Introduction to Python

Data Management and Analysis

Adapted from Data Carpentry's material:
https://datacarpentry.org/python-ecology-lesson/01-short-introduction-to-Python.html

LIBRARIES
COLORADO STATE UNIVERSITY

Community Agreements

- Be present, open, honest, & authentic
- Speak from personal experience: use “I” statements to share thoughts & feelings
- Listen actively & respectfully
- Be open to new and different perspectives
- Respect and maintain confidentiality

Virtual Expectations

- Please use mute if not speaking
- If you need to turn off video, that is fine, please participate
- Speak up, Raise hand and use chat functions
- In small groups, create and maintain expectations

What is Programming and Coding?

Programming: Writing “programs” that a computer can execute to produce some result

Multi-step

1. Identifying the aspects of the real-world problem that can be solved computationally
2. Choose computational solution
3. Implementing the solution in a specific computer language (**Coding**)
4. Testing, validating, and adjusting implemented solution

What is Python?

- General purpose programming language
- Supports rapid development of applications including data analysis and analytics
- Name refers to both language and the tool that executes the scripts
- Has built-in standard libraries and lots of community generated ones

Advantages:

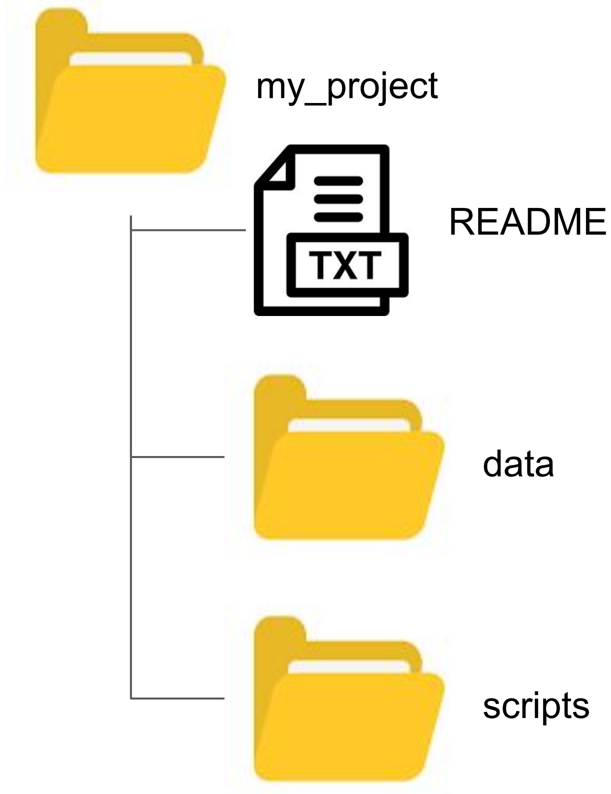
- Free
- Open-source
- Available on all major platforms (macOS, Linux, Windows)
- Supported by Python Software Foundation
- Allows multiple programming paradigms
- Has large community
- Rich ecosystem of third-party packages

Why Choose Python for Data Analysis

- Easiest to learn
- Reproducibility
  - Free and Open-Source Software (FOSS)
  - Cross-Platform
- Versatility
  - Used in many applications and powering processes at Google, NASA, Netflix
- Interdisciplinary and extensible
- Active and welcoming community
- Well documented

