Python Programming for Scientists

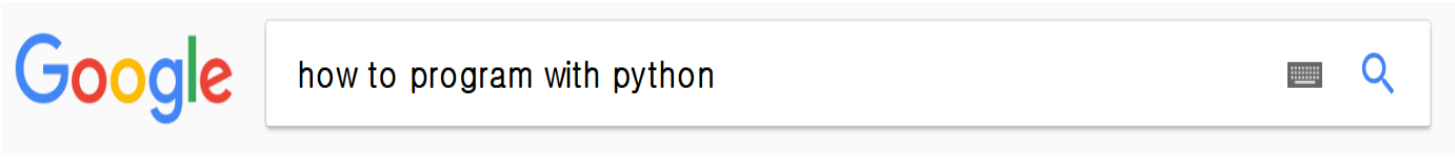
Python basics, SciPy, NumPy, matplotlib, pandas, PyPy, Cython, F2Py, mpi4py

In-Ho Lee
Korea Research Institute of Standards and Science, Daejeon 34113, Korea

Python (1991)

- an open source interpreted programming language for general-purpose programming
- a high-level, interpreted, interactive, and object-oriented scripting language
- a large and comprehensive standard library

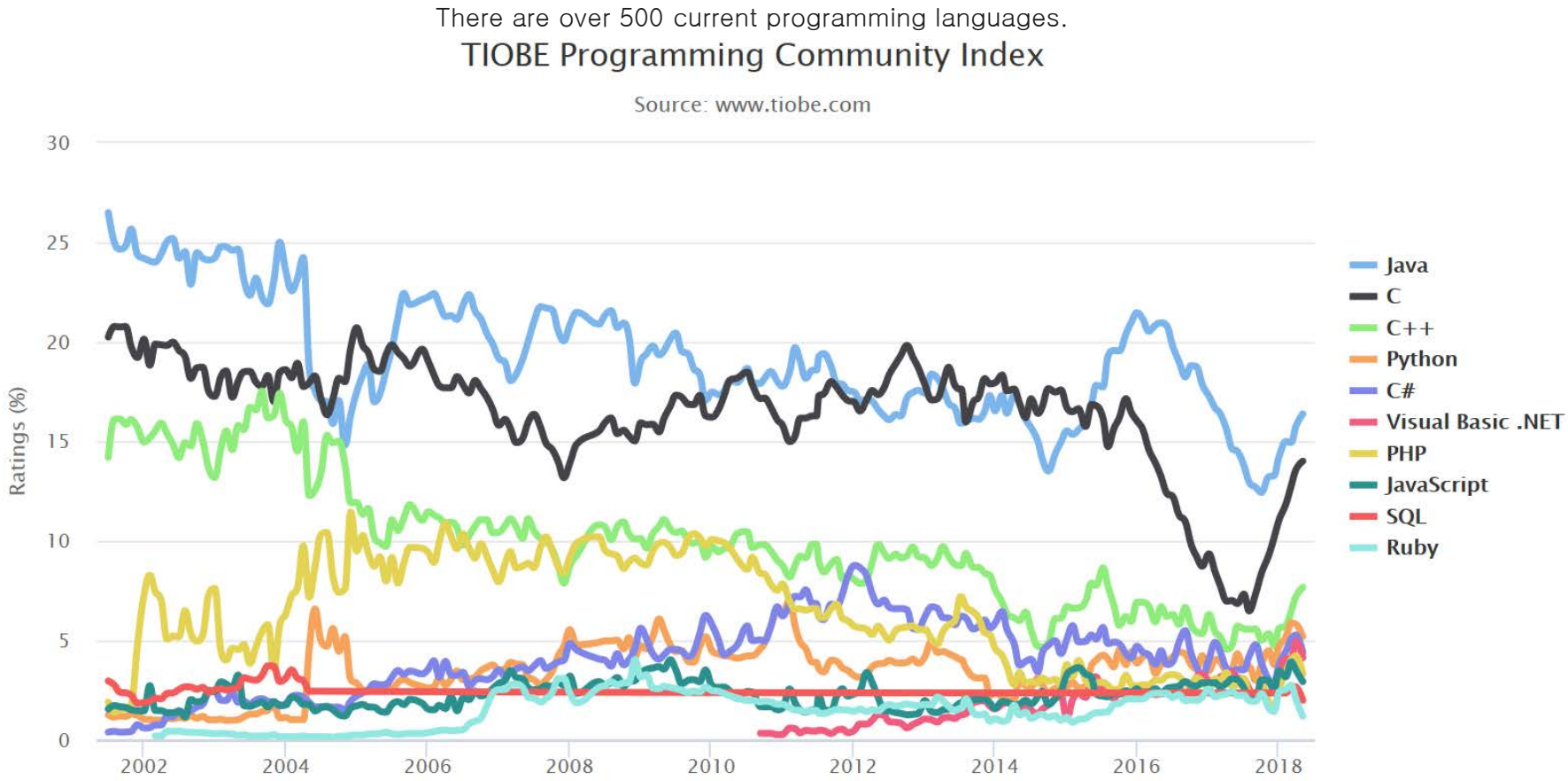
Guido van Rossum



Read → Write

What languages should you learn?

- TIOBE index (Web Search): Java, C, C++, Python, C#
- PYPL index (Google Trends): Java, Python, PHP, C#, Javascript
- RedMonk rating (Github/Stack Overflow): Javascript, Java, Python, PHP, C#

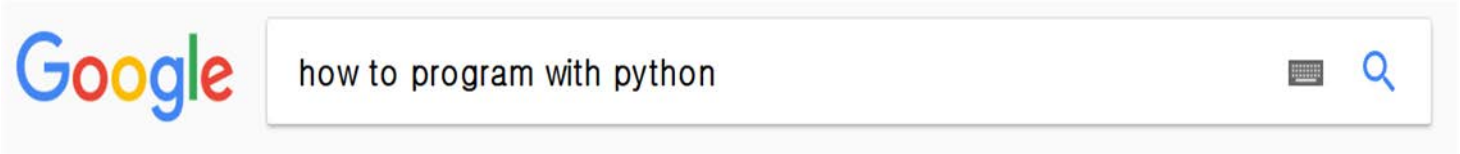


https://www.tiobe.com/tiobe-index/

Python Philosophy

There should be one—and preferably only one—obvious way to do it.

https://en.wikipedia.org/wiki/Python_(programming_language)



https://en.wikipedia.org/wiki/Stack_Overflow
https://en.wikipedia.org/wiki/GitHub

stackoverflow GitHub

"Everything should be made as simple as possible, but no simpler."

https://en.wikiquote.org/wiki/Albert_Einstein

Python 3.x *vs* Python 2.x

- Python 3.0 (2008) backward incompatibles
- Python 2.7 final 2.x version (2010)
- Current Linux distributions and Macs (2.x as default)

python --version

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
#!/usr/bin/python3

print ("Hello, Python!")
```

We want to use these libraries, so we will have to learn a little about Python 2. Fortunately, there are only a few big differences that we have to worry about.

Python 3.x *vs* Python 2.x

- Python 3.0 (2008) backward incompatibles
- Division, print, input, range, string formatting, module names, dictionary comprehensions
- Data structures and algorithms
- Strings and text
- Numbers, dates, and times
- Iterators and generators
- Files and I/O
- Data encoding and processing
- Functions
- Classes and Objects
- Modules and Packages

Libraries

- Graphical user interfaces
- Web frameworks
- Multimedia
- Databases
- Networking
- Test frameworks
- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

Reserved Words

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

```
>>> 23**2
529
>>> _+1
530
```

and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Start comments with # – the rest of line is ignored.

```
temp = eval(input('Enter a temperature in Celsius: '))
print('In Fahrenheit, that is', 9/5*temp+32)
```

```
num1 = eval(input('Enter the first number: '))
num2 = eval(input('Enter the second number: '))
print('The average of the numbers you entered is', (num1+num2)/2)
```

Reserved Words

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Indentation

- http://www.python.org
- Indentation
- Python is designed to be highly readable.

Case matters.
Spaces matter.
Indentation matters.

```
from random import randint

count = 0
for i in range(10000):
    num = randint(1, 100)
    if num%12==0:
        count=count+1
        print('Number of multiples of 12:', count)
```

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

4-blank

4-blank

print

```
print ('The value of 3+4 is', 3+4, '.')
print ('The value of 3+4 is ', 3+4, '.', sep='')
```

```
The value of 3+4 is 7 .
The value of 3+4 is 7.
```

```
print('On the first line')
print('On the second line')
```

```
On the first line
On the second line
```

```
print('On the first line', end='')
print('On the second line')
```

```
On the first lineOn the second line
```

Python 3

```
#!/usr/bin/python3

print ("Hello, Python!")
```

```
print "Hello World" #is acceptable in Python 2
print ("Hello World") # in Python 3, print must be followed by ()
```

As a result, 3/2 will show 1.

Python 3 evaluates 3 / 2 as 1.5 by default

For example, if we want Python 3.x's integer division behavior in Python 2, add the following import statement.

