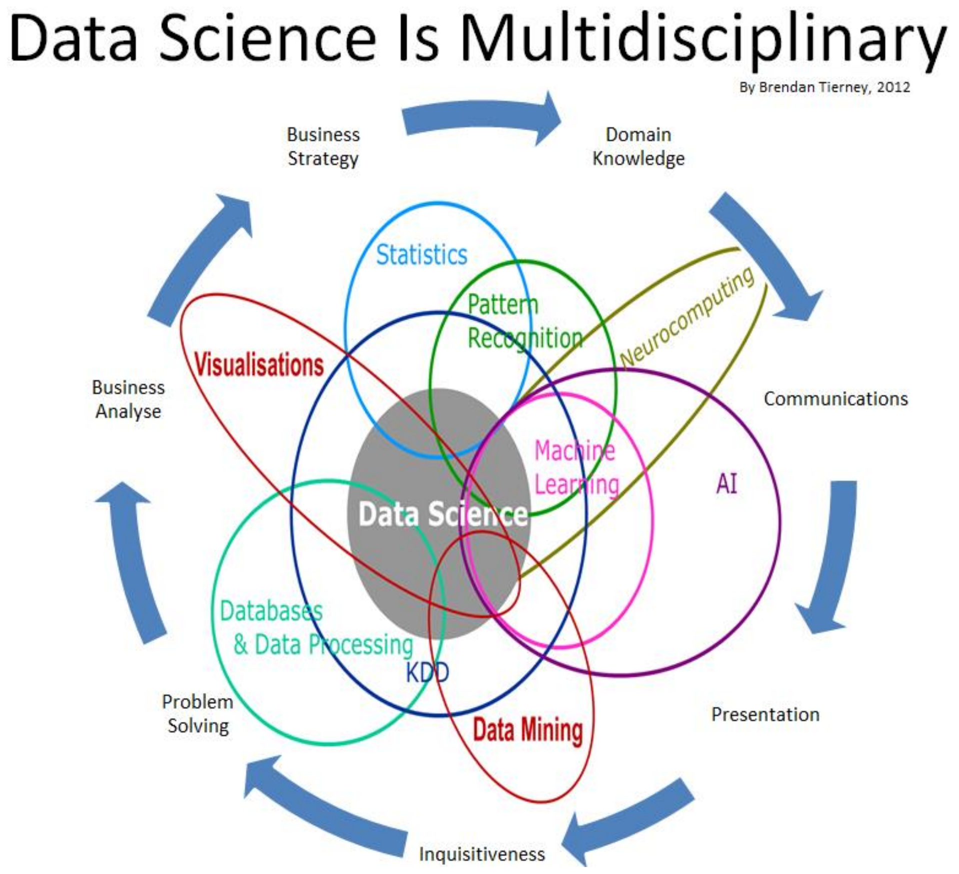
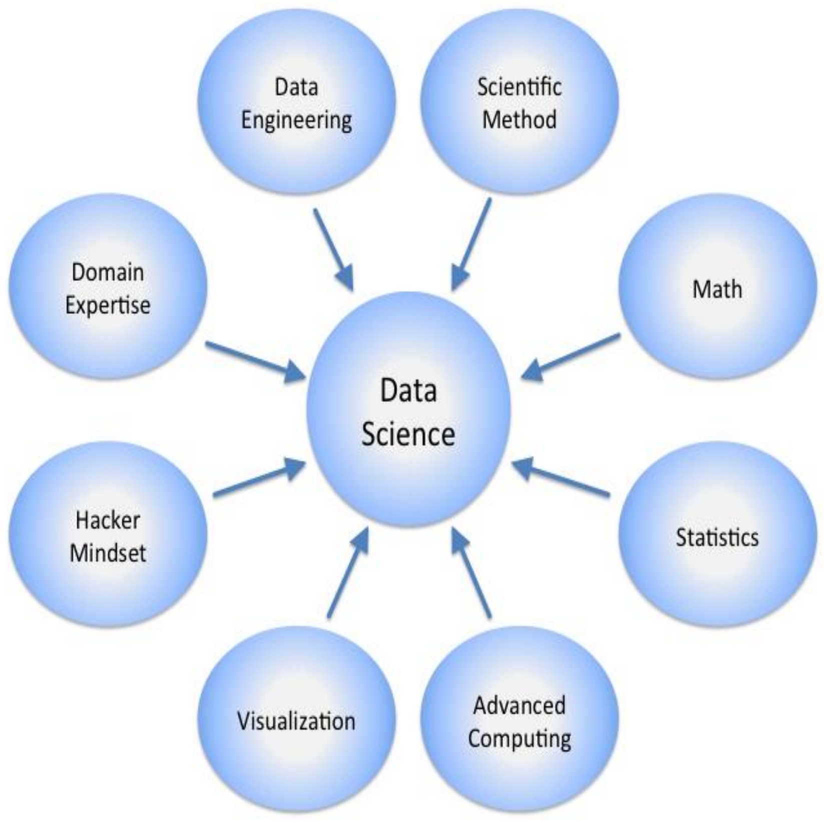
PYTHON PROGRAMMING FOR DATA SCIENCE