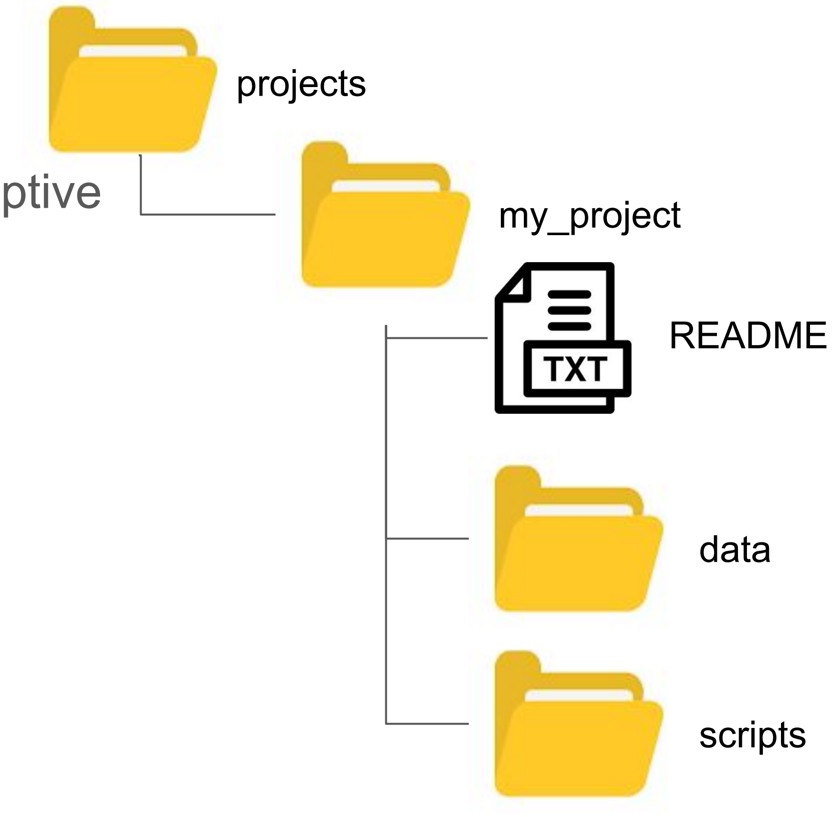
Research Project: Best Practices

- Create a project folder to work from
  - Add README to this folder
- Use folders to organize files in project folder
  - data/
    - use additional folders for raw and clean data
  - data_output/
    - to export processed results
  - documents/
    - outlines, drafts, other text
  - scripts/



Exercise: Create an Organized Project Folder

- Start by creating a project folder
  - Ideally located in 'home' folder
- Name the project something descriptive
- Create a README file
- Create data and scripts folders



Data File Naming

- Unique
- No special characters *? \ / : # % ~ { }”’
- Descriptive name
- Lowercase with underscores, or camelCase
    - No spaces
- Consistent, predictable, pattern
- Naturally ordered
    - For versioning use suffix **_v01**
- Time-series data
    - Use UTC time YYYY-MM-DD
        - E.g some_data_2021-09-13.csv
    - Best to break-up data files into chunks or use a database
- Consider stand alone names for shared data

Examples
some_composite_layer_buffer_30_meters.shp
Vs. comp_lyr_buff_30_m.shp
Vs. CompLyrBuff30M.shp

Consider acronyms
BLM_CO_MCA_Density_20181212.shp

Careful with capitalization and pluralization

File Naming Conventions: simple rules save time and effort ref: https://www.abdn.ac.uk/staffnet/documents/policy-zone-information-policies/File%20Naming%20Conventions%20July%202017.pdf

Exercise: Installing Python using Anaconda

1. Visit https://www.anaconda.com/products/individual in your web browser.
2. Download installer
3. Run installer
4. Open Anaconda