```
from __future__ import division
```

from __future__ import print_function

2to3.py

```
Here is a sample Python 2 code (area.py):
def area(x,y=3.14):
    a=y*x*x
    print a
    return a
```

```
a=area(10)
print "area",a
To convert into Python 3 version:
$2to3 -w area.py
Converted code :
def area(x,y=3.14): # formal parameters
    a=y*x*x
    print (a)
    return a
a=area(10)
print("area",a)
```

```
$ 2to3 example.py
$ 2to3 -w example.py
```

python C:₩Python34₩Tools₩Scripts₩2to3.py -w area.py

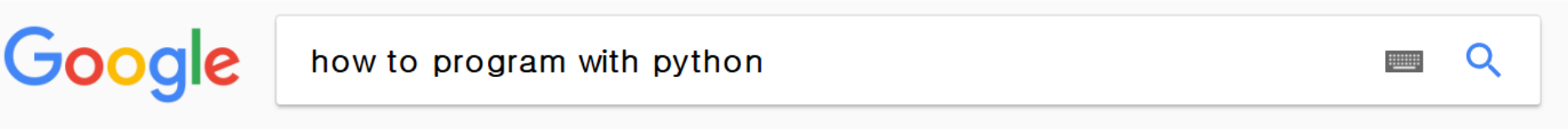
area.py.bak

https://docs.python.org/3/library/2to3.html

Python references

- Python 2.7
- Python Cookbook

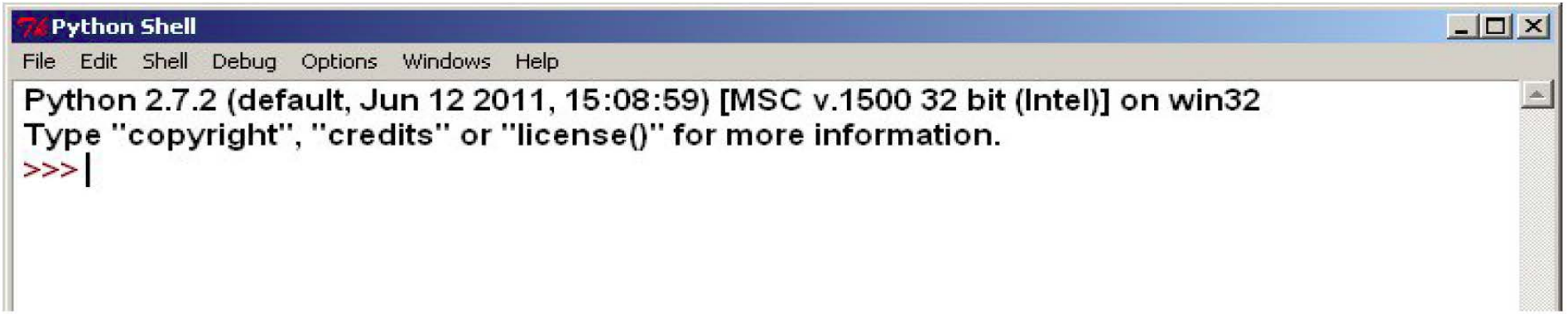
GitHub



Python (Windows, Linux)

- *Start* menu, select *All Programs* → *Python 2.7* → *IDLE(Python GUI)*

  IDLE is a simple integrated development environment.




```
$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Quicksort algorithm

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

Rosetta Code

Rosetta Code is a programming chrestomathy site. The idea is to present solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different, and to aid a person with a grounding in one approach to a problem in learning another.

Example-centric programming

http://rosettacode.org/wiki/Rosetta_Code

https://en.wikipedia.org/wiki/Example-centric_programming

Basic data types

Numbers, Booleans, Strings

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

```
>>> ((10 + 2) * 100 / 5 - 200)
40.0
```

```
>>> pow(2, 10)
1024
```

```
>>> eval( "2.5+2.5" )
5.0
```

Basic data types

Numbers, Booleans, Strings

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

Multi-Line Statements

```
>>> result = (10 + 100
... * 5 - 5
... / 100 + 10
... )
>>> print(result)
519.95
```

```
total = item_one + \
        item_two + \
        item_three
```

Explicit line continuation

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Implicit line continuation

Basic data types

Numbers, Booleans, Strings

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

String objects have a bunch of useful methods.

```
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with another;
                                # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

Basic data types

Numbers, Booleans, Strings

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

```
>>> test = 2 * 5 / 10
>>> print(test)
1.0
>>> type(test)
<class 'float'>
```

```
>>> test = 2 * 5
>>> print(test)
10
>>> type(test)
<class 'int'>
```

Containers

- **list**
  - a = [1, 3.0, 'abc']
- **dict**: named list
  - a = { 'name': 'John', 'age': 22, 'height': 179. }
- **set**
  - a= set([1,2,3,2,1,7])
- **tuple**: immutable list
  - a = (1,2,3,4)
  - Use tuple if possible (because it is *faster* than list).
  - A tuple can be a 'key' of a dict.
- **str**: string, immutable
  - a = 'abcdefg'

Lists

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

Slicing

```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0, 1]
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

List comprehensions

```python
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

Expression	Result
`[7,8]+[3,4,5]`	`[7,8,3,4,5]`
`[7,8]*3`	`[7,8,7,8,7,8]`
`[0]*5`	`[0,0,0,0,0]`

```python
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

```python
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

List comprehensions

```
>>> li = [1, 11, 3, 4, 5]

>>> li.append('a')   # Our first exposure to method syntax
>>> li
[1, 11, 3, 4, 5, 'a']

>>> li.insert(2, 'i')
>>>li
[1, 11, 'i', 3, 4, 5, 'a']

>>> li.extend([9, 8, 7])
>>>li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7]

>>> li.append([10, 11, 12])
>>> li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7, [10, 11, 12]]
```

Method	Description
`append(x)`	adds x to the end of the list
`sort()`	sorts the list
`count(x)`	returns the number of times x occurs in the list
`index(x)`	returns the location of the first occurrence of x
`reverse()`	reverses the list
`remove(x)`	removes first occurrence of x from the list
`pop(p)`	removes the item at index p and returns its value
`insert(p,x)`	inserts x at index p of the list

List comprehensions

```
>>> li = ['a', 'b', 'c', 'b']

>>> li.index('b')      # index of first occurrence
1

>>> li.count('b')      # number of occurrences
2

>>> li.remove('b')     # remove first occurrence
>>> li
  ['a', 'c', 'b']
```

Function	Description
`choice(L)`	picks a random item from `L`
`sample(L,n)`	picks a group of `n` random items from `L`
`shuffle(L)`	Shuffles the items of `L`

```
for i in range(len(L)):
    print(L[i])
```

```
for item in L:
    print(item)
```

List comprehensions

```
>>> li = [5, 2, 6, 8]

>>> li.reverse()     # reverse the list *in place*
>>> li
  [8, 6, 2, 5]

>>> li.sort()        # sort the list *in place*
>>> li
  [2, 5, 6, 8]

>>> li.sort(some_function)
    # sort in place using user-defined comparison
```

Function	Description
len	returns the number of items in the list
sum	returns the sum of the items in the list
min	returns the minimum of the items in the list
max	returns the maximum of the items in the list

```
average = sum(L)/len(L)
```

List comprehensions

```
string = 'Hello'
L = [1,14,5,9,12]
M = ['one', 'two', 'three', 'four', 'five', 'six']
```

List comprehension	Resulting list
`[0 for i in range(10)]`	`[0,0,0,0,0,0,0,0,0,0]`
`[i**2 for i in range(1,8)]`	`[1,4,9,16,25,36,49]`
`[i*10 for i in L]`	`[10,140,50,90,120]`
`[c*2 for c in string]`	`['HH','ee','ll','ll','oo']`
`[m[0] for m in M]`	`['o','t','t','f','f','s']`
`[i for i in L if i<10]`	`[1,5,9]`
`[m[0] for m in M if len(m)==3]`	`['o','t','s']`

```
L = [[i,j] for i in range(2) for j in range(2)]
```

```
[[0, 0], [0, 1], [1, 0], [1, 1]]
```

List comprehensions

```
L = [[0]*50 for i in range(100)]
```

This creates a list of zeroes with 100 rows and 50 columns.

