Introduction to Python

1 Introduction to Python

Chris Myers

Cornell University Center for Advanced Computing (CAC)
and
Cornell Department of Physics / Laboratory of Atomic & Solid State Physics (LASSP)

https://cac.cornell.edu/myers

c.myers@cornell.edu

Python is at least 3 things:

- **A programming language**
  - Syntax, keywords, data types, objects, operators, variables, etc.
- **A software ecosystem**
  - Python Standard Library + many thousands of third-party packages for different tasks
- **A program that runs code written in the Python language**
  - An interpreter

2 Outline

- python as a program
- Python as a programming language
- *Python* as a software ecosystem

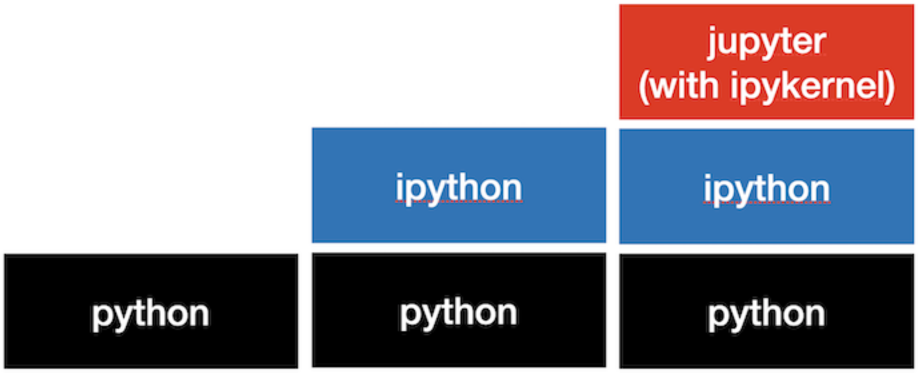
3 python as a program

3.1 Compiled vs. Interpreted Languages

- *Compiled* languages (e.g., C/C++, Fortran, Java, . . .): processed by a compiler to produce an *executable* or standalone application that can be run
- *Interpreted* languages (e.g., Python, R, MATLAB, Perl, . . .): processed by another program — an interpreter — that runs and executes program statements

3.2 Python interpreters

- python: the default/reference Python interpreter — formally known as CPython, and sometimes installed as python3
- ipython: an interpreter sitting on top of python (and written in Python), providing additional functionality for interactive work
- jupyter: a notebook-based software system that can process Python code by leveraging the ipython kernel (as well as kernels for other languages, such as R and Julia)
- various integrated development environments (IDEs) bundling code editors, ipython consoles, etc.
- other non-CPython-based interpreters that are not widely used: IronPython, PyPy, etc.



3.3 Python interpreters in action

- `python my_program.py`: especially useful for running in background or in batch submission systems
- `ipython`: an enhanced console with additional “magic” functionality to support interactive access
- `jupyter lab` and `jupyter notebook`: web-based environments merging code, documentation, graphics, and results

```
[1]: 2 + 2
```

```
[1]: 4
```

```
[2]: 'abc' + 'def'
```

```
[2]: 'abcdef'
```

```
[3]: # A nice trick I learned from Chris Cameron's last seminar on JupyterLab

     from IPython.core.interactiveshell import InteractiveShell
     InteractiveShell.ast_node_interactivity = "all"

     # I will also turn off pretty-printing

     %pprint
```

```
Pretty printing has been turned OFF
```

3.3.1 Installing Python and associated packages

- Your machine might already have a version of Python installed — best to leave that one alone (probably being used for sysadmin)
- Anaconda Python Distribution: installs a large collection of packages by default
- Alternatively, install a minimal distribution and create customized environments for different projects
- Miniconda
  - conda create -n my_env python numpy pandas jupyter ; conda activate my_env
- Python Virtual Environment
  - python -m venv my_env ; source my_env/bin/activate ; pip install numpy pandas jupyter

4 Python as a programming language

- General-purpose
- Object-oriented
- Dynamically typed
- Interpreted
- Extensible

4.1 A little bit of history

- Python was created by Guido van Rossum in the very late 1980s and early 1990s
- The language is named after the comedy group Monty Python, not the big snake
- The scientific computing community was an early adopter of Python
  - *abstractions and objects* for complex scientific/numerical concepts
  - *interfacing* to existing code written other languages
  - *scripting* and *steering* complex computations and workflows
  - *gluing* together different sorts of analyses
- In addition to being a very popular programming language, Python has inspired some important software memes and themes

  - “If Guido was hit by a bus?” — led to the creation of processes and standards for Python’s evolution
  - “Benevolent Dictator for Life” (BDFL) — Guido’s role as final arbiter of language decisions

4.2 Python as a general-purpose language

- Not constructed to support a specific problem domain
  - R: built to support statistical analysis
  - MATLAB (“Matrix Laboratory”): built to support linear algebra and matrix operations
  - Mathematica: built to support symbolic mathematics
- Much useful functionality for specific application areas is available through third-party packages
- The Python language is the substrate for tying all these pieces together
- Python is well-designed, intuitive, readable, practical, expressive, elegant, free, and open-source

4.3 Python as an object-oriented language

- Object-oriented means:
  - support for bundling together data and functions into complex data “objects”
  - support for defining new data types (classes) representing different abstractions useful for different problem domains
    - arrays, dataframes, networks, models, estimators, figures, etc.
- Python is practical and not strict — also supports procedural and functional programming
- Everything in Python is an object
  - a *type*
  - a *value*
  - some *attributes* (data defined in association with objects)
  - some *methods* (functions defined in association with objects)
  - a *namespace* that organizes attributes and methods

