Introduction to Using Python in the Digital Humanities

Elisabeth Shook, Instructor
Kyle Shannon, Helper

This lesson is a conglomeration of the Python for the Humanities course by Dr. William Mattingly, the various Carpentries lessons on Python, and my own experience.

Before Class - First Things First

You will need to download the following based on your operating system:

- ☐ Anaconda - https://www.anaconda.com/products/individual
- ☐ Download owyhee-canyonlands-wilderness-B.txt and save on your Desktop

Schedule:

Jupyter Notebook

- Open Jupyter Notebooks
    - Terminal Window will open, then a browser window will open
    - You should see an overview of your drives on your computer. Navigate to your Desktop
        - *note that working off the Desktop IS NOT best practice
    - Ensure the Owyhee Text file is available on the Desktop
    - In right hand corner, click “New” and open a Python 3 notebook
    - Jupyter Notebooks has kernels or cells
        - Alt + Enter or Option + Enter runs cell and inserts new cell below
        - Ctrl + Enter or Command + Enter runs all cells
        - Shit + Enter runs only the current cell
        - D,D deletes cell (must click outside typing area)
        - A inserts cell above
        - B inserts cell below

Why Python

- Object Oriented - A computer programming language that organizes items into objects, allowing one to interact with objects in a more modular manner, meaning you can change an object without having to change the entire program

- Example

```
lunaAge = 12
lunaSpecies = "cat"
print("Luna is a " + lunaSpecies + ". Luna is " + str(lunaAge) + " years
old.")
```

- Easy to read
  - Indents
  - Case
  - Comments
  - Commenting out code (ctrl + /)
- Good for both humanities and hard sciences - both qualitative and quantitative data
- Python starts at 0
  - Very important to remember!

Data Types and Structures in Python

- Variables
  - These are your objects
  - Creates a space in memory
  - Variable names:
    - cannot start with a digit
    - cannot contain spaces, quotation marks, or other punctuation
    - may contain an underscore (typically used to separate words in long variable names)
- Data Types
  - Strings
    - Denoted with quotation marks
    - “Hello World”
      - Find = this finds a specific string in a string
      - Join = this allows you to join together multiple strings
      - Upper = this makes a string uppercase
      - Lower = this makes a string lowercase
      - Format = this allows you to format a string based on arguments that are stored outside of the string
      - Count = this allows you to count the frequency of a string within a string
      - Contains = this allows you to determine if a string contains a string
  - Integers
    - Whole numbers, no quotation marks
  - Floats
    - Decimals
      - 3.15
- Data Structures - Method of creating relational data:

- Tuples - Uses parentheses (I remember with the p in tuple meaning I need to use parens)
  - Immutable - Cannot be changed
  - Can use a mix of data types

```
adaCounty = ("Boise", "Eagle", "Garden City", "Hidden Springs", "Kuna",
"Meridian", "Star")
print(adaCounty)
print(adaCounty[3]
```

- Lists - Uses brackets
  - Mutable - Can be changed
  - Can use a mix of data types

```
reindeer = ["Dasher", "Dancer", "Prancer", "Vixen", "Comet", "Cupid",
"Donner", "Blitzen"]
print(reindeer)
print(len(reindeer))
reindeer.append("Rudolph")
print(reindeer)
```

- Dictionaries - Uses curly brackets
  - Key and values - Creates a relationship
    - Think of this as variable with value

```
scdm = {"members": ["Elisabeth", "Ellie", "Kimberly", "Maddie", "Shannon"]}
print(scdm)
```

Printing and Math

- Printing
  - print (“This is a string”)
  - print(42)
  - print(44.5)
    - Build
      - name = “Elisabeth”
      - profession = “Librarian”
        - print(name + “ is a “ + profession)
- Mathematical Operators:
  - Addition +

    - Subtraction –
    - Multiplication *
    - Exponential Multiplication **
    - Division /
    - Modulo %
        - This will return the remainder, e.g. 2%7 will yield 1.
    - Floor //
        - This will return the max number of times two numbers can be divided into each other, e.g. 2//7 will yield 3
- Put it together

```
librarianExp = 8
retirement = 30
timeLeft = 30-8
print("Elisabeth can retire in " + str(timeLeft) + " years."
```

Conditionals, Loops, and Functions

- Conditional Statement
    - Comparison Operators
        - Equal to (==)
        - Greater than (>)
        - Less than (<)
        - Less than or equal to (<=)
        - Greater than or equal to (>=)
        - Not equal to (!=)

```
luna = 13
if luna >= 12:
    print("Luna is a senior cat.")
```

- elif statement (else if)

```
luna = 8
if luna >= 12:
      print("Luna is a senior cat.")
else:
      print("Luna is a young cat.")
```

- Loops

```
for i in reindeer:
```

```
    print("On " + i)
```

- Functions

```
x = 0
while x < len(reindeer) - 1:
    print("On " + reindeer[x])
    x = x+1
```

```
print("But do you recall the most famous reindeer of all? " + reindeer[9] +
" the Red-Nosed Reindeer!")
```

- Why didn't this work? What is the error?

```
print("But do you recall the most famous reindeer of all? " + reindeer[8] +
" the Red-Nosed Reindeer!")
```

Working with Text Files

We are working with a document that has been OCR'd, meaning the computer has not translated characters very well. As you scroll through this document, you will see several mistakes or mis-formats from the OCR. This is often how you will find text-readable items from archives. It is extremely costly both in labor and time for archivists to both scan texts and create clean, machine readable text. Finding and correcting these errors using Regular Expressions, find and replace, and other tools can help you create a clean text file to work from.

Flags for Reading Text into Python

r = read only
r+ = read and write
w+ = write only
a = append only
a = append and read

Libraries

You will often encounter needs for programming that are not built-in to base python. One of the huge benefits of Python is that it is open source and has a community of developers constantly configuring new modules or libraries that you can download with more features. We will use three such libraries: Regular Expressions, NumPy, and MatPlotLib.

```
import re
import numpy as np
import matplotlib.pyplot as plt
```

To read the data:

```
filename = "Desktop/owyhee-canyonlands-wilderness-B.txt"

with open (filename, "r") as f:
    data = f.read()
    print (data)
```

We may have a few issues with the path with some folks. We'll need to pause to make sure we can get the data read.

Question - Why don't we say print(filename) instead?

```
with open (filename, "r") as f:
    data = f.read()
    print(data.find("in"))
    print(data.count("CHAPTER"))
    print (re.findall(r"VI-+", data))
```

What are all of these pieces doing? How might you use those?

```
filename = "Desktop/owyhee-canyonlands-wilderness-B.txt"

with open (filename,'r') as f:
    data = f.read()
    output=data.replace("dept", "Department")
    f = open(filename,'w')
    f.write(output)
    f.close()
    print(data)
```

What is this doing?

Now let's do some visualization!

```
with open (filename, "r") as f:
    data = f.read()
```

```
idCount = data.find("Idaho")
nvCount = data.find("Nevada")
orCount = data.find("Oregon")
stateData = {"Idaho":idCount, "Nevada":nvCount, "Oregon":orCount}
states = list(stateData.keys())
values = list(stateData.values())

fig = plt.figure(figsize = (10, 5))
# creating the bar plot
plt.bar(states, values, color ='maroon',
    width = 0.4)

plt.xlabel("States")
plt.ylabel("Number of Mentions")
plt.title("Count of State Mentions")
plt.show()
```