```
L = [[[0]*5 for i in range(5)] for j in range(5)]
```

Here we create a 5 x 5 x 5 list.

```
random.seed(1)
print("Seed 1:", [random.randint(1,10) for i in range(5)])
random.seed(2)
print("Seed 2:", [random.randint(1,10) for i in range(5)])
random.seed(1)
print("Seed 1:",[random.randint(1,10) for i in range(5)])
```

```
Seed 1: [2, 9, 8, 3, 5]
Seed 2: [10, 10, 1, 1, 9]
Seed 1: [2, 9, 8, 3, 5]
```

String *vs* List

String methods do not change the original string, but list methods do change the original list. To sort a list L, just use L.sort() and not L=L.sort(). In fact, the latter will not work at all.

wrong	*right*
`s.replace('X','x')`	`s = s.replace('X','x')`
`L = L.sort()`	`L.sort()`

```
s = [‘H’,’e’,’l’,’l’,’o’]
s[0] = 'Y'
print s
```

```
['Y','e','l','l','o']
```

```
s = 'Hello'
s[0] = 'Y'

print s
```

```
s[0] = 'Y'
TypeError: 'str' object does not support item assignment
```

Dictionaries

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

```
#!/usr/bin/python

dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

Using dictionaries

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
>>> d['bozo']

Traceback (innermost last):
  File '<interactive input>' line 1, in ?
KeyError: bozo

>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user'] = 'clown'
>>> d
{'user':'clown', 'pswd':1234}

>>> d['id'] = 45
>>> d
{'user':'clown', 'id':45, 'pswd':1234}
```

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> del d['user']              # Remove one.
>>> d
{'p':1234, 'i':34}
>>> d.clear()                  # Remove all.
>>> d
{}

>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> d.keys()                   # List of keys.
['user', 'p', 'i']
>>> d.values()                 # List of values.
['bozo', 1234, 34]
>>> d.items()          # List of item tuples.
[('user','bozo'), ('p',1234), ('i',34)]
```

Using dictionaries

d={'A':1,'B':3}

Statement	Result	Description
`list(d)`	`['A','B']`	keys of d
`list(d.values())`	`[1,3]`	values of d
`list(d.items())`	`[('A',1),('B',3)]`	(key,value) pairs of d

Strings

```
s = 'Hello'
t = "Hello"
m = """This is a long string that is
spread across two lines."""
```

```
if 'a' in string:
    print('Your string contains the letter a.')
```

```
if ';' not in string:
    print('Your string does not contain any semicolons.')
```

Expression	Result
`'AB'+'cd'`	`'ABcd'`
`'A'+'7'+'B'`	`'A7B'`
`'Hi'*4`	`'HiHiHiHi'`

Strings

s='Python'

Statement	Result	Description
`s[0]`	`P`	first character of s
`s[1]`	`y`	second character of s
`s[-1]`	`n`	last character of s
`s[-2]`	`o`	second-to-last character of s

s='abcdefghij'

```
index:    0 1 2 3 4 5 6 7 8 9
letters:  a b c d e f g h i j
```

```
for i in range(len(s)):
    print (s[i])
```

Code	Result	Description
`s[2:5]`	`cde`	characters at indices 2, 3, 4
`s[ :5]`	`abcde`	first five characters
`s[5: ]`	`fghij`	characters from index 5 to the end
`s[-2:]`	`ij`	last two characters
`s[ : ]`	`abcdefghij`	entire string
`s[1:7:2]`	`bdf`	characters from index 1 to 6, by twos
`s[ : :-1]`	`jihgfedcba`	a negative step reverses the string

Strings

Method	Description
`lower()`	returns a string with every letter of the original in lowercase
`upper()`	returns a string with every letter of the original in uppercase
`replace(x,y)`	returns a string with every occurrence of `x` replaced by `y`
`count(x)`	counts the number of occurrences of `x` in the string
`index(x)`	returns the location of the first occurrence of `x`
`isalpha()`	returns **`True`** if every character of the string is a letter

Statement	Description
`print(s.count(' '))`	prints the number of spaces in the string
`s = s.upper()`	changes the string to all caps
`s = s.replace('Hi','Hello')`	replaces each `'Hi'` in `s` with `'Hello'`
`print(s.index('a'))`	prints location of the first `'a'` in `s`

Strings

parsing
delimiter

```
s = 'Hi! This is a test.'
print(s.split())
```

```
['Hi!', 'This', 'is', 'a', 'test.']
```

```
s = '1-800-271-8281'
print(s.split('-'))
```

```
['1', '800', '271', '8281']
```

```
L = ['A','B','C']
```

Operation	Result
`' '.join(L)`	A B C
`''.join(L)`	ABC
`', '.join(L)`	A, B, C
`'***'.join(L)`	A***B***C

.strip
.split
.join

Strings

Statement	Result
`str(37)`	`'37'`
`str(3.14)`	`'3.14'`
`str([1,2,3])`	`'[1,2,3]'`

Statement	Result
`int('37')`	37
`float('3.14')`	3.14
`int(3.14)`	3

.strip
.split
.join

Basic numeric operations

```
>>> 99 + 1
100
>>> 1 - 99
-98
>>> 7 * 5
35
>>> 81 / 9
9
```

```
>>> 1 / 5
0
```

```
>>> 1.0 / 4.0
0.25
>>> 1.0 / 3.0
0.3333333333333331
```

```
>>> 2 ** 50
1125899906842624L
>>> 2 ** 100
1267650600228229401496703205376L
>>> 2 ** 1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376L
```

long integer

Operator	Description
+	addition
–	subtraction
*	multiplication
/	division
**	exponentiation
//	integer division
%	modulo (remainder)

x ** *y*	*x* to the power *y*

factorial(531)

Numeric Types -- int, float, long, complex

range

```
>>> range(6)
[0, 1, 2, 3, 4, 5]
>>> range(2)
[0, 1]
>>> range(0)
[]
```

Statement	Values generated
`range(10)`	0,1,2,3,4,5,6,7,8,9
`range(1,10)`	1,2,3,4,5,6,7,8,9
`range(3,7)`	3,4,5,6
`range(2,15,3)`	2,5,8,11,14
`range(9,2,-1)`	9,8,7,6,5,4,3

```
>>> range(5,11)
[5, 6, 7, 8, 9, 10]
>>> range(1,5)
[1, 2, 3, 4]
```

```
>>> range ( 10, 100, 10 )
[10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> range ( 100, 0, -10 )
[100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
>>> range ( 8, -1, -1 )
[8, 7, 6, 5, 4, 3, 2, 1, 0]
```

File and execution

```
#!/usr/local/bin/python
print "Table of powers of two"
print
print "{0:>10s} {1:>2s} {2:s}".format("2**n", "n", "2**(-n)")
for n in range(13):
    print "{0:10d} {1:2d} {2:17.15f}".format(2**n, n, 2.0**(-n))
```

chmod +x *your-script-name*

```
$ ./powersof2
Table of powers of two

      2**n  n 2**(-n)
         1  0 1.000000000000000
         2  1 0.500000000000000
         4  2 0.250000000000000
         8  3 0.125000000000000
        16  4 0.062500000000000
        32  5 0.031250000000000
        64  6 0.015625000000000
       128  7 0.007812500000000
       256  8 0.003906250000000
       512  9 0.001953125000000
      1024 10 0.000976562500000
      2048 11 0.000488281250000
      4096 12 0.000244140625000
```

Line in file

```
with open(filename) as f:
    for line in f:
        print(line)
```

.strip

.split

.join

```
# Open the file with read only permit
f = open('my_text_file.txt')
# use readline() to read the first line
line = f.readline()
# use the read line to read further.
# If the file is not empty keep reading one line
# at a time, till the file is empty
while line:
    # in python 2+
    # print line
    # in python 3 print is a builtin function, so
    print(line)
    # use realine() to read next line
    line = f.readline()
f.close()
```

x = 'blue,red,green'

x.split(",")

['blue', 'red', 'green']

x.split(",")[0]

https://stackoverflow.com/questions/41903689/splitting-the-sentences-in-python

Read a line, text file

```
# file handle fh
fh = open('my_text_file.txt')
while True:
    # read line
    line = fh.readline()
    # in python 2, print line
    # in python 3
    print(line)
    # check if line is not empty
    if not line:
        break
fh.close()
```

```
>>> open('t.txt').read().count('\n')
```

```
>>> n = len(open('t.txt').read().split())
```

https://stackoverflow.com/questions/9282967/how-to-open-a-file-using-the-open-with-statement