4.3.1 Everything in Python is an object

- 2+2 -> `(2).__add__(2)` # where the + operator results in a call to the method `int.__add__`
- `'abc' + 'def'` produces the string ‘abcdef’, where the `+` operator calls the method `str.__add__`
- the dot operator accesses elements in an object’s namespace

[4]:
```
# the built-in function dir() returns a list of names in a namespace

dir(2)
```

[4]:
```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio',
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator',
'real', 'to_bytes']
```

4.4 Python as a dynamically typed language

- variables acquire the type of whatever is assigned to them
  - `x = 3`   # x is an integer (int)
  - `x = 3.14`   # x is a floating-point number (float)
  - `x = "Hello, world"`   # x is a string
- as compared to *statically typed languages*, where the types of variables are declared, and errors are reported if data of a different type are assigned to a variable
- dynamic typing is often used in interpreted languages
- static typing is often used in compiled languages

4.5 Python as an interpreted language

- processed by an interpreter
  - the python interpreter (CPython) does on-the-fly compilation to intermediate bytecodes
- each statement executed sequentially
- very useful for interactive analysis, development, and prototyping
- programs are typically slower than for compiled languages
  - trading off development time vs. execution time

4.6 Python as an extensible language

- The Python language defines a C/Python Application Programming Interface (API)
- C/Python API enables the CPython interpreter to process compiled code written in C and other languages and to make the associated data and functions accessible in a Python program
- Many programs *written in the Python language* are actually calling compiled functions written in other languages, resulting in much higher computational performance than for pure Python code alone
- Many tools exist for generating interfaces to compiled code, compiling bits of Python code to "extension modules", etc.
- See our Cornell Virtual Workshop (CVW) topic on Python for High Performance at https://cvw.cac.cornell.edu/python

4.7 Built-in data types in Python

- numeric types: int, float, complex, bool
- string data types
- containers: lists, dictionaries, sets, tuples
- functions
- classes

- modules
- etc.

4.8 Built-in container types in Python

- **lists**: ordered, mutable sequences of objects, indexed by their integer position (starting at 0)
- **dictionaries**: mappings from a set of keys to associated values (akin to maps, hashes, associative arrays, etc.)
- **sets**: unordered collections of unique elements with support for set algebra (unions, intersections, differences, etc.)
- **tuples**: ordered, immutable sequences of objects, useful for bundling together related items
- **strings**: ordered, immutable sequences of characters, supporting many string-processing operations

Along with many other non-built-in container types defined in external packages, such as:

- arrays (of any dimensionality) — defined in numpy
- series and dataframes — defined in pandas

4.9 Python as a calculator

- addition + ; subtraction - ; multiplication * ; division /
- power ** ; modulo % ; floor division //

[5]:

```
(19 + (2*3 - 4*7) / (8 % 3))**3
```

[5]:

```
512.0
```

[6]:

```
x = 3
y = 14

z = (x * y) - (x + y)

z
```

[6]:

```
25
```

4.10 Code blocks and indentation

The readability of Python code is a key goal of its design. Using indentation to identify code blocks is central to that goal. Using a code editor that understands Python syntax and indentation helps a lot.

Python:

C/C++:

4.11 Code blocks and indentation (continued)

Python:

MATLAB:

4.12 Control flow

- Looping: for, while, continue, break
- Branching: if-elif-else
- Exception handling: try-except

4.13 Iteration and iterables

[7]:
```
for c in ['A', 'B', 'C', 'D', 'E']:
    print(c)
```

```
A
B
C
D
E
```

[8]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

[9]:
```
print(range(10))
```

```
range(0, 10)
```

[10]:
```
range?
```

[11]:
```
for i in range(4, 17, 3):
    print(i)
```

```
4
7
10
13
16
```

```
[12]: list(range(10))
```

```
[12]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[13]: sum(range(10))
```

```
[13]: 45
```

4.14 Iterating over other iterables

```
[14]: a_dictionary = {'A': 1, 'B': 2, 'C': 3}

      for key,value in a_dictionary.items():
          print(key, value)
```

```
A 1
B 2
C 3
```

4.15 Comprehensions

```
[15]: # List comprehensions

      squares = [n*n for n in range(10)]

      squares
```

```
[15]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
[16]: # Dictionary comprehensions

      import string
      mapping = {c:i for i,c in enumerate(string.ascii_letters)}

      mapping
```

```
[16]: {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9,
      'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18,
      't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27,
      'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33, 'I': 34, 'J': 35, 'K': 36,
      'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44, 'T': 45,
      'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51}
```

4.16 Exceptions and error handling

[17]:

```python
for denominator in [5,4,3,2,1,0]:
    print(1 / denominator)
```

```
0.2
0.25
0.3333333333333333
0.5
1.0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [17], in <cell line: 1>()
      1 for denominator in [5,4,3,2,1,0]:
----> 2     print(1 / denominator)

ZeroDivisionError: division by zero
```

[18]:

```python
for denominator in [5,4,3,2,1,0]:
    try:
        print(1 / denominator)
    except ZeroDivisionError:
        print("Cannot divide by 0")
```

```
0.2
0.25
0.3333333333333333
0.5
1.0
Cannot divide by 0
```

4.17 Exceptions and error handling (continued)

[ ]:

```python
#filename = 'a_file_that_does_not_exist.txt'
#inputfile = open(filename, 'r')
#lines = inputfile.readlines()
#inputfile.close()
```