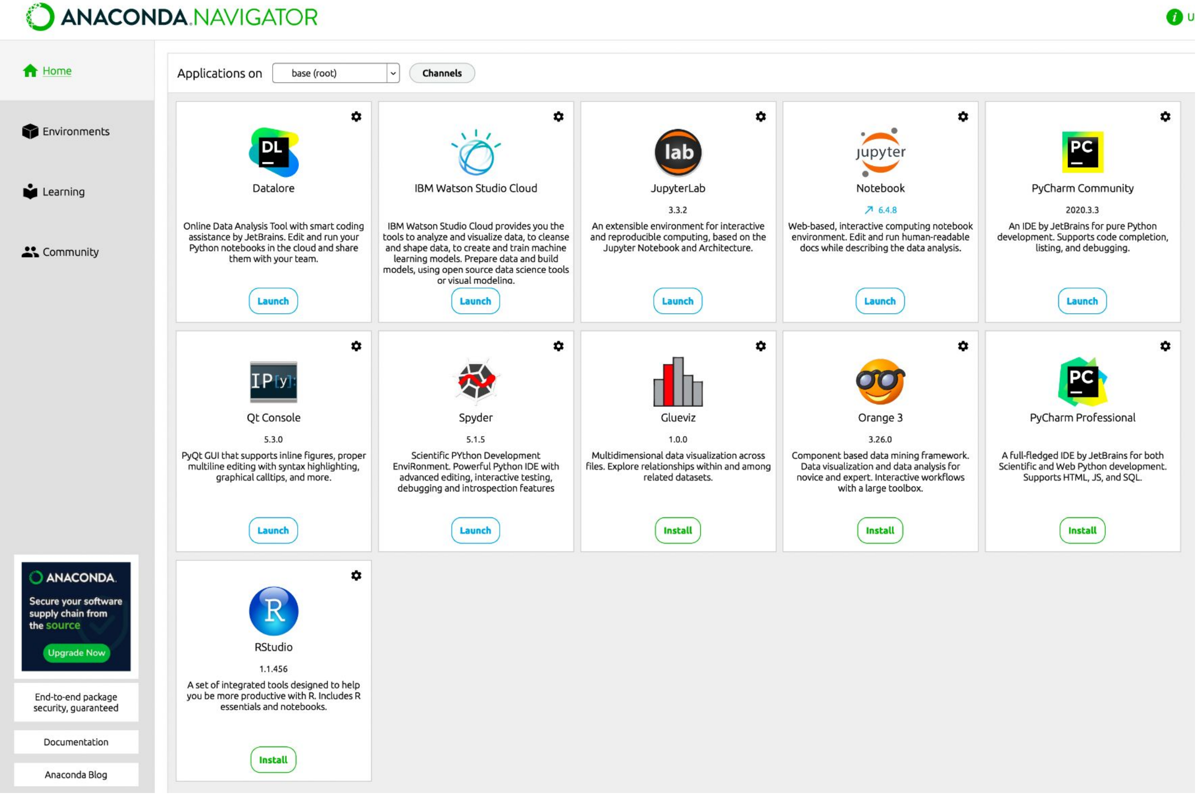
Knowing Your Way Around Anaconda

Lots of ways to work

- IPython console
- Jupyter Notebook
- Spyder IDE

Package Management:

- Comes with a package manager called **conda**
- Alternatively use **pip**
  - Works great with virtual environments



Applications on base (root)

Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

JupyterLab
3.3.2
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
↗ 6.4.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyCharm Community
2020.3.3
An IDE by JetBrains for pure Python development. Supports code completion, listing, and debugging.

Qt Console
5.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
5.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.26.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