Write a line, text file

```
f = open('writefile.txt', 'w')
print('This is line 1.', file=f)
print('This is line 2.', file=f)
f.close()
```

```
outF = open("myOutFile.txt", "w")
for line in textList:
  print >>outF, line
outF.close()
```

```
outF.write(line)
```

Current line and test

```
# open a file using with statement
with open(filename,'r') as fh
     for curline in fh:
         # check if the current line
         # starts with "#"
         if curline.startswith("#"):
            ...
            ...
         else:
            ...
            ...
```

```
f = open("/home/jeffery/wfile.txt", mode="w")

# Write string str to file.
str = "the content you want to write"
f.write(str)

f.close()
```

```
with open('my_file.txt') as fh:
    # Skip initial comments that starts with #
    while True:
        line = fh.readline()
        # break while statement if it is not a comment line
        # i.e. does not startwith #
        if not line.startswith('#'):
            break

    # Second while loop to process the rest of the file
    while line:
        print(line)
        ...
        ...
```

http://cmdlinetips.com/2018/01/3-ways-to-read-a-file-and-skip-initial-comments-in-python/

Gzip files

rb, wb → binary

```
1 import gzip
```

How to read a gzip file line by line in Python?

```
1 with gzip.open('big_file.txt.gz', 'rb') as f:
2     for line in f:
3         print(line)
```

```
1 all_of_of_your_content = "all the content of a big text file"
2 with gzip.open('file.txt.gz', 'wb') as f:
3     f.write(all_of_your_content)
```

```
1 import shutil
2 # open the unzipped file with flie handler inp_f
3 with open("test_file.txt","rb") as inp_f:
4     # open the output zipped file with file handler out_f
5     with gzip.open("test_file.txt.gz","wb") as out_f:
6         # use shutil to copy the file objec
7         shutil.copyfileobj(inp_f,out_f)
```

http://cmdlinetips.com/2018/02/how-to-read-a-gzip-file-in-python/

functions

```
>>> def pythag(a, b):
...     """Returns the hypotenuse of a right triangle with sides a and b.
...     """
...     return (a*a + b*b)**0.5
...
>>> pythag(3,4)
5.0
>>> pythag(1,1)
1.4142135623730951
>>> print pythag.__doc__
Returns the hypotenuse of a right triangle with sides a and b.

>>>
```

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

Optional arguments

```
def func(a, b, c=10, d=100):
  print a, b, c, d

>>> func(1,2)
1 2 10 100

>>> func(1,2,3,4)
1,2,3,4
```

Default values

Built-in functions
dir(_builtins_)

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'Bytes Warning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'Depre cationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingP ointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'Interrup tedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryEr ror',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessL ookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEn codeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'Zero DivisionError',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii',
'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir',
'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float',
'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help',
```

Math functions

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'g amma', 'hypot',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>>
```

seed(10)

endpoints included

```
from random import randint
x = randint(1,10)
print('A random number between 1 and 10: ', x)
```

```
from math import sin, pi
print('Pi is roughly', pi)
print('sin(0) =', sin(0))
```

Math functions/Random numbers

```
from math import sin, pi
print('Pi is roughly', pi)
print('sin(0) =', sin(0))
```

seed(10)

endpoints included

```
from random import randint
x = randint(1,10)
print('A random number between 1 and 10: ', x)
```

```
from math import sin, pi
print('Pi is roughly', pi)
print('sin(0) =', sin(0))
```

How to use lambda functions in Python?

```
1  def my_add(a,b):
2        return(a+b)
```

```
1  lambda a, b: a+b
```

```
1  >f = lambda a, b: a+b
2  >f(1,2)
3  3
```

```
1  >my_list = range(20)

   >map(lambda x: x*2, my_list)
```

```
1  filter(lambda x: x >10, my_list)
2  [11, 12, 13, 14, 15, 16, 17, 18, 19]
```

http://cmdlinetips.com/2018/01/getting-started-with-lambda-functions-in-python/

assert

```
# recursive way of writing factorial
def fact( n ):
    if ( isinstance( n, complex) ):
        raise Exception(“Cannot calculate factorial of complex function”)
    if ( n == 0 ):
        return 1.0
    elif ( n < 0 ):
        raise Exception("Cannot calculate factorial for negative numbers")
    assert( n > 0 ) #since all other cases are treated elsewhere
    return fact( n - 1)*n;

if __name__ == "__main__":
    for i in range(0,10+1):
        print("%d! = %g"%(i,fact(i)))
    print("%d! = %g"%(-5+4j,fact( -5 + 4j ))) #raises exception for complex
    print("%d! = %g"%(-5,fact( -5 ))) #raises exception
```

AssertionError:

Classes

- **A software item that contains variables and methods**
- **Object Oriented Design focuses on**

• Encapsulation:
—dividing the code into a public interface, and a private implementation of that interface
• Polymorphism:
—the ability to overload standard operators so that they have appropriate behavior based on their context
• Inheritance:
—the ability to create subclasses that contain specializations of their parents

Class	Instance	Attribute	Operation
Airplane	Wright Flyer	Wingspan	Take off
Mountain	Socorro Peak	Altitude	Erupt
Clock	Skeen Library Clock	Amount slow per day	Decorate

Classes

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

__ → private

Special method names

+	__add__(self, other)
*	__mul__(self, other)
-	__sub__(self, other)
/	__truediv__(self, other)
%	__mod__(self, other)
<	__lt__(self, other)
<=	__le__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)
idex]	__getitem__(self, index)
in	__contains__(self, value)
len	__len__(self)
str	__str__(self)

Method: Either class or class instance can call.

Classes

```python
#An example of a class
class Shape:

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y

    def perimeter(self):
        return 2 * self.x + 2 * self.y

    def describe(self, text):
        self.description = text

    def authorName(self, text):
        self.author = text

    def scaleSize(self, scale):
        self.x = self.x * scale
        self.y = self.y * scale
```

constructor

methods

Classes

```
rectangle = Shape(100, 45)
```

```
#finding the area of your rectangle:
print(rectangle.area())

#finding the perimeter of your rectangle:
print(rectangle.perimeter())

#describing the rectangle
rectangle.describe("A wide rectangle, more than twice\
 as wide as it is tall")

#making the rectangle 50% smaller
rectangle.scaleSize(0.5)

#re-printing the new area of the rectangle
print(rectangle.area())
```

Inheritance

```python
#An example of a class
class Shape:

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y

    def perimeter(self):
        return 2 * self.x + 2 * self.y

    def describe(self, text):
        self.description = text

    def authorName(self, text):
        self.author = text

    def scaleSize(self, scale):
        self.x = self.x * scale
        self.y = self.y * scale
```

```python
class Square(Shape):
    def __init__(self,x):
        self.x = x
        self.y = x
```

```python
class DoubleSquare(Square):
    def __init__(self,y):
        self.x = 2 * y
        self.y = y
    def perimeter(self):
        return 2 * self.x + 3 * self.y
```

Classes

```
class atom(object):
  def __init__(self,atno,x,y,z):
       self.atno = atno
       self.position = (x,y,z)
  def symbol(self):    # a class method
       return Atno_to_Symbol[atno]
  def __repr__(self): # overloads printing
       return '%d %10.4f %10.4f %10.4f' %
              (self.atno, self.position[0],
               self.position[1],self.position[2])

>>> at = atom(6,0.0,1.0,2.0)
>>> print at
6  0.0000  1.0000 2.0000
>>> at.symbol()
'C'
```

__ → private

→ string

Classes

```
class molecule:
  def __init__(self,name='Generic'):
       self.name = name
       self.atomlist = []
  def addatom(self,atom):
       self.atomlist.append(atom)
  def __repr__(self):
       str = 'This is a molecule named %s\n' % self.name
       str = str+'It has %d atoms\n' % len(self.atomlist)
       for atom in self.atomlist:
              str = str + `atom` + '\n'
       return str
```

__ → private

→ string

```
>>> mol = molecule('Water')
>>> at = atom(8,0.,0.,0.)
>>> mol.addatom(at)
>>> mol.addatom(atom(1,0.,0.,1.))
>>> mol.addatom(atom(1,0.,1.,0.))
>>> print mol
This is a molecule named Water
It has 3 atoms
8   0.000 0.000 0.000
1   0.000 0.000 1.000
1   0.000 1.000 0.000
```

Classes

```
class qm_molecule(molecule):
   def addbasis(self):
      self.basis = []
      for atom in self.atomlist:
         self.basis = add_bf(atom,self.basis)
```

- **__init__, __repr__, and __addatom__ are taken from the parent class (molecule)**
- **Added a new function addbasis() to add a basis set**
- **Another example of code reuse**
  - Basic functions don't have to be retyped, just inherited
  - Less to rewrite when specifications change

https://www.youtube.com/watch?v=BJ-VvGyQxho

Instance variables and class variables

Modules

A module is a file containing Python definitions and statements.
The file name is the module name with the suffix .py appended.
A module can be imported into other modules.