[19]:

```python
filename = 'a_file_that_does_not_exist.txt'
try:
    inputfile = open(filename, 'r')
    lines = inputfile.readlines()
    inputfile.close()
except FileNotFoundError:
    print(f'{filename} does not exist')
```

```
a_file_that_does_not_exist.txt does not exist
```

4.18 Defining functions

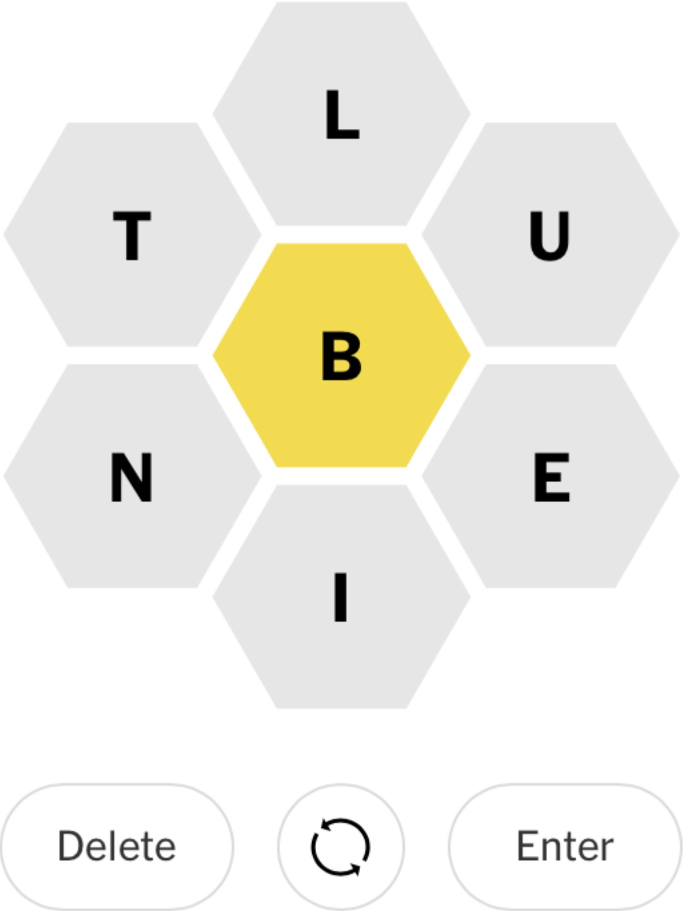
[20]:
```
# def is keyword to define a new function; return is keyword to return a value␣
 ↪from a function

def concatenate(string1, string2, separator=' '):
    return string1 + separator + string2

concatenate('abc', 'def')   # uses default argument for separator

concatenate('abc', 'def', '..')   # overrides default argument

concatenate(separator='--', string2='DEF', string1='ABC')   # uses keyword␣
 ↪arguments
```

[20]:
```
'abc def'
```

[20]:
```
'abc..def'
```

[20]:
```
'ABC--DEF'
```

4.19 Putting the pieces together: Spelling Bee



```python
# spellingbee.py

import itertools

available = 'BEILNTU'
center = available[0]

start = 'BL'
length = 5
exclude = {'II', 'UU', 'BBB', 'TTT', 'NNN', 'LLL', 'EEE'}

def words(available, start, length):
    num_unknown = length - len(start)
    iterator = itertools.product(available, repeat=num_unknown)
    return [start + ''.join(letters) for letters in iterator]

allwords = words(available, start, length)

for w in allwords:
    not_excluded = True
    for ex in exclude:
        if ex in w:
            not_excluded = False
    if not_excluded and center in w:
        print(w)
```

Import the itertools module, rather than figuring out the combinatorics for myself

Assign some variables to store data

Use built-in Python set object to hold exclusions

Define a function (note the indented code block)

Call a function from the itertools module

Use a list comprehension to create a list of all possible words

Call our function to generate all possible words

Loop over all words (note the nested indented code blocks)

Check if the word contains an excluded substring

Print the word if it is not excluded and it contains the center letter

[21]:

```python
# spellingbee.py

import itertools

available  = 'BEILNTU'
center = available[0]

start  = 'BL'
length = 5
exclude = {'II', 'UU', 'BBB', 'TTT', 'NNN', 'LLL', 'EEE'}

def words(available, start, length):
    num_unknown = length - len(start)
    iterator = itertools.product(available, repeat=num_unknown)
    return [start + ''.join(letters) for letters in iterator]


allwords = words(available, start, length)

for w in allwords:
    not_excluded = True
    for ex in exclude:
        if ex in w:
            not_excluded = False
    if not_excluded and center in w:
        print(w)
```

```
BLBBE
BLBBI
BLBBL
BLBBN
BLBBT
BLBBU
BLBEB
BLBEE
BLBEI
BLBEL
BLBEN
BLBET
BLBEU
BLBIB
BLBIE
BLBIL
BLBIN
BLBIT
BLBIU
BLBLB
```

```
BLBLE
BLBLI
BLBLL
BLBLN
BLBLT
BLBLU
BLBNB
BLBNE
BLBNI
BLBNL
BLBNN
BLBNT
BLBNU
BLBTB
BLBTE
BLBTI
BLBTL
BLBTN
BLBTT
BLBTU
BLBUB
BLBUE
BLBUI
BLBUL
BLBUN
BLBUT
BLEBB
BLEBE
BLEBI
BLEBL
BLEBN
BLEBT
BLEBU
BLEEB
BLEEI
BLEEL
BLEEN
BLEET
BLEEU
BLEIB
BLEIE
BLEIL
BLEIN
BLEIT
BLEIU
BLELB
BLELE
BLELI
```