Anasua Sarkar
Computer Science & Engineering Department
Jadavpur University
anasua.sarkar@jadavpuruniversity.in

Data Science – A Definition

Data Science is the science which uses computer science, statistics and machine learning, visualization and human-computer interactions to collect, clean, integrate, analyze, visualize, interact with data to create data products.





Python : Introduction

- Most recent popular (scripting/extension) language
  - although origin ~1991
- heritage: teaching language (ABC)
  - Tcl: shell
  - perl: string (regex) processing
- object-oriented
- It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals, manipulating regular expressions, and unit testing.
- Large organizations that use Python include Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram and Spotify. The social news networking site Reddit is written entirely in Python.

What's Python?

- Python is a general-purpose, interpreted high-level programming language.
- Its syntax is clear and emphasize readability.
- Python has a large and comprehensive standard library.
- Python supports multiple programming paradigms, primarily but not limited to procedural, object-oriented, event-driven and, to a lesser extent, functional programming styles.
- It features a fully dynamic type system and automatic memory management

It’s Named After Monty Python

- Despite all the reptile icons in the Python world, the truth is that Python creator Guido van Rossum named it after the BBC comedy series Monty Python’s Flying Circus. He is a big fan of Monty Python, as are many software developers (indeed, there seems to almost be a symmetry between the two fields).
- This legacy inevitably adds a humorous quality to Python code examples. For instance, the traditional “foo” and “bar” for generic variable names become “spam” and “eggs” in the Python world. The occasional “Brian,” “ni,” and “shrubbery” likewise owe their appearances to this namesake. It even impacts the Python community at large: talks at Python conferences are regularly billed as “The Spanish Inquisition.”
- All of this is, of course, very funny if you are familiar with the show, but less so otherwise.

Timeline

- Python born, name picked - Dec 1989
  - By Guido van Rossum, the officially anointed Benevolent Dictator for Life (BDFL) of Python
- First public release (USENET) - Feb 1991
- python.org website - 1996 or 1997
- 2.0 released - 2000
- Python Software Foundation - 2001
- …
- 2.4 released - 2004
- 2.5 released – 2006
- 3.0 released - 2008
- Current versions: Python 2.7.13 and Python 3.8.2.

Extensions in Python 2.6 and 3.0

- The print function in 3.0
- The nonlocal x,y statement in 3.0
- The str.format method in 2.6 and 3.0
- String types in 3.0: str for Unicode text, bytes for binary data
- Text and binary file distinctions in 3.0
- Class decorators in 2.6 and 3.0: @private('age')
- New iterators in 3.0: range, map, zip
- Dictionary views in 3.0: D.keys, D.values, D.items
- Division operators in 3.0: remainders, / and //
- Set literals in 3.0: {a, b, c}
- Set comprehensions in 3.0: {x**2 for x in seq}
- Dictionary comprehensions in 3.0: {x: x**2 for x in seq}
- Binary digit-string support in 2.6 and 3.0: 0b0101, bin(I)

Extensions in Python 2.6 and 3.0

- The fraction number type in 2.6 and 3.0: Fraction(1, 3)
- Function annotations in 3.0: def f(a:99, b:str)->int
- Keyword-only arguments in 3.0: def f(a, *b, c, **d)
- Extended sequence unpacking in 3.0: a, *b = seq
- Relative import syntax for packages enabled in 3.0: from .
- Context managers enabled in 2.6 and 3.0: with/as
- Exception syntax changes in 3.0: raise, except/as, superclass
- Exception chaining in 3.0: raise e2 from e1
- Reserved word changes in 2.6 and 3.0
- New-style class cutover in 3.0
- Property decorators in 2.6 and 3.0: @property
- Descriptor use in 2.6 and 3.0
- Metaclass use in 2.6 and 3.0
- Abstract base classes support in 2.6 and 3.0