Function	Description
`sin, cos, tan`	trig functions
`asin, acos, atan`	inverse trig functions
`atan2(y, x)`	gives arctan(y/x) with proper sign behavior
`sinh, cosh, tanh`	hyperbolic functions
`asinh, acosh, atanh`	inverse hyperbolic functions
`log, log10`	natural log, log base 10
`log1p`	`log(1+x)`, more accurate near 1 than `log`
`exp`	exponential function e^x
`degrees, radians`	convert from radians to degrees or vice-versa
`floor`	`floor(x)` is the greatest integer $\leq$ `x`
`ceil`	`ceil(x)` is the least integer $\geq$ `x`
`e, pi`	the constants e and π
`factorial`	factorial
`modf`	returns a pair (fractional part, integer part)
`gamma, erf`	the Γ function and the Error function

Modules

- **Modules are functions and variables defined in separate files**
- **Items are imported using from or import**

```
from module import function
function()
import module
module.function()
```

- **Modules are namespaces**
  - Can be used to organize variable names, i.e.

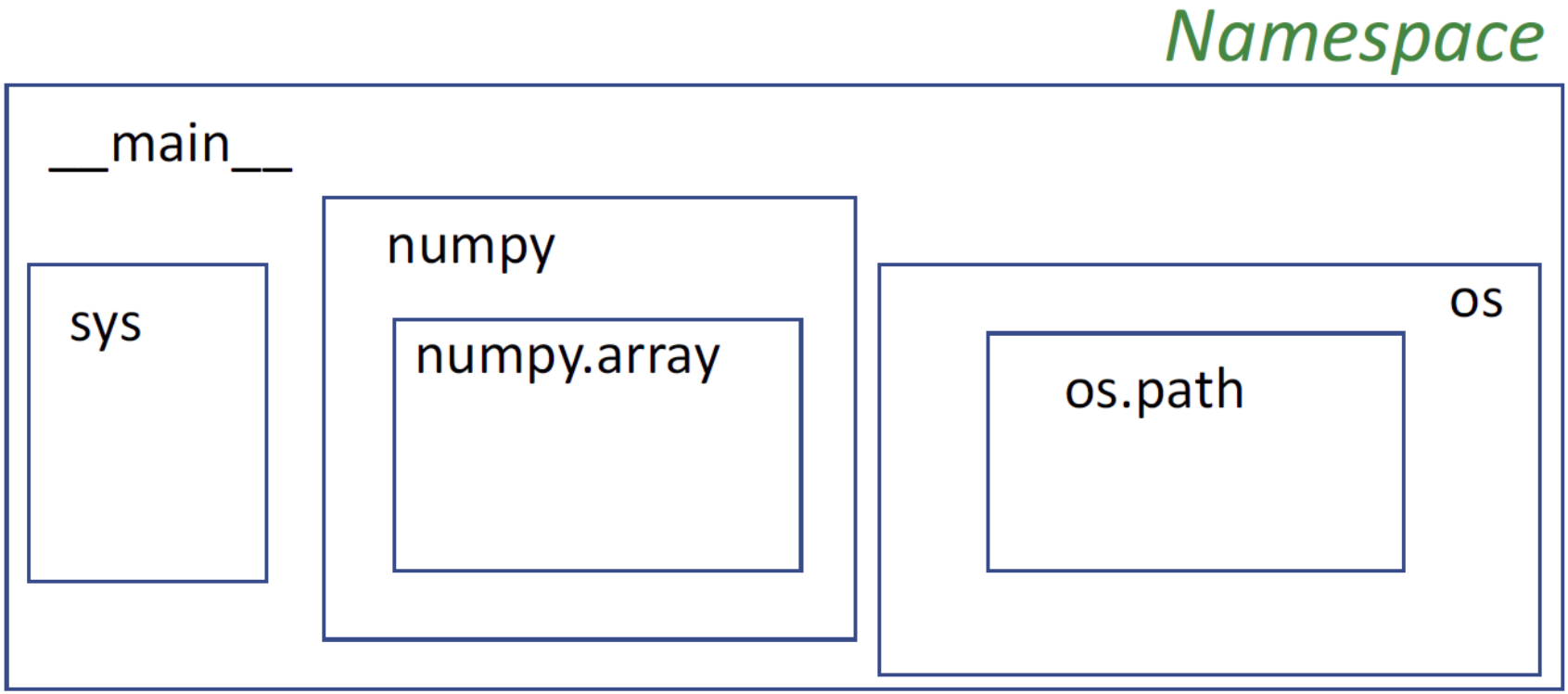
```
atom.position = atom.position - molecule.position
```

A module is a container for a unique set of names.

Python namespace

- sys.stdout (module.object)
- numpy.array.mean (module.class.object)
- os.path.basename (module.module.object)
- ‘a b c’.split (object.object)

Module.object
Class.object



import

```
>>> from math import *
>>> dir()
['__builtins__', '__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2',
'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod',
'frankness', 'frexp', 'hypot', 'ldexp', 'log', 'log10', 'modf', 'oi', 'pi',
 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
>>> sqrt(64)
8.0
```

```
>>> pi*10.0
31.415926535897931
>>> cos(0.0)
1.0
>>>
```

1. The present directory
2. $PYTHONPATH
3. sys.path

```
>>> import some_module
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named some_module
```

input

```
#  File tempconv.py
#  Author: Rick Halterman
#  Last modified: August 22, 2014
#  Converts degrees Fahrenheit to degrees Celsius
#  Based on the formula found at
#  http://en.wikipedia.org/wiki/Conversion_of_units_of_temperature

#  Prompt user for temperature to convert and read the supplied value
degreesF = float(input('Enter the temperature in degrees F: '))
#  Perform the conversion
degreesC = 5/9*(degreesF - 32)
#  Report the result
print(degreesF, 'degrees F =', degreesC, 'degrees C')
```

```
Enter the temperature in degrees F: 212
212 degrees F = 100.0 degrees C
```

```
Enter the temperature in degrees F: 32
32 degrees F = 0.0 degrees C
```

Control of flow

```
if x == 3:
    print "X equals 3."
elif x == 2:
    print "X equals 2."
else:
    print "X equals something else."
print "This is outside the 'if'."
```

```
assert(number_of_players < 5)
```

```
x = 3
while x < 10:
    if x > 7:
        x += 2
        continue
    x = x + 1
    print "Still in the loop."
    if x == 8:
        break
print "Outside of the loop."
```

```
for x in range(10):
    if x > 7:
        x += 2
        continue
    x = x + 1
    print "Still in the loop."
    if x == 8:
        break
print "Outside of the loop."
```

if

```
#  Get two integers from the user
dividend = int(input('Please enter the number to divide: '))
divisor = int(input('Please enter dividend: '))
#  If possible, divide them and report the result
if divisor != 0:
    print(dividend, '/', divisor, "=", dividend/divisor)
else:
    print('Division by zero is not allowed')
```

```
Please enter the number to divide: 32
Please enter dividend: 0
Division by zero is not allowed
```

```
value = int(input("Please enter an integer value in the range 0...10: ")
if value >= 0:          # First check
    if value <= 10:     # Second check
        print("In range")
print("Done")
```

Incorrect	Correct
`if x=1:`	`if x==1:`

Expression	Description
`if x>3:`	if x is greater than 3
`if x>=3:`	if x is greater than or equal to 3
`if x==3:`	if x is 3
`if x!=3:`	if x is not 3

while

```
#  Counts up from zero.  The user continues the count by entering
#  'Y'.  The user discontinues the count by entering 'N'.

count = 0      # The current count
entry = 'Y'    # Count to begin with

while entry != 'N' and entry != 'n':
    # Print the current value of count
    print(count)
    entry = input('Please enter "Y" to continue or "N" to quit: ')
    if entry == 'Y' or entry == 'y':
        count += 1    # Keep counting
    # Check for "bad" entry
    elif entry != 'N' and entry != 'n':
        print('"' + entry + '" is not a valid choice')
    # else must be 'N' or 'n'
```

```
0
Please enter "Y" to continue or "N" to quit: y
1
Please enter "Y" to continue or "N" to quit: y
2
Please enter "Y" to continue or "N" to quit: y
3
Please enter "Y" to continue or "N" to quit: q
"q" is not a valid choice
3
Please enter "Y" to continue or "N" to quit: r
"r" is not a valid choice
3
Please enter "Y" to continue or "N" to quit: W
"W" is not a valid choice
3
Please enter "Y" to continue or "N" to quit: Y
4
Please enter "Y" to continue or "N" to quit: y
5
Please enter "Y" to continue or "N" to quit: n
```

for

```
word = input('Enter text: ')
vowel_count = 0
for c in word:
    if c == 'A' or c == 'a' or c == 'E' or c == 'e' \
       or c == 'I' or c == 'i' or c == 'O' or c == 'o':
        print(c, ', ', sep='', end='')  # Print the vowel
        vowel_count += 1                # Count the vowel
print(' (', vowel_count, ' vowels)', sep='')
```

```
Enter text: Mary had a little lamb.
a, a, a, i, e, a,  (6 vowels)
```