BLELL
BLELN
BLELT
BLELU
BLENB
BLENE
BLENI
BLENL
BLENN
BLENT
BLENU
BLETB
BLETE
BLETI
BLETL
BLETN
BLETT
BLETU
BLEUB
BLEUE
BLEUI
BLEUL
BLEUN
BLEUT
BLIBB
BLIBE
BLIBI
BLIBL
BLIBN
BLIBT
BLIBU
BLIEB
BLIEE
BLIEI
BLIEL
BLIEN
BLIET
BLIEU
BLILB
BLILE
BLILI
BLILL
BLILN
BLILT
BLILU
BLINB
BLINE
BLINI

BLINL
BLINN
BLINT
BLINU
BLITB
BLITE
BLITI
BLITL
BLITN
BLITT
BLITU
BLIUB
BLIUE
BLIUI
BLIUL
BLIUN
BLIUT
BLLBB
BLLBE
BLLBI
BLLBL
BLLBN
BLLBT
BLLBU
BLLEB
BLLEE
BLLEI
BLLEL
BLLEN
BLLET
BLLEU
BLLIB
BLLIE
BLLIL
BLLIN
BLLIT
BLLIU
BLLNB
BLLNE
BLLNI
BLLNL
BLLNN
BLLNT
BLLNU
BLLTB
BLLTE
BLLTI
BLLTL

```
BLLTN
BLLTT
BLLTU
BLLUB
BLLUE
BLLUI
BLLUL
BLLUN
BLLUT
BLNBB
BLNBE
BLNBI
BLNBL
BLNBN
BLNBT
BLNBU
BLNEB
BLNEE
BLNEI
BLNEL
BLNEN
BLNET
BLNEU
BLNIB
BLNIE
BLNIL
BLNIN
BLNIT
BLNIU
BLNLB
BLNLE
BLNLI
BLNLL
BLNLN
BLNLT
BLNLU
BLNNB
BLNNE
BLNNI
BLNNL
BLNNT
BLNNU
BLNTB
BLNTE
BLNTI
BLNTL
BLNTN
BLNTT
```

```
BLNTU
BLNUB
BLNUE
BLNUI
BLNUL
BLNUN
BLNUT
BLTBB
BLTBE
BLTBI
BLTBL
BLTBN
BLTBT
BLTBU
BLTEB
BLTEE
BLTEI
BLTEL
BLTEN
BLTET
BLTEU
BLTIB
BLTIE
BLTIL
BLTIN
BLTIT
BLTIU
BLTLB
BLTLE
BLTLI
BLTLL
BLTLN
BLTLT
BLTLU
BLTNB
BLTNE
BLTNI
BLTNL
BLTNN
BLTNT
BLTNU
BLTTB
BLTTE
BLTTI
BLTTL
BLTTN
BLTTU
BLTUB
```

```
BLTUE
BLTUI
BLTUL
BLTUN
BLTUT
BLUBB
BLUBE
BLUBI
BLUBL
BLUBN
BLUBT
BLUBU
BLUEB
BLUEE
BLUEI
BLUEL
BLUEN
BLUET
BLUEU
BLUIB
BLUIE
BLUIL
BLUIN
BLUIT
BLUIU
BLULB
BLULE
BLULI
BLULL
BLULN
BLULT
BLULU
BLUNB
BLUNE
BLUNI
BLUNL
BLUNN
BLUNT
BLUNU
BLUTB
BLUTE
BLUTI
BLUTL
BLUTN
BLUTT
BLUTU
```

[22]:

