First Steps with Python

Overview

- Overview of Python, Anaconda, and Jupyter Notebooks
- Data Values and interpretations
- Variables, Functions, and Arguments
- Comparisons, Conditional, and Loops
- Libraries

Resources

- O'Reilly Learning Platform: https://databases.lib.wvu.edu/connect/1540334373

Basic Concepts _ What is Python and Programming Language Software

Python is a programming language software packages that allows you to give commands to your computer.

- https://www.python.org/

Why use Python

- Rich Ecosystem of Libraries
- Integration with Other Tools and Languages
- Platform Independent and Non-proprietary
- Reproducibility and Transparency
- Integrates into Proprietary Software
- Shareable
- Add-on Libraries

Jupyter Notebooks

https://jupyter.org/

Jupyter Notebooks are an interactive web-based tool that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

Anaconda

https://www.anaconda.com/

Anaconda is a popular and powerful distribution of Python and R programming languages specifically tailored for data science, machine learning, and scientific computing.

Google Colab

https://colab.research.google.com/

a free cloud-based platform provided by Google that allows you to write, execute, and share Python code in a collaborative environment.

Getting Started

1. Open Anaconda Navigator
2. Open Jupyterlab
3. Using the directory on the right, select the folder you will be working in
4. Start a new notebook
5. Select the python kernel
6. Click on File > **Save Notebook As** and give the file a name
7. Click on File > **Save Current Workspace As** and save the workspace

Inputing and Running Code

In Jupyter Notebooks you input your code into cells.

- You can add new cells by clicking on the plus buttons in the cell or in the top menu.
- You can change how the kernel is interpreting the cell. By default it is code.
- You can run the cell by clicking on the play button or using the **keyboard shortcut: CTRL + Enter (PC) / CMD + Return (MAC)**.

In [ ]:

```
#lets try and input code and run a cell

2+2
```

Commenting

Since use will be performing several operations in a single document and even in a particular code chunk, it becomes important to document what processes you were performing or make notes to use for yourself or others about your intentions.

Entering a hastag (\#) into your code will comment anything that comes after for one single line.

In [ ]:

```
# what is the mean of the variable

import numpy as np # load numpy

example = [1,50, 100, 1000]

average = np.mean(example)

# Get the mean

print("The mean of this list is", average)
```

Data Values in Python

- String – string of characters with no numeric value – “hello world” “26501”
- Integer – whole number
- Float – number w/ decimal place
- Boolean – t/f – True or False
- None – nothing, nul, nil

In [ ]:

```
#Type functions

type("26501") #string
```

In [ ]:

```
type(4) #integer
```

In [ ]:

```
type(4.5673) #float
```

In [ ]:

```
type(True) #boolean
```

In [ ]:

```
#Strings - anything entered in "" will be interpreted as a string

hello = "Hello World"
type(hello)
```

Python Standard Library

https://docs.python.org/3/library/index.html

In [ ]:

```
#Help function

help(print)
```

Fundamental Concepts in Python

- Variables
- Types

- Functions
- Libraries
- Comparison
- Conditionals
- Looping
- Lists

Variables

Variables are containers for storing data values.

- Syntax: named_container = value_assigned
- x = 5

In [ ]:

```
#Examples

x = 5
y = "Hello World"
z= [1, 2, 3]
```

Explore Variables

In [ ]:

```
#list created variables

%whos
```

In [ ]:

```
# call the variables

print(z)
```

In [ ]:

```
# data values for variables

type(z)
print("Type of values of z:", type(z))
```

In [ ]:

```
# length of the variable

len(z)
```

Functions

Functions are how you give commands using python code.

It is highly suggested to use **TAB**, use the **documentation** for libraries, and use the **help function** to understand what functions and arguments are available to you.

Built-In Functions

Standard Library: https://docs.python.org/3/library/functions.html

Pandas: https://pandas.pydata.org/docs/reference/general_functions.html

Built-in functions are pre-defined functions that are available as part of the core language. They are build into the standard library as well as any loaded libraries.

Create a Function

In [ ]:

```
#You can create functions using def

def my_function():
  print("Hello from a function")

my_function()
```

Exercise: Create a Function

In [ ]:

```
# Step 1: Create Variables

name = input("What is your name: ")
age = input("What is your age: ")


#Step 2: Define the function

def demo(name, age):
    print(name, age)

#Step 3: Call the function

demo(name, age)
```

Indentation

In Python, indents are used to define the structure of the code. Unlike some other programming languages that use curly braces {} or keywords to indicate blocks of code, Python uses indentation. This makes the code more readable and easier to understand.