Portability

- As a partial list, Python
- is available on:
- • Linux and Unix systems
- • Microsoft Windows and DOS (all modern flavors)
- • Mac OS (both OS X and Classic)
- • BeOS, OS/2, VMS, and QNX
- • Real-time systems such as VxWorks
- • Cray supercomputers and IBM mainframes
- • PDAs running Palm OS, PocketPC, and Linux
- • Cell phones running Symbian OS and Windows Mobile
- • Gaming consoles and iPods
- • And more
- Like the language interpreter itself, the standard library modules that ship with Python are implemented to be as portable across platform boundaries as possible. Further, Python programs are automatically compiled to portable byte code, which runs the same on any platform with a compatible version of Python installed

Python interpreter

- When the Python package is installed on your machine, it generates a number of components—minimally, an interpreter and a support library. Depending on how you use it, the Python interpreter may take the form of an executable program, or a set of libraries linked into another program. Depending on which flavor of Python you run, the interpreter itself may be implemented as a C program, a set of Java classes, or something else. Whatever form it takes, the Python code you write must always be run by this interpreter. And to enable that, you must install a Python interpreter on your computer.

- C:\temp> python script0.py
- hello world
- 1267650600228229401496703205376

Python’s view

- When you instruct Python to run your script, there are a few steps that Python carries out before your code actually starts crunching away. Specifically, it’s first compiled to something called “byte code” and then routed to something called a “virtual machine.”
- Byte code compilation
    - Compilation is simply a translation step, and byte code is a lower-level, platform-independent representation of your source code. Roughly, Python translates each of your source statements into a group of byte code instructions by decomposing them into individual steps. This byte code translation is performed to speed execution.
    - If the Python process has write access on your machine, it will store the byte code of your programs in files that end with a .pyc extension (“.pyc” means compiled “.py” source).
    - The next time you run your program, Python will load the .pyc files and skip the compilation step, as long as you haven’t changed your source code since the byte code was last saved. Python automatically checks the timestamps of source and byte code files to know when it must recompile—if you resave your source code, byte code is automatically re-created the next time your program is run.

The Python Virtual Machine (PVM)

- It’s not a separate program, and it need not be installed by itself. In fact, the PVM is just a big loop that iterates through your byte code instructions, one by one, to carry out their operations.
- The PVM is the runtime engine of Python

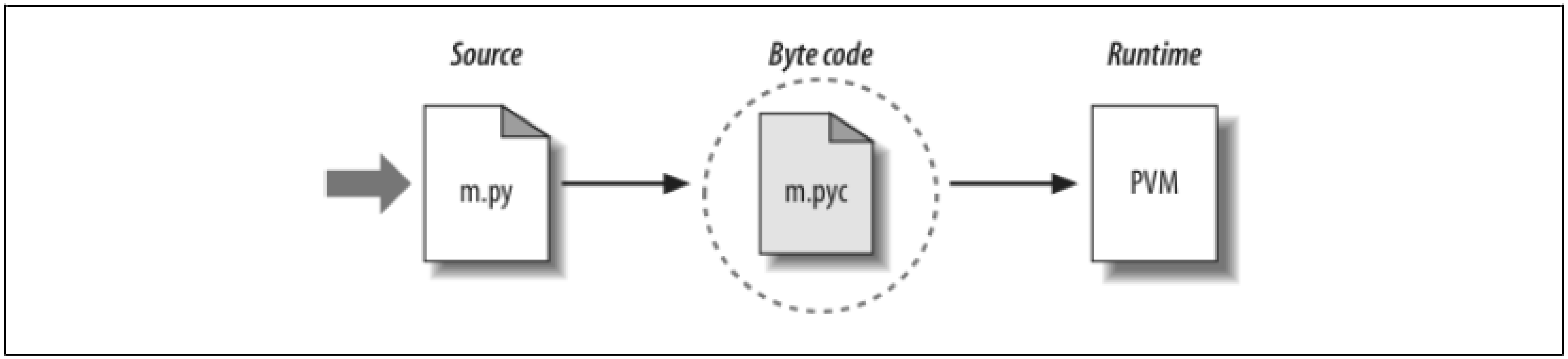


Figure 2-2. Python’s traditional runtime execution model: source code you type is translated to byte code, which is then run by the Python Virtual Machine. Your code is automatically compiled, but then it is interpreted.

Differences from C, C++

- For one thing, there is usually no build or "make" step in Python work: code runs immediately after it is written. For another, Python byte code is not binary machine code (e.g., instructions for an Intel chip). Byte code is a Python-specific representation.
  - This is why some Python code may not run as fast as C or C++ code —the PVM loop, not the CPU chip, still must interpret the byte code, and byte code instructions require more work than CPU instructions. On the other hand, unlike in classic interpreters, there is still an internal compile step—Python does not need to reanalyze and reparse each source statement repeatedly. The net effect is that pure Python code runs at speeds somewhere between those of a traditional compiled language and a traditional interpreted language.
  - there is really no distinction between the development and execution environments. That is, the systems that compile and execute your source code are really one and the same.