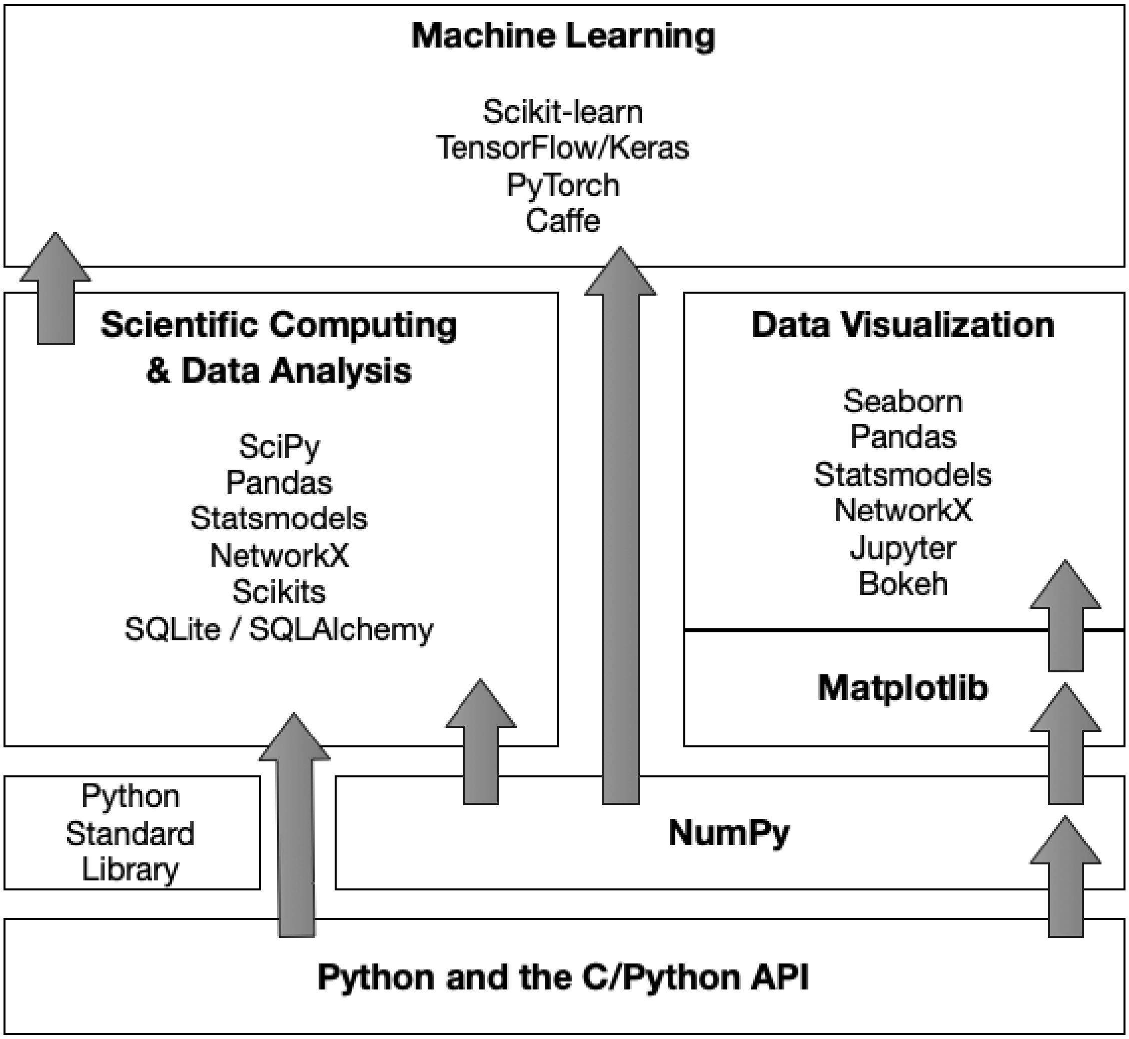
```
%whos
```

```
Variable            Type              Data/Info
---------------------------------------------
InteractiveShell    MetaHasTraits     <class
'IPython.core.inte<...>eshell.InteractiveShell'>
a_dictionary        dict              n=3
allwords            list              n=343
available           str               BEILNTU
c                   str               E
center              str               B
concatenate         function          <function concatenate at 0x10749dee0>
denominator         int               0
ex                  str               LLL
exclude             set               {'II', 'BBB', 'UU', 'NNN', 'EEE', 'TTT',
'LLL'}
filename            str               a_file_that_does_not_exist.txt
i                   int               16
itertools           module            <module 'itertools' (built-in)>
key                 str               C
length              int               5
mapping             dict              n=52
not_excluded        bool              False
squares             list              n=10
start               str               BL
string              module            <module 'string' from
'/U<...>lib/python3.9/string.py'>
value               int               3
w                   str               BLUUU
words               function          <function words at 0x10749de50>
x                   int               3
y                   int               14
z                   int               25
```

5 *Python* as a software ecosystem

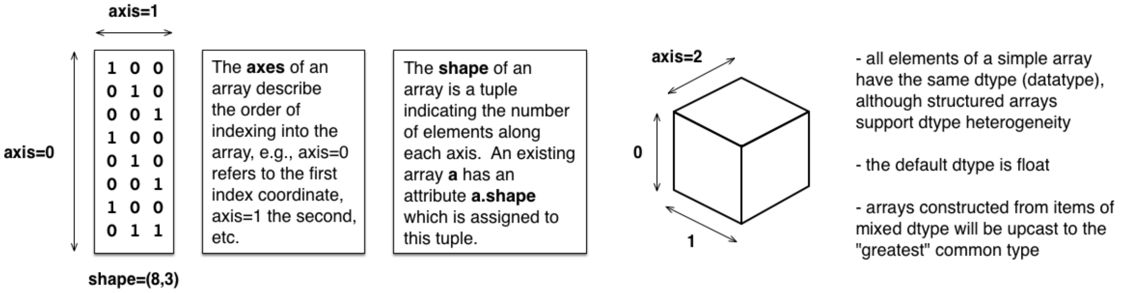
- The core Python language provides a substrate
  - for using programming constructs to define functions, classes, and control flows
  - for importing and using functions and classes defined in external packages
- Actually, Python consists of multiple ecosystems used for different tasks
  - a scripting environment used in operating systems and for systems administration tasks
  - a set of tools for web programming and website development
  - a set of packages for generation of graphical user interfaces (GUIs)
  - an environment for scientific computing, data science, and machine learning
- Python Standard Library: https://docs.python.org/3/library/index.html

5.1 Python for Scientific Computing, Data Science, and Machine Learning



5.1.1 NumPy (Numerical Python)

- multidimensional arrays (ndarray = "N-dimensional array")
- "array syntax" enabling compact expressions and efficient computations
- access to functionality for linear algebra and random numbers
- a substrate for array-based computations throughout the Python ecosystem
- similar in spirit to the role that arrays/matrices play in MATLAB
  - see https://numpy.org/doc/stable/user/numpy-for-matlab-users.html



$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{matrix}$$

The **axes** of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which is assigned to this tuple.

