Python for Data Analysis and Visualization

Instructor: Claudia Carroll

Spring 2024

Session 1 (March 25)

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Arts & Sciences at Washington University in St. Louis
Signature Initiative

Goals for the Workshop

- Develop further fluency in the Python programming language
- Learn how to access and perform basic analysis of data contained in tabular and JSON files using the Pandas library
- Practice creating static and interactive visualizations of file-based data using Python

Workshop Plan

Class 1	Accessing tabular data from files
Class 2	Accessing JSON data from files; dealing with dates and times in tabular data
Class 3	Pandas for data analysis 1
Class 4	Pandas for data analysis 2
Class 5	Static data visualizations with Matplotlib and Seaborn
Class 6	Interactive data visualizations with Plotly and Bokeh

Lesson Structure

1. Review of previous class's material
2. Brief lecture on new concepts/material
3. Demo and Exercise (x2)

*Each class will have accompanying homework exercises, which you are strongly encourage to complete between class sessions.

Today's Lesson Plan

1. Lecture on Python basics
2. Demo: Accessing data in tabular files
3. Exercise: Practicing open and reading data files

Questions?

Transdisciplinary
Institute *in* Applied
Data Sciences (TRIADS)

Lecture: Crash Course on Python Basics

Python Syntax

Indentation:

- Improves readability
- Affects how code is interpreted and executed
- Especially crucial for control flow structures (loops and conditionals)

Case sensitive:

- Python treats **name** and **Name** as two different things
- Built-in keywords (like **print**, **True**, and **if**) are also case sensitive

Quotation marks:

- used to define and delimit text (strings)
- Single ‘ or double “ quotes are both acceptable—choose one and stick with it

Python Syntax cont.

Parentheses ( ):

- used to call and define functions and to define tuples
- Contain the arguments or parameters of a function
- Also used in math expressions to control order of operations

Commas , :

- used to separate elements of data structures like lists, tuples, sets, strings, dictionaries, etc., as well as function arguments

Square brackets [ ]:

- defining and accessing lists, as well as performing list operations

Variables

- Used to store data
- Different data types can be assigned to variables
- Variables are used within code

```
# Assigning values to variables
x = 10
name = "Alice"
numbers = [1, 2, 3, 4, 5]

# Using variables in operations
y = x + 5
greeting = "Hello, " + name

# Accessing and printing variables
print(x)          # Output: 10
print(greeting)   # Output: "Hello, Alice"
print(numbers)    # Output: [1, 2, 3, 4, 5]
```

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Data Types

Strings	"Heuston, we have a problem"
Integers	35
Floats	35.6
Booleans	True/False

Data Collection Types

List: ["apple", 12, "computer science", "apple", 13.2] • Order is saved • Can be rearranged after list is defined • Can contain duplicates • Elements can be added or removed • Indicated by square brackets	**Set**: {"orange", "house", 102} ☐ Order is not saved (unordered) ☐ Cannot contain duplicates ☐ Cannot add or remove elements once defined ☐ Indicated by curly brackets
Tuple : ("apple", 29, 32) ☐ Order is saved ☐ Order cannot be rearranged after tuple is defined ☐ Can contain duplicates ☐ Elements cannot be added or removed ☐ Indicated by parentheses	**Dictionary:** {"name": "Anne", "age": 19, "major": "communications"} ☐ Stores data in key/value pairs ☐ Order is saved (as of Python 3.7) ☐ Duplicate values permitted, but not duplicate keys within one item ☐ Elements can be added and removed ☐ Indicated by curly brackets and colons

Identify Data Types

```
1    x = 42
2    y = "Hello, World!"
3    z = [1, 2, 3]
4    w = {"name": "Alice", "age": 30}
5    b = ["ringo", "paul", "george", "john"]
6
```

Identify Elements of Syntax

```
6
7   fruits = ["apple", "banana", "cherry"]
8   for x in fruits:
9     print(x)
10    if x == "banana":
11      break
12
```

Errors

- Syntax Error
- Name Error
- Type Error
- Index Error

Error Example

- **Syntax Error:**

```
>>> print ("Hello, world!)
  File "<stdin>", line 1
    print ("Hello, world!)
             ^
SyntaxError: unterminated string literal (detected at line 1)
```

