Very conside intro to Python

General properties of Python

- General purpose language
- Large set of well-developed libraries
- Supports a range of programming styles
  - Under the hood, Python is object-oriented
  - We will aim to keep it simple and cover related details as needed
- Simple, consistent, readable syntax

General properties of Python (cont'd)

- Delimited by whitespace
  - Good news: minimal use of brackets and punctuation
  - Bad news: pay attention to spaces/tabs at beginning of line
- Interpreted language (vs. compiled language)
  - Efficient for development:
    No need to declare data types or compile before testing
  - Usually less efficient for execution,
    but there are ways to get around that…

Comments in Python

- The goal is to write code that is clear, reliable, and maintainable
- Commenting your code is a great tool for achieving that goal!
  - Consider writing comments first then filling in the code
  - When you return to your code, you will thank your past self for providing comments that clearly communicate the code intent
- Line comment: Anything following the hash character (#)
- Docstrings: Give essential info about a function (in triple quotes)
  - Single quotes '''docstring info'''
  - Double quotes """docstring info""" (work equally well)

Common data types in Python

- Integer('int'): exact (no decimal), but with limited range; usually 64bit
- Float('float'): finite precision, usually 64bit including sign/abscissa/exponent
- Boolean('bool'): True/False
- String('str'): sequence of characters in quotes (single or double)

Container types

- Tuple('tuple'): "immutable sequence of immutable objects"
- List('list'): "mutable sequence of objects"
- Tuple and List indexed by integer in square brackets starting at 0 (yes, ZERO)
- Set('set'): unordered collection of unique, immutable objects
- Dictionary('dict'): a set of key:value pairs
- More on mutability etc. soon…

Creating functions/methods in Python

- Definition line:
  - Keyword ‘def’
  - Function name
  - Parentheses containing comma-separated list of arguments
  - A closing colon
- Code block indented relative to the definition line
  - Often includes a statement specifying a `return` value
- Function terminates when a non-indented line is reached
- Can include control statements (`for`, `if`) also delimited by indentation

Dynamic type inference

Do NOT need to specify/declare argument or data types of returned values in advance

Calling/executing a function

- Function name followed by parentheses enclosing comma-separated arguments
- Looks like definition line with `def` and `:` removed
- Example:

```
# define a function
def myfun(x,y):
    val = x*y
    return val

# call/execute the function
myfun(3,4)

# Output value???
```

Alternative function def: lambda function

- 1 line
- (`lambda` `comma-separated args` : `code`)(`comma-separated inputs`)
- Example:

```
(lambda x, y: x*y)(3,4)
```

- Output value ???

Libraries, packages, and namespaces

- Starting python loads the “base” code
- Additional software libraries (called “packages”) can be loaded
  - numpy, matplotlib, math, …
- Deal with name conflicts (for variables and functions)
  - Everything is an object; objects are identified by name
  - “namespace” = dictionary mapping names to object values
    - Python starts in the “global” namespace
    - Each imported package has a namespace
    - Each function has a namespace
    - Separate namespaces allow use of the same name in different contexts

Importing/Calling package functions

Import statement	Function call
import math	math.sqrt(x)
from math import sqrt	sqrt(x)
import math as whatever	whatever.sqrt(x)
import numpy as np	math.cos(x) #cos of scalar from base np.cos(x) # cos of array from numpy
import matplotlib.pyplot as plt	plt.plot(x)
from libname import *	# Use with caution! # Can overwrite your function def'ns

Numpy basics (matrix/vector capabilities)

- python's primary package for scientific computing: `import numpy as np`
- implemented in C to provide efficiency at execution time
- `ndarray`: contiguous memory array
    - Table of elements indexed just like a python list

numpy array Constructors

- `np.array(object)` converts a list or tuple to an ndarray
- `np.empty(shape)` creates an array with dimensions specified by `shape`
  - Does not change whatever junk values are currently in allocated storage
  - Only use `empty` when you are sure you will assign values before reading
- `np.zeros(shape)`, `np.ones(shape)` create arrays filled with 0, 1
- `id_n = np.eye(n)` makes name `id_n` refer to nxn identity matrix (for int n)
- `np.arange(start, end, increment)` #sequence over range with specified spacing
- #sequence over range with specified length (used frequently…)
  `np.linspace(start, end, size)`

Accessing elements in numpy arrays

Access i^{th} element:

- array name followed by element index in square brackets
  `a[i]`
- Example

```
# create array named a
a = np.array([3,5,7])

#get the first value
first_val = a[0]

#get the last value
last_val = a[2] # or last_val = a[-1]
```

`ndarray` constructors and attributes

import numpy as np

list0 = [1,2,3,4]

list1 = [5,6,7,8]

a0 = np.array(list0)

a1 = np.array(list1)

a0.size => 4 #using '=>' as shorthand for "returns"

a0.shape => (4,) #a 1-tuple

a0.dtype => int64 #data types will be discussed in more detail soon

a2 = np.array([list0, list1]) => [[1,2,3,4],[5,6,7,8]]

`ndarray` functions

list0 = [1,2,3,4]; a0 = np.array(list0)

list1 = [5,6,7,8]; a1 = np.array(list1)

a2 = np.array([list0, list1]) => [[1,2,3,4],[5,6,7,8]]

2*a0 => [2,4,6,8] #This is where “list” does not suffice!

a0 + a1 => [6,8,10,12]

np.dot(a0,a1) OR a0.dot(a1) => 70

np.dot(a2,a0) => [30,70]

a2_t = a2.T => [[1,5],[2,6],[3,7],[4,8]]

np.dot(a2_t,a0) => ??? #the arrays are not conformable. Expect error…

Python demo

Ways to run python:

- Interactively using terminal (or Anaconda prompt)
  - Probably first only the very first time…
- In Integrated Development Environment (IDE) VSCode, PyCharm, Eclipse,…
- Execute code stored in file(s) with .py extension
  - Navigate to directory
    - View files/subdirectories with ‘ls’ (or ‘dir’)
    - Change directory with ‘cd subdirectoryName’ (or ‘cd ..’ to back up a level)
  - Execute ‘python filename.py ’
- In Jupyter notebook
- See Ch. 1 for specifics to work through