PyCharm Professional
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

ANACONDA.
Secure your software supply chain from the source

End-to-end package security, guaranteed

Jupyter Notebook

Code cells and output structure

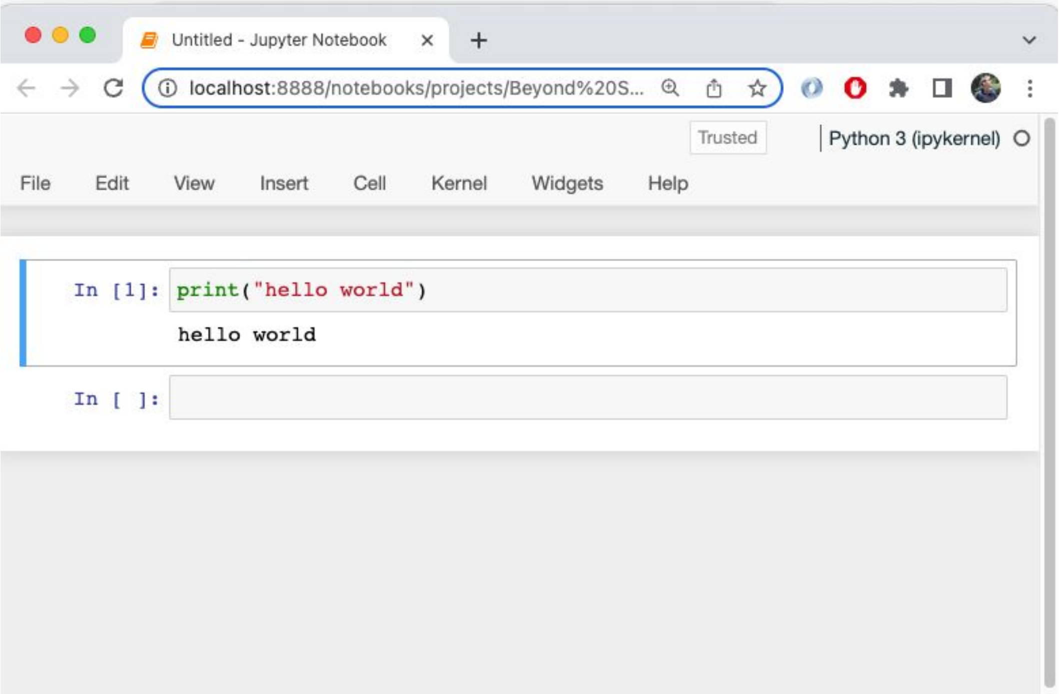
Use Shift+Return to execute cell

In[#] indicates execution count

Create new cells as needed

All commands and output saved to notebook, great for sharing analysis via GitHub

JupyterLab has all the same features and then some


Untitled - Jupyter Notebook
localhost:8888/notebooks/projects/Beyond%20S...
Trusted
Python 3 (ipykernel)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help

```
In [1]: print("hello world")
        hello world

In [ ]:
```