- Name Error
- Type Error
- Index Error

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Indices

- An index is the **position of a subset of data** within a data type (e.g. letters in a string, or elements in a list)
- In python, the index starts with **0**
- Elements can be access by index with the following syntax: **list[i]**
- **Example:**

```
>>> fruits = ["apples", "oranges", "melons"]
>>> print(fruits[1])
"oranges"
```

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Boolean Statements

- A **Boolean statement** is a statement that is either True or False
- Boolean statements are primarily used to filter data or methods based on certain conditions
- Boolean statements are usually produced using **Boolean operators**

```
>>> age = 15
>>> print(age < 12)
False
>>> print (age > 12)
True
```

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (divide and return the remainder)
**	Exponential
//	Floor division (divide and round down to the nearest whole number)

Comparison Operators

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Great than or equal to
<=	Less than or equal to

Conditionals

- Used to *filter* data for further operations

```
number = 0

if number > 0:
 print('Positive number’)
elif number <0:
 print('Negative number’)
else:
 print('Zero') print('This statement is always executed')
```

For Loops

- Used to *iterate* over a sequence of data (string, list, dictionary etc.)
- Can be *nested*

Syntax:

```
fruits = ["apple", "banana", "cherry"]

for x in fruits:
  for letter in x:
    print(letter)
```

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Functions

- Blocks of pre-written code
- Created by user or included in Python libraries
- Must be *defined* before being *called*
- Data can be passed to a function as arguments

```
foods = [“eggs”, “bread”, “milk”, “cookies”]

>>> def long_strings(x):
      for food in foods:
        if len(food) > 5:
          print(food)
      return

>>> print(long_string(foods))
cookies
```

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Transdisciplinary
Institute *in* Applied
Data Sciences (TRIADS)

Demo 1: Accessing Tabular Data

Setup

1. Create a folder called 'python-data' on your Desktop

2. Download the file title 'SAFI_results.csv' from the following repo: https://github.com/ClaudiaECarroll/python_data_class

3. Move the file into your new python-data folder

Exercise 1

1. From the SAFI_results.csv file extract all of the records
   where the C01_respondent_roof_type (index 18) has a value of 'grass' and
   the C02_respondent_wall_type (index 19) has a value of 'muddaub' and write them
   to a file.

2. Within the same program write all of the records
   where C01_respondent_roof_type (index 18) has a value of 'grass' and
   the C02_respondent_wall_type (index 19) has a value of 'burntbricks' and write them
   to a separate file.

In both files include the header record.

Exercise 1 Solution

```
with open ("SAFI_results.csv") as fr:

    with open ("SAFI_grass_roof_muddaub.csv", "w") as fw1:
        with open ("SAFI_grass_roof_burntbricks.csv", "w") as fw2:

            headerline = fr.readline()
            fw1.write(headerline)
            fw2.write(headerline)

            for line in fr:
                if line.split(",")[18] == 'grass' :
                    if line.split(",")[19] == 'muddaub' :
                        fw1.write(line)
                    if line.split(",")[19] == 'burntbricks' :
                        fw2.write(line)
```

Transdisciplinary Institute *in* Applied Data Sciences (TRIADS)

Transdisciplinary
Institute *in* Applied
Data Sciences (TRIADS)

Demo 2: Dictionaries

Exercise

1. Create a dictionary called dict_roof_types with initial keys of type1 and type2 and give them values of 1 and 3.

2. Add a third key type3 with a value of 6.

3. Add code to check if a key of type4 exists. If it does not add it to the dictionary with a value of 1 if it does, increment its value by 1

4. Add code to check if a key of type2 exists. If it does not add it to the dictionary with a value of 1 if it does, increment its value by 1

5. Print out all of the keys and values from the dictionary

Solution

```
dict_roof_types = {'type1' : 1 , 'type2' : 3}

dict_roof_types['type3'] = 6

key = 'type4'
if key in dict_roof_types :
    dict_roof_types[key] += 1
else :
    dict_roof_types[key] = 1

key = 'type2'
if key in dict_roof_types :
    dict_roof_types[key] += 1
else :
    dict_roof_types[key] = 1

for item in dict_roof_types:
    print(item, "=", dict_roof_types[item])
```

Solution

1. Complete in-class exercises
2. Complete Class One Homework Exercises

Materials and homework can be access via my GitHub repo for this class:

https://github.com/ClaudiaECarroll/python_data_class