In [ ]:

```
def check_number(number):  # Starts the function; the indented lines below are part
    if number > 0:
        print("The number is positive.")
    elif number < 0:
        print("The number is negative.")
    else:
        print("The number is zero.")  # This is where the function's code ends
print("Hello World")  # This starts a new section of code
```

In [ ]:

```
def check_number(number):
print("The number is positive.")  # This line will cause an IndentationError
```

Arguments

Arguments are the values that are passed to a function when it is called. In Python, functions can accept zero or more arguments. Arguments are separted by **commas (,)**.

In [ ]:

```
#example of a using arguments with the round function

round(3.141592653589793)
round(3.141592653589793, 4)
```

Comparisons and Boolean Operators

Use comparison operators to determine if objects in python are identical to each other.

- Equal (==)
- Not equal (!=)
- Greater than (>)
- Less than (<)
- Greater than or equal (>=)
- Less than or equal (<=)

In [ ]:

```
#Comparison

fruit1= "apple"
fruit2= "orange"

#Are items/variables == to eachother
fruit1 == fruit2
```

In [ ]:

```
#Comparison

fruit1= "apple"
fruit2= "apple"

#Are items/variables == to eachother
fruit1 == fruit2
```

In [ ]:

```
#Comparison

fruit1= "apple"
fruit2= "Apple"
```

```
#Are items/variables == to eachother
fruit1 == fruit2
```

In [ ]:

```
#Comparison

number1 = 1
number2 = 2

#Are items/variables == to eachother
number1 != number2
```

In [ ]:

```
##boolean operators - > Can be combined variables/inputs with boolean operators AND

6 > 3 and 1 < 2
```

In [ ]:

```
#Can check existence of an object in a string

"n" in "mississippi"
```

Conditionals

Python operators that looks to see if an object meets stated conditions and will then run operations based of those determinations.

- If = If the conditions made in the statement are met then perform the operation.
- Else = If the conditions made in the statement are NOT met then perform another operations.
- Elif = If the condition made in the statement are NOT met by either the IF statement or the ELSE statement perform this operation.

In [ ]:

```
#Conditionals

hungry = True

if hungry:
    print("Go eat something")

print("Continue with your day")
```

In [ ]:

```
#Conditionals

hungry = False

if hungry:
    print("Go eat something")
else:
    print("Eat this anyway")
```

In [ ]:

```
#Multiple Criteria elif
#elif water_temp == 0: --> not bad
```

```python
water_temp = 0

if water_temp < 0:
    print("brrrrrr")
elif water_temp > 100:
    print("tccchhhhh")
else:
    print("I can drink this")
```

Loops

The requested operation will repeat until it is told to stop.

For Loops

The 'for' loop is typically used when you know the number of iterations in advance or when you want to iterate over a sequence or an iterable object.

*syntax: **for** variable **in** iterable*

- variable: This is a placeholder variable that will take on the value of each element in the iterable object during each iteration of the loop.
- iterable: This is the object over which the loop iterates. It can be a sequence (like a list, tuple, or string) or any other iterable object.

In [ ]:

```python
#for loop

fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

While Loops

The while loop is typically used when you don't know the number of iterations in advance or when you want to loop as long as a certain condition is true.

In [ ]:

```python
while True:
    print("It's still true")
```

In [ ]:

```python
x = 0
while x <= 20:
    print(x)
    x = x + 1

print("Stop looping")
```

In [ ]:

```python
x = 0
while x <= 20:
    if x == 8:
        break
    print(x)
    x = x + 1
```

Libraries

Libraries in Python are collections of pre-written code that provide ready-made functions and tools for common tasks. They save you time and effort by providing solutions to problems that programmers commonly encounter.

In [ ]:

```
import datetime

# Get the current date and time
current_date = datetime.datetime.now()

# Print the current date and time
print("Today's date and time:", current_date)
```

In [ ]:

```
#view installed libraries

# !pip list
!conda list
```

In [ ]:

```
#search for library

!conda search beautifulsoup
```

In [ ]:

```
#update pacakge

!conda update beautifulsoup4
```

Call A Library

To use the functions of a library you must call it during your current kernel.

Import

Using import for a library will bring all functions for the library into your workspace

In [ ]:

```
import math

# Now you can use any tool from the math toolbox
print(math.sqrt(16))  # Output: 4.0
print(math.pi)        # Output: 3.141592653589793
```

From

Using from will just bring one particular function from the library into your workspace.

In [ ]:

```
from math import sqrt

# Now you can use just the sqrt tool directly
print(sqrt(16))  # Output: 4.0
```

Installing a Library

You install Python Libraries using the terminal

- Go to File > New > Terminal
- Copy the following **conda install -c conda-forge geopandas** and past into the terminal.

Important Libraries for Data Science

- Pandas
- Matplotlib
- NumPy
- SciPy
- Plotly

Magic Commands

Magic commands in Python, specifically in environments like Jupyter Notebook, are special commands that help you perform various tasks more easily.

In [ ]:

```
%time
```