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

for

```
#  Get the number of rows and columns in the table
size = int(input("Please enter the table size: "))
#  Print a size x size multiplication table
for row in range(1, size + 1):
    for column in range(1, size + 1):
        product = row*column                      # Compute product
        print('{0:4}'.format(product), end='')    # Display product
    print()                                       # Move cursor to next row
```

```
Please enter the table size: 10
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

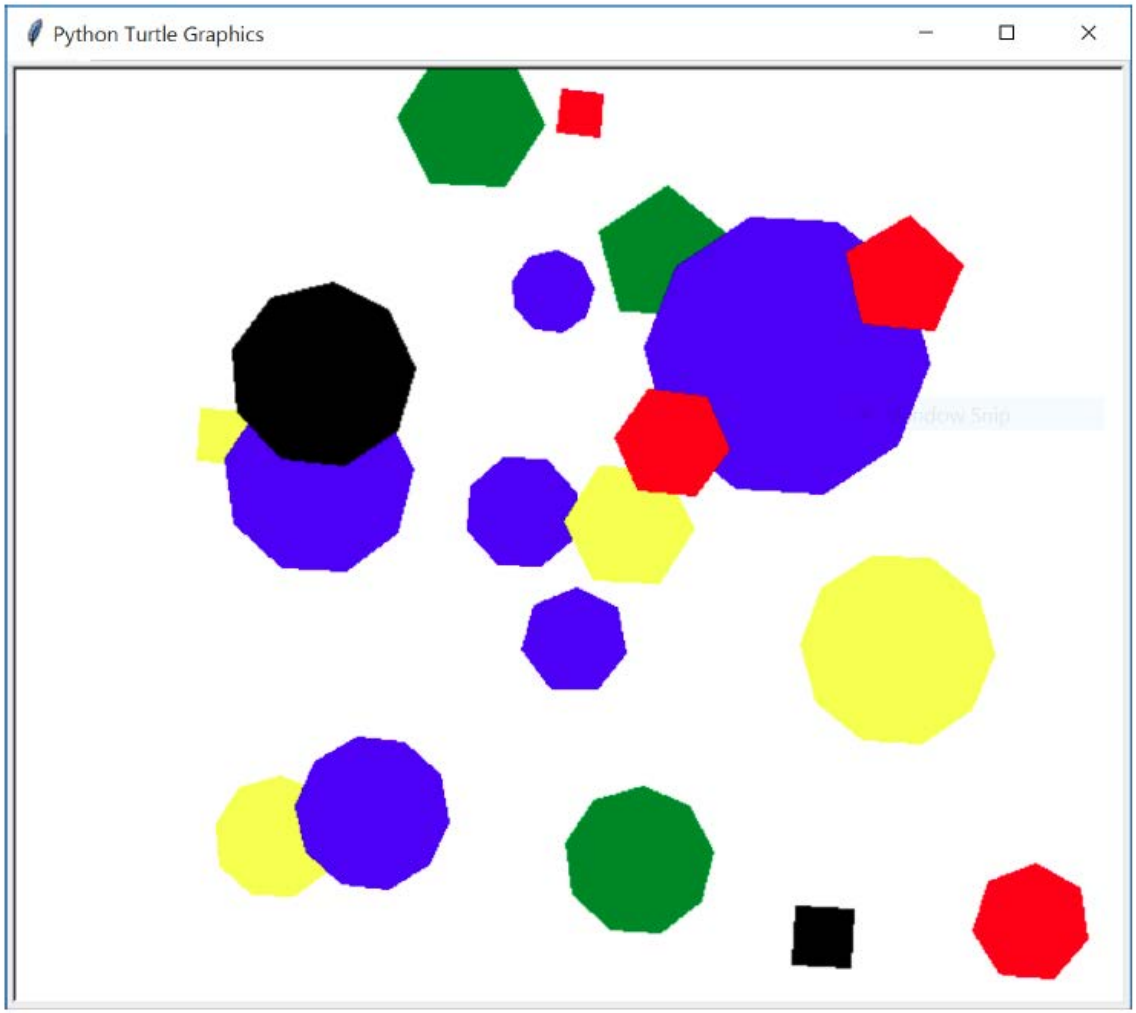
Prime numbers

```
#!/usr/bin/python

i = 2
while(i < 100):
    j = 2
    while(j <= (i/j)):
        if not(i%j): break
        j = j + 1
    if (j > i/j) : print i, " is prime"
    i = i + 1

print "Good bye!"
```

turtle

```
import turtle
import random


# Draws a regular polygon with the given number of sides.
# The length of each side is length.
# The pen begins at point(x, y).
# The color of the polygon is color.
def polygon(sides, length, x, y, color):
    turtle.penup()
    turtle.setposition(x, y)
    turtle.pendown()
    turtle.color(color)
    turtle.begin_fill()
    for i in range(sides):
        turtle.forward(length)
        turtle.left(360//sides)
    turtle.end_fill()


# Disable rendering to speed up drawing
turtle.hideturtle()
turtle.tracer(0)

# Draw 20 random polygons with 3-11 sides, each side ranging
# in length from 10-50, located at random position (x, y).
# Select a color at random from red, green, blue, black, or yellow.
for i in range(20):
    polygon(random.randrange(3, 11), random.randrange(10, 51),
            random.randrange(-250, 251), random.randrange(-250, 251),
            random.choice(("red", "green", "blue", "black", "yellow")))

turtle.update()  # Render image
turtle.exitonclick()  # Wait for user's mouse click
```

turtle



System call

```
import os

cmd = "git --version"

returned_value = os.system(cmd)  # returns the exit code in unix
print('returned value:', returned_value)
```

```
git version 2.14.2
returned value: 0
```

```
import sys

os.system ('ls -al | grep "user")
```

https://docs.python.org/2/library/subprocess.html

Plotnine: A Python library to use ggplot2 in Python

If you are coming from R background and know ggplot2, you might want to still use ggplot2 in Python for making great visualizations.

Altair is kind of the new kid in the data visualization block.

https://media.readthedocs.org/pdf/plotnine/latest/plotnine.pdf

http://cmdlinetips.com/2018/04/an-introduction-to-altair-a-python-visualization-library/

http://cmdlinetips.com/2018/05/plotnine-a-python-library-to-use-ggplot2-in-python/

matplotlib

- High quality plotting library.
- Downloads: http://matplotlib.sourceforge.net/
- https://matplotlib.org/

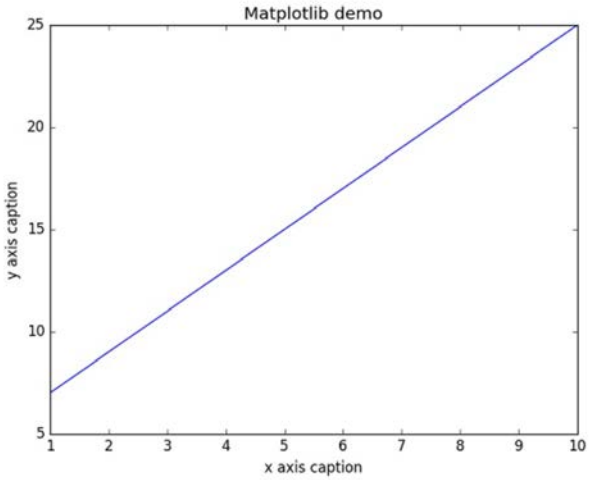

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
line1, = ax.plot(x1,y1,options)
line2, = ax.plot(x2,y2,options)
plt.show()
```

```
import Gnuplot
p = Gnuplot.Gnuplot()
p('set term X11')
p('plot "textfile" u 1:2 w lines')
```

http://www.gnuplot.info

matplotlib

```
import numpy as np
from matplotlib import pyplot as plt

x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y)
plt.show()
```