- all elements of a simple array have the same dtype (datatype), although structured arrays support dtype heterogeneity

- the default dtype is float

- arrays constructed from items of mixed dtype will be upcast to the "greatest" common type

5.1.2 Numpy

```
[23]: import numpy as np

      x = np.array([[1,2,3], [4,5,9], [7,8,9]])
      y = np.random.random((3,3))

      w = 3*x + 4*y

      x
      y
      w

      x.sum(axis=0)
```

```
[23]: array([[1, 2, 3],
             [4, 5, 9],
             [7, 8, 9]])

[23]: array([[0.61288401, 0.91090789, 0.56815764],
             [0.30120234, 0.9396962 , 0.06910196],
             [0.91803854, 0.85746682, 0.06649826]])

[23]: array([[ 5.45153604,  9.64363156, 11.27263056],
             [13.20480935, 18.7587848 , 27.27640783],
             [24.67215416, 27.42986727, 27.26599303]])

[23]: array([12, 15, 21])
```

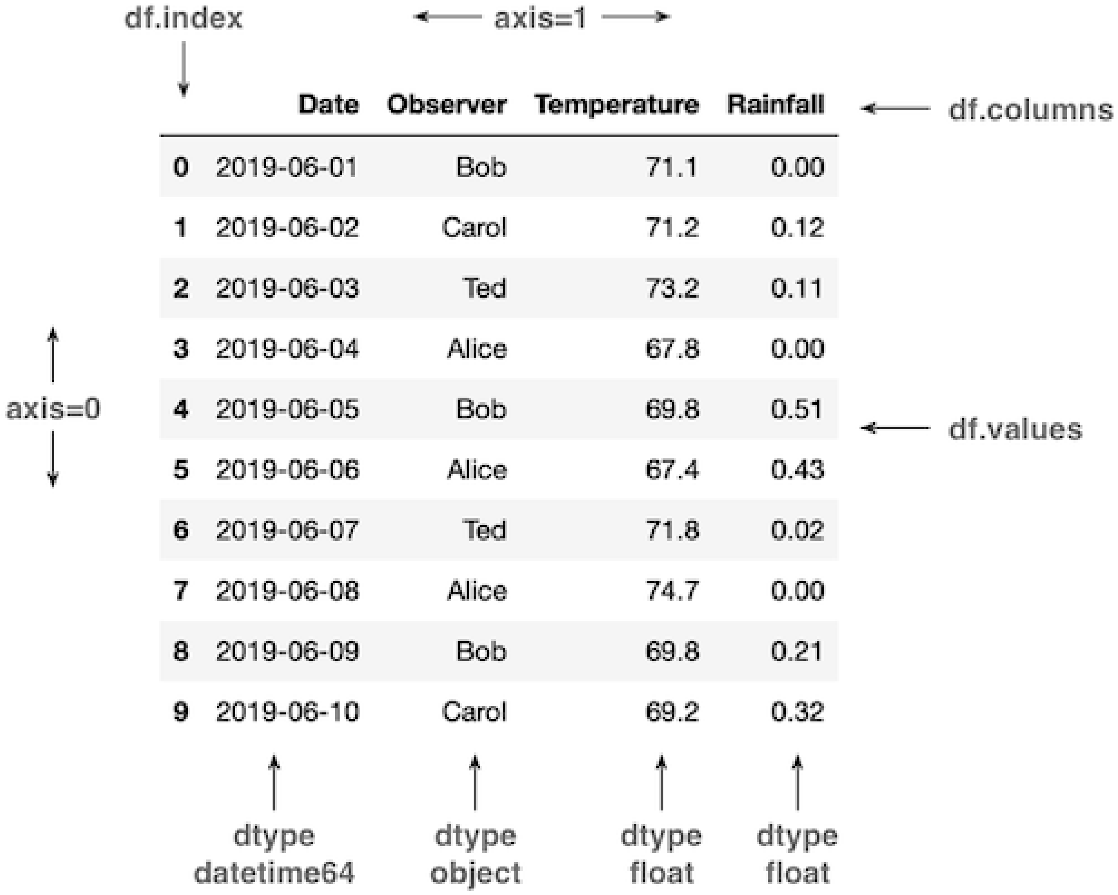
5.1.3 SciPy (Scientific Python)

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fft)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse eigenvalue problems with ARPACK
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial data structures and algorithms (scipy.spatial)
- Statistics (scipy.stats)
- Multidimensional image processing (scipy.ndimage)
- File IO (scipy.io)

5.1.4 Pandas

- DataFrames and Series for dealing with tabular data (e.g., spreadsheets)

- uses NumPy underneath for much of the data processing
- Support for:
  - reading from csv/excel files and SQL databases (and dealing with missing data)
  - adding new columns derived from existing columns
  - groupby functions that perform aggegrate computations over subsets of data
  - lots more



	Date	Observer	Temperature	Rainfall
0	2019-06-01	Bob	71.1	0.00
1	2019-06-02	Carol	71.2	0.12
2	2019-06-03	Ted	73.2	0.11
3	2019-06-04	Alice	67.8	0.00
4	2019-06-05	Bob	69.8	0.51
5	2019-06-06	Alice	67.4	0.43
6	2019-06-07	Ted	71.8	0.02
7	2019-06-08	Alice	74.7	0.00
8	2019-06-09	Bob	69.8	0.21
9	2019-06-10	Carol	69.2	0.32

See our Cornell Virtual Workshop (CVW) topic on Python for Data Science:

- Data Processing and Visualization: https://cvw.cac.cornell.edu/pydatasci1
- Data Modeling and Machine Learning: https://cvw.cac.cornell.edu/pydatasci2