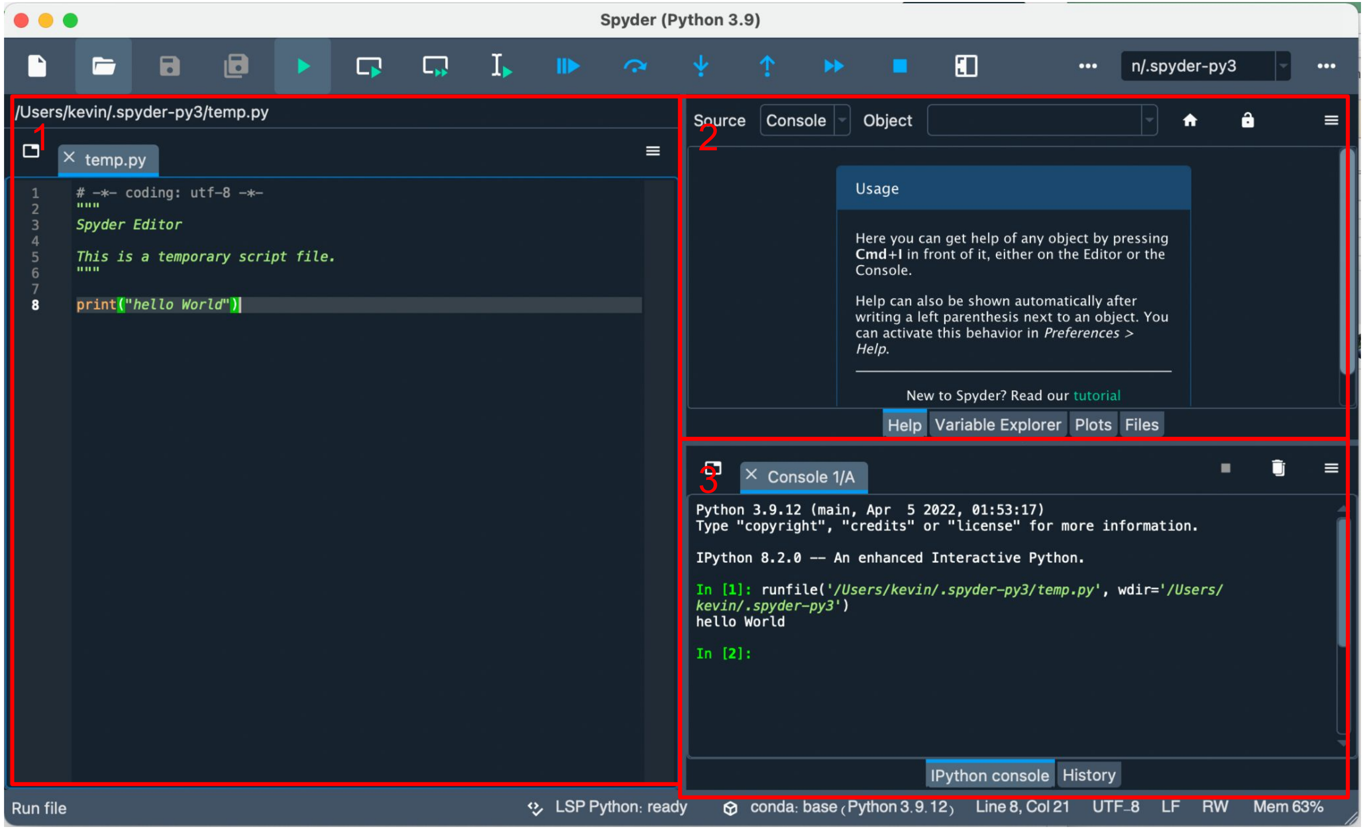
Spyder

Resembles Matlab

Helpful for writing scripts (i.e *.py* files) and testing code chunks

Other options include

- PyCharm (heavier)
- VS Code (very popular)



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

print("hello World")
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
Python 3.9.12 (main, Apr  5 2022, 01:53:17)
Type "copyright", "credits" or "license" for more information.

IPython 8.2.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/kevin/.spyder-py3/temp.py', wdir='/Users/
kevin/.spyder-py3')
hello World

In [2]:
```

Exercise: Using the Spyder Console

1. Within the Spyder console, type the below text in **bold** followed by the 'Enter' key:
   - **text = "foo"** # An example of a string
   - **number = 42** # An example of an integer
   - **pi_value = 3.1415** # An example of a float
2. 'print' each created variable to screen
   - Simply type the variable in the command line and press the enter key

Note a '#' character starts a comment, useful in documenting code

Every variable in python is an object with a *type*

3. Determine the type of each with the *type()* function
   - Enter **type(text)**
     - Do the same for each created variable to reveal its type

Exercise: Checking a Python Data Type

- Using the function type():
    - `type (10)` # An example of a int
    - `type(10.0)` # An example of a float
    - `type(‘10.0’)` # An example of str

Note a '# 'character starts a comment, useful in documenting code

- See how different data types interact with each other:
    - `type(10+10)`
    - `type(10+10.0)`
    - `type(10*10.0)`
    - `type(10+‘10.0’)`
    - `type(‘10.0’+‘10’)`
    - `print(‘10.0’+‘10’)`

Common Object Types

- Integers: 1,2,3, 100, 0, -1, -2, -3, -100
- Floats: -1.234, 1.234, 3.1415, 0.000000001,
- Strings: 'Hello World', "That's Correct"
- Boolean: True, False
- None and Null: No value at all
- Lists: [1,2,3]
- Tuples: (1,2,3) **immutable**
- Sets: {1,2,3} **each item must be unique**
- Dictionaries: {'Sun': 'Orange', 'Grapes': 'purple', 'red': rgb(255, 0, 0)}
- DataFrames: tabular data structure

Operators

- Symbols in python to perform a mathematical or logical operations

Type	Symbol
Arithmetic	**+ - * /**
Assignment	**= (right-side assigned to left)**
Relational	**> < == != >= <=**
Logical	**and or in**
Extraction	variable**[ ]**

Exercise: Experiment with Operators

- Within the Spyder console type **bold** text below and press the enter key:
  - **6 * 7** # Multiplication
  - **2 ** 16** # Power
  - **13 % 5** # Modulo
  - **3 > 4**
  - **True and True**
  - **True or False**
  - **True == False**
- What questions do you have on the output of each?

Sequences: Lists

Lists

- Data structure to hold ordered sequence of elements
- Each element can be accessed by an index.
  - Note that python indexes start with 0 instead of 1, e.g.:
    - `numbers = [1, 2, 3]`
    - `numbers[0]`
    - `output: 1`
- **for** loop able to access elements in list one at a time:
  - `for num in numbers:`
  - `    print(num)`

Sequences: Lists continued

Indentation critical in Python.