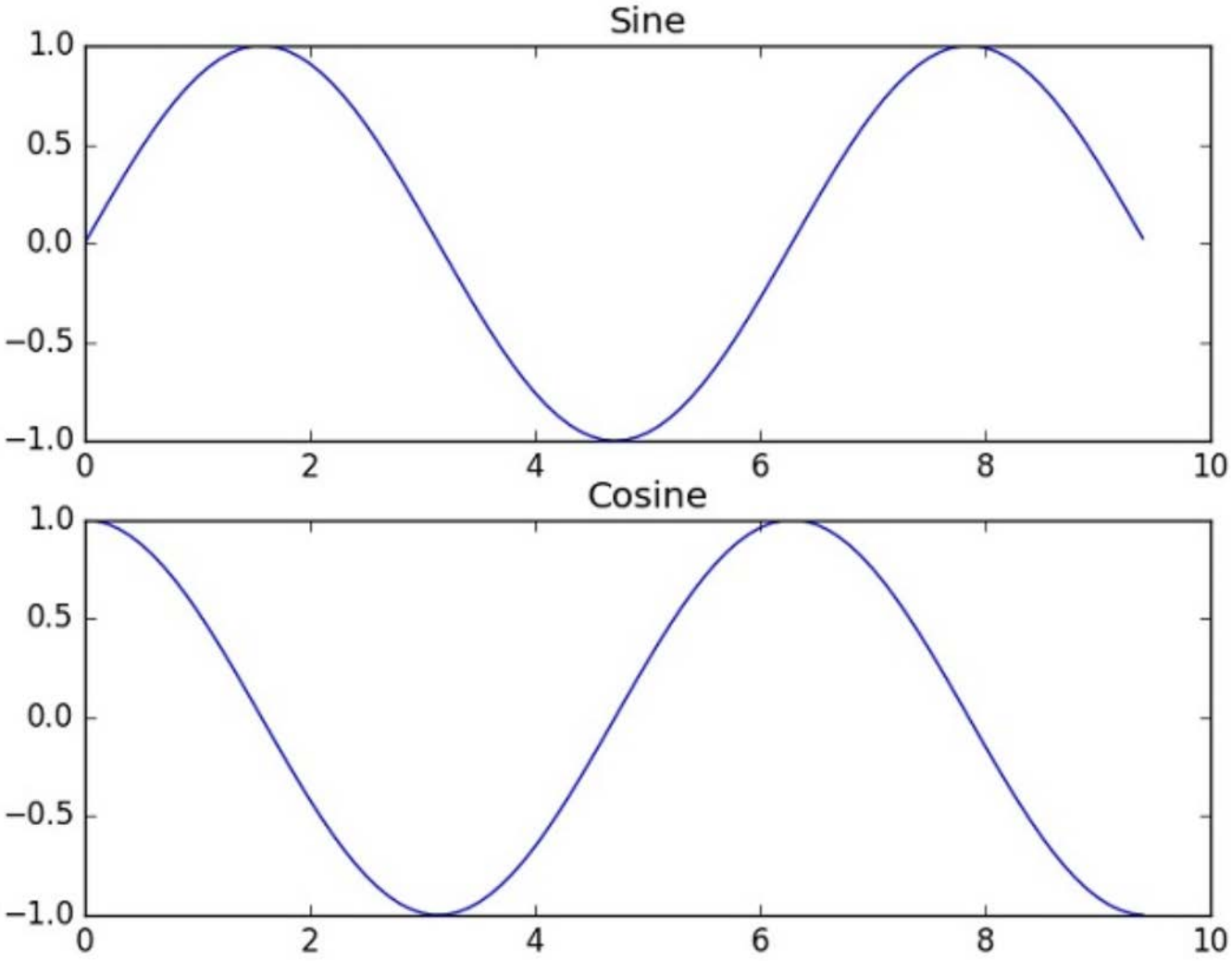
matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```

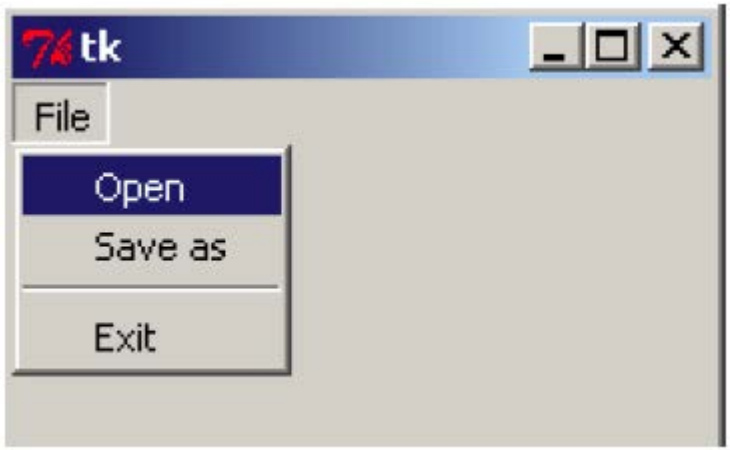
matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

matplotlib



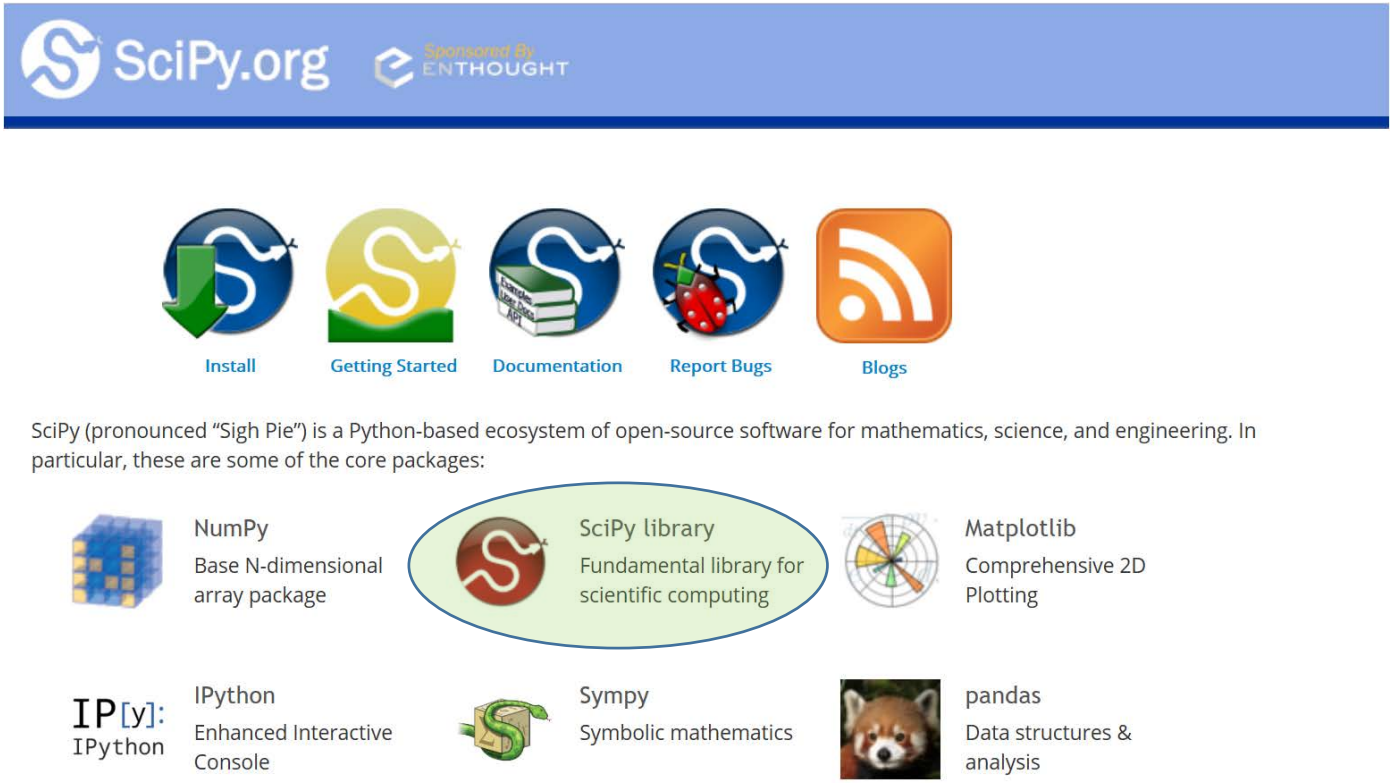
GUI Programming with Tkinter



Python 2	Python 3
Tkinter	tkinter
ScrolledText	tkinter.scrolledtext
tkMessageBox	tkinter.messagebox
tkFileDialog	tkinter.filedialog