To run other Python program at runtime

- The eval and exec built-ins, for instance, accept and run strings containing Python program code.
- This structure is also why Python lends itself to product customization—because Python code can be changed on the fly, users can modify the Python parts of a system onsite without needing to have or compile the entire system’s code.

Python Implementation Alternatives

- Cpython
  - The original, and standard, implementation of Python is usually called Cpython. This is the Python that you fetch from http://www
  - .python.org, get with the ActivePython distribution, and have automatically on most Linux and Mac OS X machines
- Jython
  - The Jython system (originally known as JPython) is an alternative implementation of the Python language, targeted for integration with the Java programming language.
  - Jython consists of Java classes that compile Python source code to Java byte code and then route the resulting byte code to the Java Virtual Machine (JVM). Programmers still code Python statements in .py text files as usual; the Jython system essentially just replaces the rightmost two bubbles in Figure 2-2 with Java-based equivalents.
- Jython scripts can serve as web applets and servlets, build Java-based GUIs, and so on. Because Jython is slower and less robust than CPython, though, it is usually seen as a tool of interest primarily to Java developers looking for a scripting language to be a
- frontend to Java code.

Removals in Python 3.0

Removed	Replacement
reload(M)	imp.reload(M) (or exec)
apply(f, ps, ks)	f(*ps, **ks)
\`X\`	repr(X)
X <> Y	X != Y
long	int
9999L	9999
D.has_key(K)	K in D (or D.get(key) != None)
raw_input	input
old input	eval(input())
xrange	range
file	open (and io module classes)
X.next	X.__next__, called by next(X)
X.__getslice__	X.__getitem__ passed a slice object
X.__setslice__	X.__setitem__ passed a slice object
reduce	functools.reduce (or loop code)
execfile(filename)	exec(open(filename).read())
exec open(filename)	exec(open(filename).read())
0777	0o777
print x, y	print(x, y)

Removals in Python 3.0

Removed	Replacement
`print >> F, x, y`	`print(x, y, file=F)`
`print x, y,`	`print(x, y, end=' ')`
`u'ccc'`	`'ccc'`
`'bbb'` for byte strings	`b'bbb'`
`raise E, V`	`raise E(V)`
`except E, X:`	`except E as X:`
`def f((a, b)):`	`def f(x): (a, b) = x`
`file.xreadlines`	`for line in file:` (or `X=iter(file)`)
`D.keys()`, etc. as lists	`list(D.keys())` (dictionary views)
`map()`, `range()`, etc. as lists	`list(map())`, `list(range())` (built-ins)
`map(None, ...)`	`zip` (or manual code to pad results)
`X=D.keys(); X.sort()`	`sorted(D)` (or `list(D.keys())`)
`cmp(x, y)`	`(x > y) - (x < y)`
`X.__cmp__(y)`	`__lt__`, `__gt__`, `__eq__`, etc.
`X.__nonzero__`	`X.__bool__`
`X.__hex__`, `X.__oct__`	`X.__index__`
Sort comparison functions	Use `key=transform` or `reverse=True`
Dictionary `<`, `>`, `<=`, `>=`	Compare `sorted(D.items())` (or loop code)
`types.ListType`	`list` (`types` is for nonbuilt-in names only)
`__metaclass__ = M`	`class C(metaclass=M):`
`__builtin__`	`builtins` (renamed)
`Tkinter`	`tkinter` (renamed)
`sys.exc_type`, `exc_value`	`sys.exc_info()[0]`, `[1]`
`function.func_code`	`function.__code__`
`__getattr__` run by built-ins	Redefine `__X__` methods in wrapper classes
`-t`, `-tt` command-line switches	Inconsistent tabs/spaces use is always an error
`from ... *`, within a function	May only appear at the top level of a file
`import mod`, in same package	`from . import mod`, package-relative form
`class MyException:`	`class MyException(Exception):`
`exceptions` module	Built-in scope, library manual
`thread`, `Queue` modules	`_thread`, `queue` (both renamed)
`anydbm` module	`dbm` (renamed)
`cPickle` module	`_pickle` (renamed, used automatically)
`os.popen2/3/4`	`subprocess.Popen` (`os.popen` retained)
String-based exceptions	Class-based exceptions (also required in 2.6)

Removed	Replacement
String module functions	String object methods
Unbound methods	Functions (`staticmethod` to call via instance)
Mixed type comparisons, sorts	Nonnumeric mixed type comparisons are errors

Changes from 2.6 to 3.0

- Changes in the Standard Library
- 2to3 automatic code conversion code available in Python 3.0

Python Language changes

- The new B if A else C conditional expression
- • with/as context managers
- • try/except/finally unification
- • Relative import syntax
- • Generator expressions
- • New generator function features
- • Function decorators
- • The set object type
- • New built-in functions: sorted, sum, any, all, enumerate
- • The decimal fixed-precision object type
- • Files, list comprehensions, and iterators
- • New development tools: Eclipse, distutils, unittest and doctest, IDLE enhancements,
- Shedskin, and so on.