5.2 Python for Data Visualization

- Tools for generating figures and images
  - Matplotlib: the cornerstone and workhorse of the Python data visualization universe
    - Pandas: uses Matplotlib for visualizing data from DataFrames
    - Seaborn: uses Matplotlib with a focus on statistical distributions and multivariate relationships
    - Statsmodels: uses Matplotlib for plotting results of statistical modeling (e.g., regressions)
  - Plotnine: a Python implementation of the “grammar of graphics” (R/gpplot2)
- Tools for generating interactive data visualizations

  - Bokeh, Plotly, Altair
- Tools for 3D visualization of 3D objects
  - VTK, Paraview, Mayavi

5.2.1 Plotting with matplotlib

[24]:

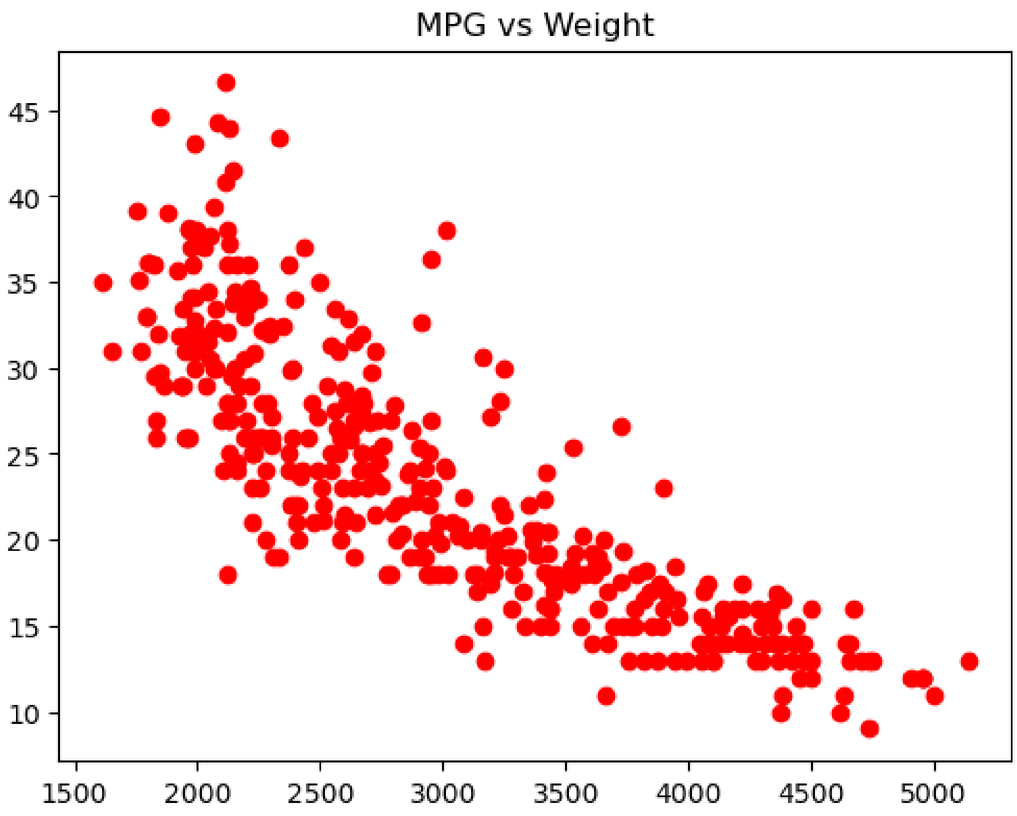
```
import pandas as pd
from bokeh.sampledata.autompg import autompg_clean as df

df.head()
```

[24]:

```
    mpg  cyl  displ   hp  weight  accel  yr         origin  \
0  18.0    8  307.0  130    3504   12.0  70  North America
1  15.0    8  350.0  165    3693   11.5  70  North America
2  18.0    8  318.0  150    3436   11.0  70  North America
3  16.0    8  304.0  150    3433   12.0  70  North America
4  17.0    8  302.0  140    3449   10.5  70  North America

                        name        mfr
0  chevrolet chevelle malibu  chevrolet
1          buick skylark 320      buick
2         plymouth satellite   plymouth
3              amc rebel sst        amc
4                ford torino       ford
```

[25]:

```
import matplotlib.pyplot as plt

plt.scatter(df.weight, df.mpg, color='red')
plt.title('MPG vs Weight');
```



