in a string, using sep as the delimiter.	>>> txt.split("\") ['I:' , 'GIS' , 'clayton.shp']
.strip(chars)	Returns a copy of the string with the leading/ trailing chars removed. If no chars given, whitespace is removed.	>>> txt.strip(".shp") 'I:\GIS\clayton'

Lists

- Lists are ordered sets of data elements enclosed in square brackets
- Items in lists are ordered 0, 1, 2, 3, etc.
- Items in the list may be of different types
- To retrieve a specific item, give the list name followed by the item's index (i.e.,order) number enclosed in square brackets
- List comprehensions are a special construct for the creation of lists

In [41]:
```
emptyList = []
Cities = ["Houston" , "Austin" , "Dallas"]
print(Cities)
print(Cities[1])
```

```
['Houston', 'Austin', 'Dallas']
Austin
```

In [42]:
```
list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list)
list = [x for x in list if (x % 2) == 0 ]
print(list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

List Method	Description	Examples: >>> list = ['a', 'd', 'c', 'b']
.append(item)	Add item to the list.	>>> list.append('e') ['a', 'd', 'c', 'b', 'e']
.sort()	Sort the items of a list.	>>> print list.sort() ['a', 'd', 'c', 'b']
.reverse()	Reverse the item order.	>>> print list.reverse() ['b', 'c', 'd', 'a']
.remove(x)	Remove the first item from the list whose value is x.	>>> print list.remove("a") ['d', 'c', 'b']
.insert(#, item)	Insert item into the list at the list position #.	>>> print list.insert(0,"z") ['z', 'a', 'd', 'c', 'b']
.count(x)	Count the number of times x appears in the list.	>>> print list.count("c") 1

Dictionaries

- Dictionaries are unordered sets of key value pairs enclosed in curley brackets
- Items in the dictionary may be of different types
- To retrieve a specific item, give the dictionary name followed by the item's key enclosed in square brackets
- Dictionary comprehensions are a special construct for the creation of dictionaries

In [43]:
```
emptyDictionary = {}
emptyDictionary["cat"] = 1
emptyDictionary["dog"] = "happy"
print("emptyDictionary[\"cat\"] = {0}".format(emptyDictionary["cat"]))
print("emptyDictionary[\"dog\"] = {0}".format(emptyDictionary["dog"]))
```

```
emptyDictionary["cat"] = 1
emptyDictionary["dog"] = happy
```

In [44]:
```
CityPopulations = { "Houston":2000000 , "Austin":1500001 , "Dallas":30000
print(CityPopulations)
print(CityPopulations["Austin"])
```

```
{'Houston': 2000000, 'Austin': 1500001, 'Dallas': 3000000}
1500001
```

In [61]:
```
CityPopulations = {k:v for (k,v) in CityPopulations.items() if (v % 2) ==
print(CityPopulations)
```

```
{'Houston': 2000000, 'Dallas': 3000000}
```

In [46]:
```
CityPopulations.items()
```

Out[46]:
```
dict_items([('Houston', 2000000), ('Dallas', 3000000)])
```

Dictionary Method	**Description**	**Examples:** **>>> dict = {'a':0, 'd':1}**
.items()	Returns a view of the dictionary's (key,value) pairs.	>>> dict.items('e') dict_items([('a':0), ('d':1)])
.keys()	Returns a view object of all keys.	>>> print dict.keys() dict_keys(['a', 'd'])
.values()	Returns a view object of all values.	>>> print dict.values() dict_values([0, 1])
.pop(key)	Returns an item and deletes it from the dictionary.	>>> print dict.pop("a") {'d':1}
.clear()	Removes all items from the dictionary.	>>> print dict.clear() {}

Tuples

- Tuples are ordered sets of data elements enclosed in parenthesis
- Tuples are very similar to lists
- Tuples are immutable and therefore are very useful as indexers

In [47]:
```
person = ("Chuck", "Knight")
print(person[0])
```

```
Chuck
```

In [48]:
```
people = {}
people[("Chuck", "Knight")] = 10
people[("Kody", "Knight")] = 11
people[("William", "Tell")] = 0
print(people)
```

```
{('Chuck', 'Knight'): 10, ('Kody', 'Knight'): 11, ('William', 'Tell'):
0}
```

Conditions and Loops

- All conditional and loop statements end with a ':'
- Code within a conditional or loop is nested using spaces

if - elif - else Statements

- Perform some operation if a statement is true otherwise (else) do something else
- Boolean operators were listed previously (under Numbers)

In [49]:
```
pi = 3.14
if pi >= 4:
    print("pi is greater than or equal to 4")
elif pi > 3:
    print("pi is greater than or equal to 3 but less than 4")
else:
    print("pi is not greater than or equal to 3")
```

```
pi is greater than or equal to 3 but less than 4
```

In [50]:
```
pi = 3.14
if pi >= 4:
    print("pi is greater than or equal to 4")
else:
    if pi > 3:
        print("pi is greater than or equal to 3 but less than 4")
    else:
        print("pi is not greater than or equal to 3")
```

```
pi is greater than or equal to 3 but less than 4
```

for - else Statements

- Loops are used to iterate over a collection/range of elements
- Like other languages Python has the concept of break and continue for loops
- Unlike other languages a for loop may have an else clause
- Introducing range() - used to create iteration indices

In [51]:
```
list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for x in list:
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [52]:
```
for x in range(0, 10):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [53]:
```
list = []
for x in list:
    print(x)
else:
    print("There were no elements")
```

```
There were no elements
```

while - else Statements

- Loops are used to iterate some criteria are met (some statement is true)
- Again
  - Like other languages Python has the concept of break and continue for loops
  - Unlike other languages a while loop may have an else clause
- Introducing pass - The no op command

In [54]:
```
count = 0
while (count < 9):
   print("The count is: {0}".format(count))
   count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
```

In [55]:
```
count = 10
while (count < 9):
   print("The count is: {0}".format(count))
   count = count + 1
else:
    print("count was never less than 9")
```

```
count was never less than 9
```

In [56]:
```
count = 10
while (count < 9):
    pass
```

Modules

- At some point you will want to use other peoples Python code (OPPC)
- Importing allows your Python code to see installed Python packages
- You can use OPPC either from it's native namespace or merge it into yours

Some core modules

- os – Operating system functions
- sys – System-specific parameters and functions
- glob – Unix-style pathname patterns
- csv – CSV file reading and writing

In [57]:
```
import sys
print(sys.version)
```

```
3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
```

In [58]:
```
import os as o
print(o.getcwd())
```

```
/Users/chuck/Desktop
```

In [59]:
```
# Merge import into our namespace (no qualifier needed)
from time import sleep
print("Sleeping...")
sleep(1)
print("done!")
```

```
Sleeping...
done!
```

In [60]:
```
# Merge all imports into our namespace (no qualifier needed)
# Other than those prefixed by "_"
from random import *
print("randint = {0}".format(randint(0, 10)))
```

```
randint = 5
```