Key features of Python

- Python programs can run on any platform, you can carry code created in Windows machine and run it on Mac or Linux.
- Python has inbuilt large library with prebuilt and portable functionality, also known as the standard library.
- Python is an expressive language.
- Python is free and open source.
- Python code is about one third of the size of equivalent C++ and Java code.
- Python can be both dynamically and strongly typed—
  - Dynamically typed means it is a type of variable that is interpreted at runtime, which means, in Python, there is no need to define the type (int or float) of the variable.

What is python?

- Object oriented language - polymorphism, operator overloading, and multiple inheritance, ideal as a scripting tool - For example, with the appropriate glue code, Python programs can subclass (specialize) classes implemented in C++, Java, and C#.
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- High-level internal object data types
- Automatic memory management
- It’s free (open source)!

Advantages of Python

Advantages of Python

- Simple
- Easy to study, reusable and maintainable
- Free and open source
- High-level programming language
- Portability - Programs run unchanged on all major computer platforms
- Expansibility, Component integration, Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms.
- Embedability
- Large and comprehensive standard libraries
- Canonical code
- Readability, coherence, and software quality
- Python code can invoke C and C++ libraries, can be called from C and C++ programs, can integrate with Java and .NET components, can communicate over frameworks such as COM, can interface with devices over serial ports, and can interact over networks with interfaces like SOAP, XML-RPC, and CORBA.

@Yang Li

High-level data types

- Basic types - int, float, complex, bool, bytes
- Strings: immutable
- Container types – tuples, sets, Lists and dictionaries
- User-defined types - class
- Other types for e.g. binary data, regular expressions
- Extension modules can define new “built-in” data types

Why learn python? (cont.)

- Reduce development time
- Reduce code length
- Easy to learn and use as developers
- Easy to understand codes
- Easy to do team projects
- Easy to extend to other languages
- use GUI support in other toolkits in Python, such as Qt with PyQt, GTK with PyGTK, MFC with PyWin32, .NET with IronPython, and Swing with Jython (the Java version of Python or Jpype.
- Python's standard pickle module provides a simple object persistence system—it allows programs to easily save and restore entire Python objects to files and file-like objects.

Where to use python?

- System management (i.e., scripting)
- Systems Programming-shell tools, searching directories,socket, process,thread, Stackless python-multiprocessing system
- Graphic User Interface (GUIs) - Tk GUI API called tkinter, PMW, wxPython,
- GUI support in other toolkits in Python - Qt with PyQt, GTK with PyGTK, MFC with PyWin32, .NET with IronPython, and Swing with Jython or JType
- Internet programming -CGI scripts, XML, FTP, XML-RPC, TelNet : HTMLGen, modPython, Django, TurboGears, web2py, Pylons, Zope, and WebWare – MVC, AJAX
- Database (DB) programming – pickle, ZODB, SQLObject and SQLAlchemy, SQLite
- Component integration – Cython
- Rapid Prototyping
- Text data processing-NLTK
- Distributed processing
- Numerical operations – NumPy, SciPy and ScientificPython
- Graphics –pygame, PIL, PyOpenGL, Blender, Maya
- PyRo , PySol
- And so on…

- Implementations of Python
  - Major implementations include CPython, Jython, IronPython, MicroPython, and PyPy.

Python vs. Perl

- Easier to learn
  - important for occasional users
- More readable code
  - improved code maintenance
- Fewer “magical” side effects
- More “safety” guarantees
- Better Java integration
- Less Unix bias

Python vs. Java

- Code 5-10 times more concise
- Dynamic typing
- Much quicker development
  - no compilation phase
  - less typing
- Yes, it runs slower
  - but development is so much faster!
- Similar (but more so) for C/C++

- Use Python with Java: JPython!