5.2.2 Interative plotting with bokeh

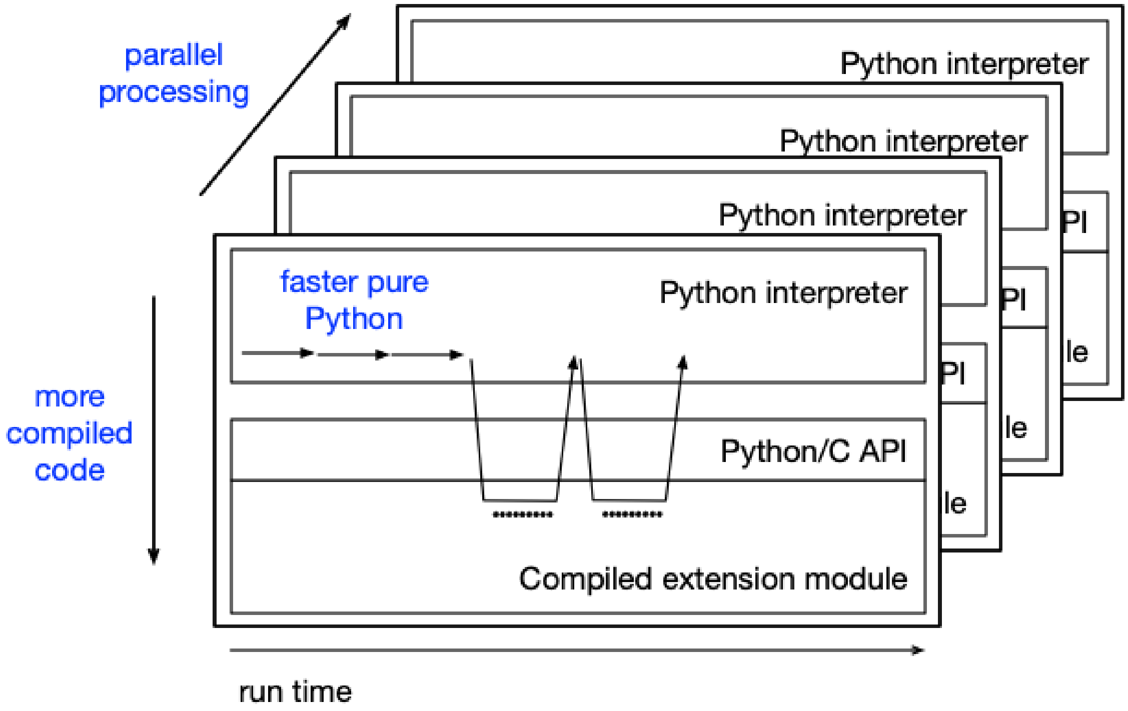
[26]:
```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
from bokeh.io import output_notebook
output_notebook()

p = figure()

source = ColumnDataSource(df)
hover_tips = [(c, "@"+c) for c in source.column_names]

p = figure(tools='pan,box_zoom,hover,reset', tooltips = hover_tips, width=400,␣
 ↪height=400)

p.circle(x='weight', y='mpg', source=source, size=10, color='green', alpha=0.5)
p.xaxis.axis_label = 'weight'
p.yaxis.axis_label = 'mpg'

show(p)
```

[26]:
```
GlyphRenderer(id='1066', ...)
```

5.3 Python for Machine Learning and Deep Learning

- Scikit-learn (sklearn)
  - a large variety of algorithms and lots of documentation about different ML methods
  - classification, regression, clustering, dimensionality reduction, model selection, etc.
  - estimators, pre-processors, transformers, pipelines
- Deep Learning with Neural Networks
  - TensorFlow / Keras ; PyTorch ; Caffe
  - widely used for a broad array of tasks, such as image classification, speech recognition, text generation, protein structure prediction, etc.
  - packages extend numpy-like arrays with the power of *automatic differentation* to support gradient computations and backpropagation for use in training neural networks

5.4 Accelerating Python Code (Python and Performance)



5.4.1 Array operations with NumPy

[ ]:
```
import numpy as np
a = np.random.random((1000,1000))
b = np.random.random((1000,1000))

c = a + b     # throws ValueError if a and b not the same shape
```

[ ]:

```
%%timeit

c = a + b
```

[ ]:

```
%%timeit

assert(a.shape == b.shape) # throws AssertionError if a and b not the same shape
c = np.zeros_like(a) # prefills a zero array of the correct shape

for i in range(a.shape[0]):
    for j in range(a.shape[1]):
        c[i,j] = a[i,j] + b[i,j]
```

[ ]:

```
# Speedup?
```

5.5 An Introduction to Python and an overview of possible future topics

- ~~Introduction to Python~~
- Python for Scientific Computing and Data Science
- Python for Data Visualization
- Python for Machine Learning and Deep Learning
- Accelerating Python Code

6 Python as a language and an ecosystem

- An expressive programming language for crafting custom analyses and workflows
- A rich set of interoperating packages and libraries for processing data and investigating complex systems

6.1 Any Questions?

[ ]:

```

```

7 Supplemental material

7.1 Putting the pieces together: Ciphers

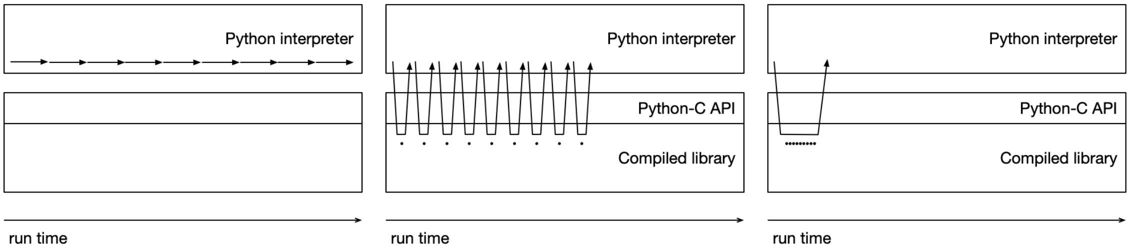
[ ]:

```
letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
cipher = {letters[i]: letters[(i-3) % len(letters)] for i in range(len(letters))}

cipher
```

[ ]:

```
def transform_message(message, cipher):
    tmsg = ''
    for c in message:
        tmsg = tmsg + cipher.get(c, c)
    return tmsg

test = "I come to bury Caesar, not to praise him."

transform_message(test, cipher)

decoder = {v:k for k,v in cipher.items()}

transform_message(transform_message(test, cipher), decoder)
```

7.2 Compiled extension modules



7.2.1 Plotting with pandas (and tweaking with matplotlib)

[ ]:

```
df.plot.scatter(x="weight", y="mpg", color='blue')
plt.title('MPG vs Weight');
```