```
for num in numbers:

    print(num)
```

- Note that the second line in the previous example is indented
  - python’s way of marking a block of code
  - Used to nest lines associated with those above
- Add elements to list using the *append* method, denoted by '.' and method name followed by '()'
  - `numbers.append(4)`
  - `print(numbers)`

Indexed looping with Enumerate

When you want to know the index and value of what you're iterating on

```
values = ['foo','bar', 'baz']

for idx, val in enumerate(values):

    print(str(idx)+"="+val)
```

Sequences: Tuples

Tuple

- Like a list but uses '()' instead of '[]'
- Can not be changed once created (“immutable”)

  ```
  # Tuples use parentheses
  a_tuple = (1, 2, 3)
  another_tuple = ('blue', 'green', 'red')
  ```

Dictionaries

- Container for pairs of objects - keys and values

    -
      ```
      translation = {'one': 'first', 'two': 'second'}
      ```

    -
      ```
      print(translation['one'])
      ```

    -
      ```
      output: 'first'
      ```
- Two ways to access dictionary values with **for** loops:
    - *items* method

        -
          ```
          for key, value in translation.items():
          ```

        -
          ```
              print(key, '->', value)
          ```
    - *keys* method

        -
          ```
          for key in translation.keys():
          ```

        -
          ```
              print(key, '->', translation[key])
          ```

Exercise: Experiment with object types

- Create one of the following:
    - list
    - tuple
    - dictionary
- Print the first value from each
- Bonus:
    - Print the last value in each
    - Loop through each and print all values

Functions and Help

Function

- A block of code only run when called
- Declared with the **def** keyword, e.g.:

```
# function that takes two arguments and returns the first plus the
second
def add_function(a, b):
    result = a + b
    return result

 z = add_function(5, 10)

print(z)
```

Help

- To find out what methods are available for an object use the built-in **help** command
  - `help(object)`

Reserved Words in Python

Boolean values `True` and `False`,

Operators `and`, `or`, and `not`

`list`, `number`, etc

Given python's flexibility, reserved words can be overwritten e.g. list = ['a','b','c']

- Potentially causing problems

Full list of reserved words for Python version 3: https://docs.python.org/3/reference/lexical_analysis.html#identifiers.

5 Minute Post Workshop Evaluation

https://forms.office.com/r/E1Yy7RNv3y

If then Else

```
# Test a single condition

a = 5

if a == 5:
    print('A equals 5')


# If the outcome of the logical operator is not what we tested for we make use the construct of if-then-else

a = 6

if a == 5:
    print('A equals 5')

else:

    print('A is not equal to 5')

# use 'elif' after 'if' when you'd like to test for more conditions
```

Nested Statements and Loops

```
a = 'foo'
b = 'bar'

if a == 'foo':
    if b == 'bar':
        print('OK this is cool!')
    else:
        print('All foo no bar')
```

And - Or - Not

# We can make our conditions more powerful by using the keywords and, or, or not

```
a = 7

if a > 5 and a < 10:
    print('A is between 5 and 10')



a = 15
b = 5
c = 20

if a > b or a < c:
    print(f'A is between {b} and {c}')



a = 5
b = 10

if not a > b:
    print('A is not greater than B)
```

Shorthanded methods - making things more Pythonic

# If you only have one condition to test you 'could' put it all on one line.

```
a = 5

if a != 10: print('A is not 10')
```

```
a = 5
b = 10

print('A') if not a < b else print('B')
```

Nested, breaking and shorthanding loops

```
for i in range(4):
    print(f'\n')
    for j in range(3):
          print(f'i = {i}     j = {j}')



states = ['Colorado', 'Florida', 'Hawaii', 'California', 'Alaska']

for state in states:

    if state == 'Colorado':

        break

    print(state)



mylist = [1, 'One', 'Two', 2, 'three', 3.14]

[print(x) for x in mylist]
```