Python features

Lutz, *Programming Python*

no compiling or linking	rapid development cycle
no type declarations	simpler, shorter, more flexible
automatic memory management	garbage collection
high-level data types and operations	fast development
object-oriented programming	code structuring and reuse, C++
embedding and extending in C	mixed language systems
classes, modules, exceptions	"programming-in-the-large" support
dynamic loading of C modules	simplified extensions, smaller binaries
dynamic reloading of C modules	programs can be modified without stopping

Python features

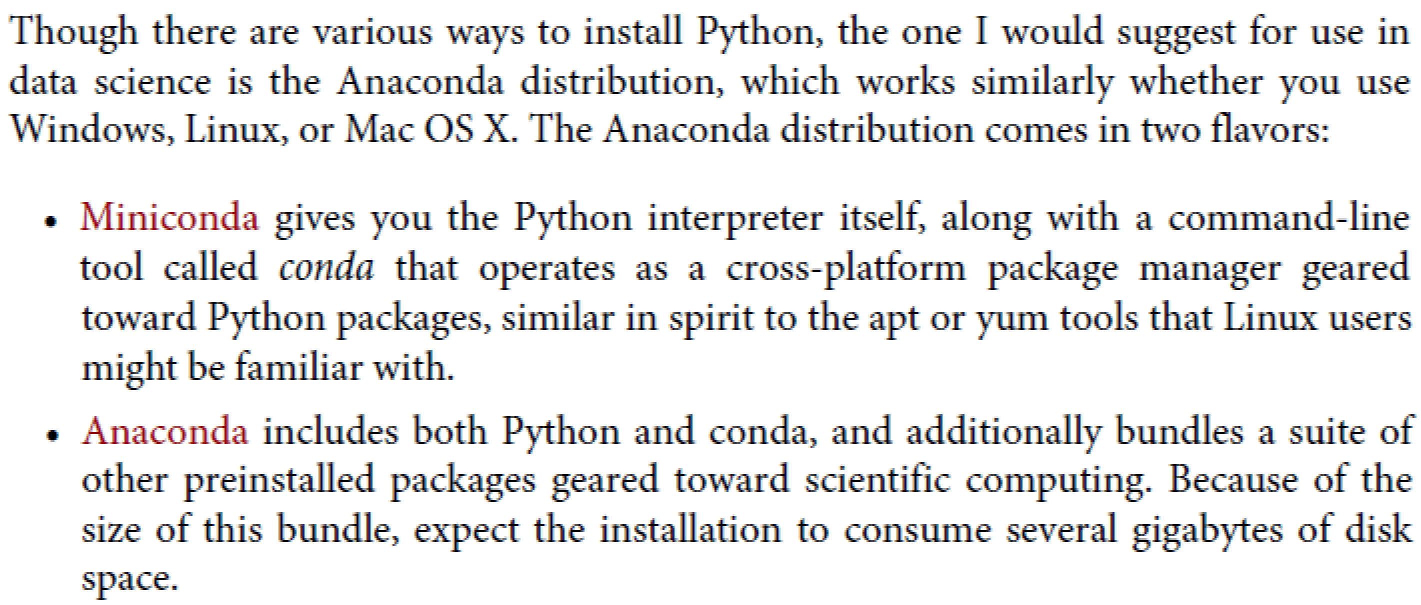
Lutz, *Programming Python*

universal "first-class" object model	fewer restrictions and rules
run-time program construction	handles unforeseen needs, end-user coding
interactive, dynamic nature	incremental development and testing
access to interpreter information	metaprogramming, introspective objects
wide portability	cross-platform programming without ports
compilation to portable byte-code	execution speed, protecting source code
built-in interfaces to external services	system tools, GUIs, persistence, databases, etc.

What not to use Python (and kin) for

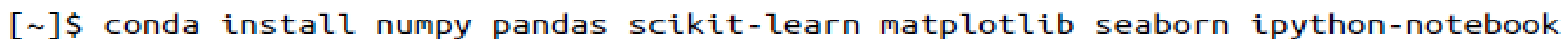
- Most scripting languages share these
- Not as efficient as C
  - but sometimes better built-in algorithms (e.g., hashing and sorting)
- Delayed error notification
- Lack of profiling tools
- Its execution speed may not always be as fast as that of compiled languages such as C and C++.
- Python today compile (i.e., translate) source code statements to an intermediate format known as byte code and then interpret the byte code. Byte code provides portability, as it is a platform-independent format.
- Python has been optimized numerous times, and Python code runs fast enough by itself in most application domains. Furthermore, whenever you do something “real” in a Python script, like processing a file or constructing a graphical user interface (GUI), your program will actually run at C speed, since such tasks are immediately dispatched to compiled C code inside the Python interpreter.

Install Python



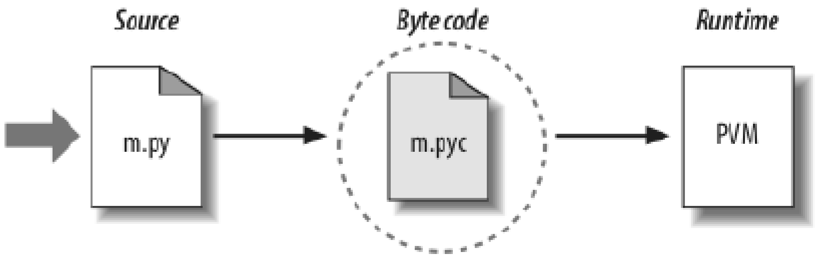
Though there are various ways to install Python, the one I would suggest for use in data science is the Anaconda distribution, which works similarly whether you use Windows, Linux, or Mac OS X. The Anaconda distribution comes in two flavors:

- Miniconda gives you the Python interpreter itself, along with a command-line tool called *conda* that operates as a cross-platform package manager geared toward Python packages, similar in spirit to the apt or yum tools that Linux users might be familiar with.
- Anaconda includes both Python and conda, and additionally bundles a suite of other preinstalled packages geared toward scientific computing. Because of the size of this bundle, expect the installation to consume several gigabytes of disk space.



```
[~]$ conda install numpy pandas scikit-learn matplotlib seaborn ipython-notebook
```

Using python



- /usr/local/bin/python
  - `#! /usr/bin/env python`
- interactive use

```
Python 1.6 (#1, Sep 24 2000, 20:40:45) [GCC 2.95.1 19990816 (release)] on sunos5
Copyright (c) 1995-2000 Corporation for National Research Initiatives.
All Rights Reserved.
Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.
>>>
```

- `python –c` *`command`* `[`*`arg`*`] ...`
- `python –i` *`script`*
  - read script first, then interactive

Running Python Interactively

- Start python by typing "python"
  - /afs/isis/pkg/isis/bin/python
- ^D (control-D) exits
  - % python
  - >>> ^D
  - %
- Comments start with '#'
  - >>> 2+2 #Comment on the same line as text
  - 4
  - >>> 7/3 #Numbers are integers by default
  - 2
  - >>> x = y = z = 0 #Multiple assigns at once
  - >>> z
  - 0

Running Python Programs

- In general
    - % python ./myprogram.py
- Can also create executable scripts
  - Compose the code in an editor like `vi/emacs`
    - % vi ./myprogram.py # Python scripts with the suffix .py.
  - Then you can just type the script name to execute
    - % python ./myprogram.py
- The first line of the program tells the OS how to execute it:
    - #! /afs/isis/pkg/isis/bin/python
  - Make the file executable:
    - % chmod +x ./myprogram.py
  - Then you can just type the script name to execute
    - % ./myprogram.py

Running Python Programs Interactively

Suppose the file `script.py` contains the following lines:

```
print 'Hello world'
x = [0,1,2]
```

Let's run this script in each of the ways described on the last slide:

-
  ```
  python -i script.py
  Hello world
  >>> x
  [0,1,2]
  ```

-
  ```
  $ python
  >>> execfile('script.py')
  >>> x
  [0,1,2]
  ```

Running Python Programs Interactively

# Pretend that script.py contains multiple stored quantities. To promote x(and only x) to the top level context, type the following:

```
$ python
>>> from script import x
Hello world
>>> x
[0,1,2]
>>>
```

# To promote all quantities in `script.py` to the top level context, type `from script import *` into the interpreter. Of course, if that's what you want, you might as well type `python -i script.py` into the terminal.

```
>>> from script import *
```

Python structure

- modules: Python source files
  - import, top-level via from, reload
- statements
  - control flow
  - create objects
  - indentation matters – instead of {}
- objects
  - everything is an object
  - automatically reclaimed when no longer needed

First example

```
#!/usr/local/bin/python
# import systems module
import sys
marker = '::::::'
for name in sys.argv[1:]:
  input = open(name, 'r')
 print marker + name
  print input.read()
```

Python Syntax

- Much of it is similar to C syntax
- Exceptions:
  - missing operators: **++**, **--**
  - no curly brackets, **{ } ,** for blocks; uses whitespace
  - different keywords
  - lots of extra features
  - no type declarations!

Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- ***Everything*** is a variable:
  - functions, modules, classes

Simple data types: operators

- `+ - * / %` (like C) // (floor division)
- `+= -=` etc. (no `++` or `--`)
- Assignment using `=`
  - but semantics are different!

```
a = 1
a = "foo"  # OK
```

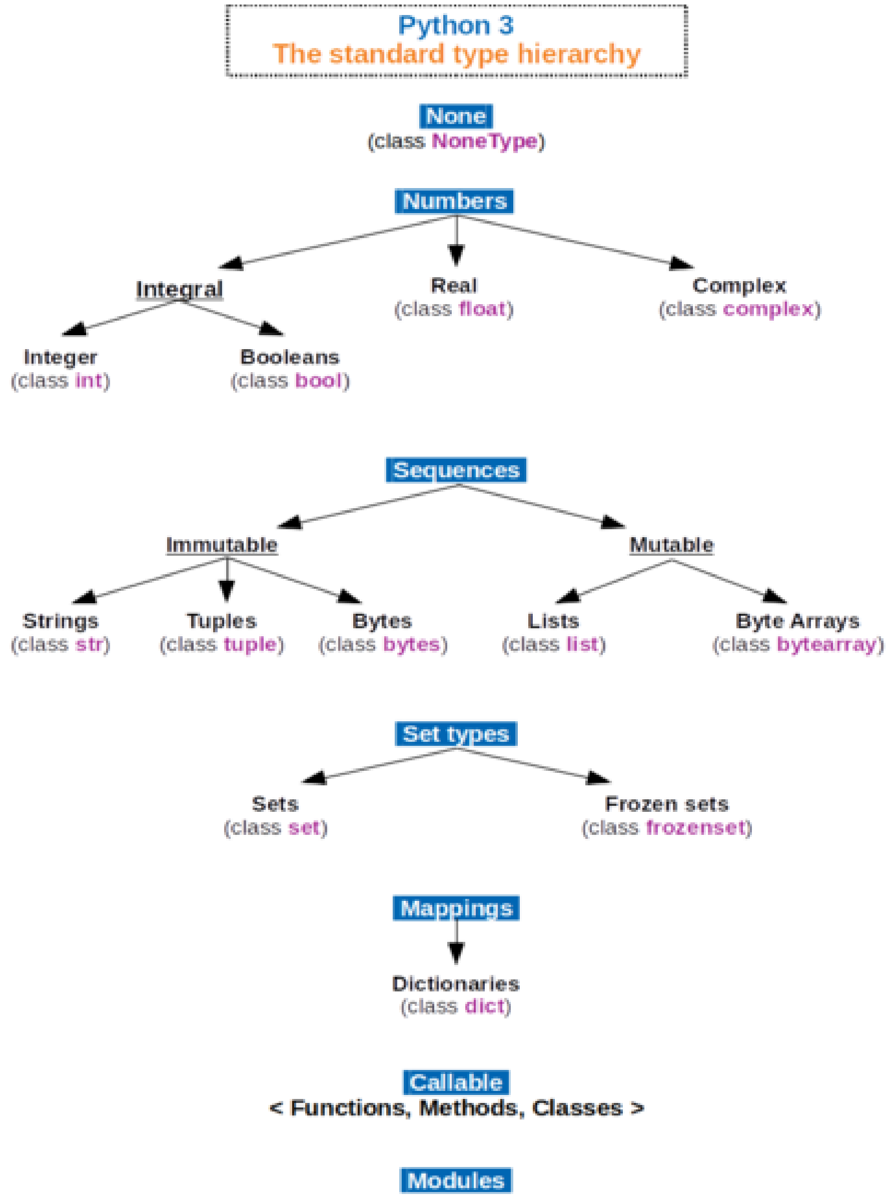
- Can also use `+` to concatenate strings
- Infix operator
- Walrus operator := in Python 3.8

Bitwise Operators

```
>>> X = 0b0001 # Binary literals
>>> X << 2 # Shift left
4
>>> bin(X << 2) # Binary digits string
'0b100'
>>> bin(X | 0b010) # Bitwise OR
'0b11'
>>> bin(X & 0b1) # Bitwise AND
'0b1'
>>> X = 0xFF # Hex literals
>>> bin(X)
'0b11111111'
>>> X ^ 0b10101010 # Bitwise XOR
85
>>> bin(X ^ 0b10101010)
'0b1010101'
>>> int('1010101', 2) # String to int per base
85
>>> hex(85) # Hex digit string
'0x55'
```

Simple data types

- Numbers
  - Integer, floating-point, complex!
- Strings
  - characters are strings of length 1
- Booleans are `False` or `True`



References

- Python Homepage
  - http://www.python.org
- Python Tutorial
  - http://docs.python.org/tutorial/
- Python Documentation
  - http://www.python.org/doc
- Python Library References
  - http://docs.python.org/release/2.5.2/lib/lib.html
- Python Add-on Packages:
  - http://pypi.python.org/pypi
- Data Science from Scratch – First principles with Python, Joel Grus, O'Reilly.
- Python Data Science Handbook – Essential Tools for Working with Data, Jake VanderPlas, O'Reilly.