SciPy

- Scientific Computing Tools for Python
- https://www.scipy.org/



Pandas

providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language

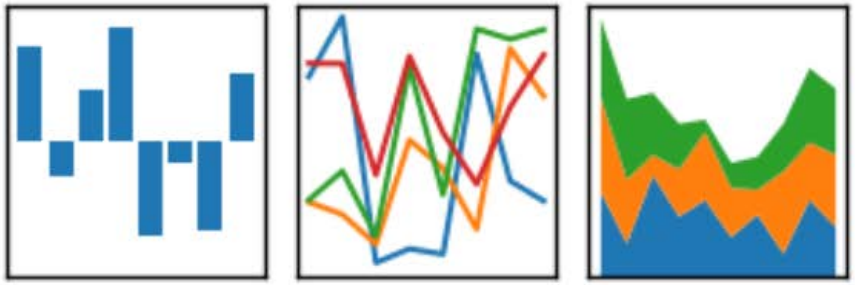
Data Structures

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

Comma-separated values

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



http://pandas.pydata.org/pandas-docs/stable/index.html

SciPy

```
from scipy.misc import imread, imsave, imresize

# Read an JPEG image into a numpy array
img = imread('assets/cat.jpg')
print(img.dtype, img.shape)  # Prints "uint8 (400, 248, 3)"

# We can tint the image by scaling each of the color channels
# by a different scalar constant. The image has shape (400, 248, 3);
# we multiply it by the array [1, 0.95, 0.9] of shape (3,);
# numpy broadcasting means that this leaves the red channel unchanged,
# and multiplies the green and blue channels by 0.95 and 0.9
# respectively.
img_tinted = img * [1, 0.95, 0.9]

# Resize the tinted image to be 300 by 300 pixels.
img_tinted = imresize(img_tinted, (300, 300))

# Write the tinted image back to disk
imsave('assets/cat_tinted.jpg', img_tinted)
```

SciPy

```
>>> import scipy.integrate as integrate
>>> import scipy.special as special
>>> result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)
>>> result
(1.1178179380783249, 7.8663172481899801e-09)
```

https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

SciPy

```
>>> from scipy.integrate import odeint
>>> from scipy.special import gamma, airy
>>> y1_0 = 1.0 / 3**(2.0/3.0) / gamma(2.0/3.0)
>>> y0_0 = -1.0 / 3**(1.0/3.0) / gamma(1.0/3.0)
>>> y0 = [y0_0, y1_0]
>>> def func(y, t):
...     return [t*y[1],y[0]]
...
```

- Special functions (**`scipy.special`**)
- Integration (**`scipy.integrate`**)
- Optimization (**`scipy.optimize`**)
- Interpolation (**`scipy.interpolate`**)
- Fourier Transforms (**`scipy.fftpack`**)
- 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`**)

https://docs.scipy.org/doc/scipy/reference/tutorial/index.html
https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

PyFITS

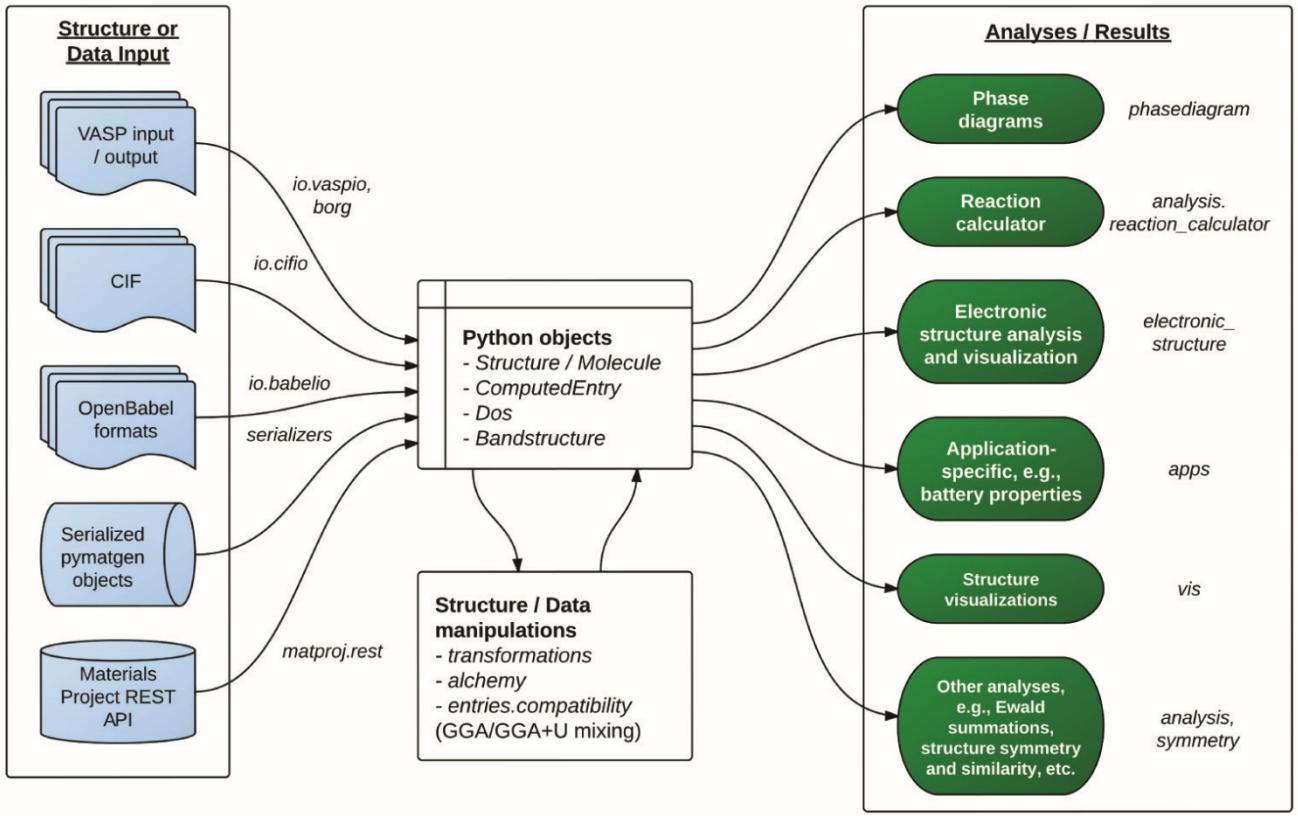
- Downloads: http://www.stsci.edu/resources/software_hardware/pyfits

FITS (Flexible Image Transport System)

https://pythonhosted.org/pyfits/

pymatgen

Pymatgen (Python Materials Genomics)



pymatgen

http://pymatgen.org/

NumPy

- The fundamental module for scientific computing
  - Many scientific modules are dependent on the numpy module.
- An efficient data container
  - *N*-dimensional matrix
  - Easy to communicate with C/C++ and Fortran
- Fast looping
  - Loop calculation is as fast as C/Fortran
- Many built-in functions for data handling and linear algebra

NumPy

- Offers Matlab-ish capabilities within Python
- Fast array operations
- 2D arrays, multi-D arrays, linear algebra etc.
- Downloads: http://numpy.scipy.org/
- Tutorial: http://www.scipy.org/Tentative_NumPy_Tutorial

https://www.youtube.com/watch?v=WmXiPyEcChg&list=PLuHVbKL2C7TDPDoBginJhSC48UM3We-3p

https://docs.scipy.org/doc/numpy/user/quickstart.html

NumPy data types

- Integer (np.int64)
- Float (np.float64) ← DEFAULT
- Complex (np.complex128)

Popular ways to create numpy data

- **np.array**([1.0, 2.0, 3.0])
- **np.empty**((2,4),dtype=np.int64)
- **np.zeros**(10,dtype=np.uint8)
- **np.ones**((3,3,3),dtype=np.float64)

NumPy

```
import numpy as np
a = np.array([1,2,3])
print a
```

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
print a
```

```
[[1, 2]
 [3, 4]]
```

```
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print a
```

```
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

NumPy

```
import numpy as np

a = np.array([[1,2,3],[4,5,6]])
a.shape = (3,2)
print a
```

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

```
import numpy as np
a = np.arange(24)
print a
```

```
[0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 17 18 19 20 21 22 23]
```

NumPy

```
import numpy as np
a = np.arange(24)
a.ndim

# now reshape it
b = a.reshape(2,4,3)
print b
# b is having three dimensions
```

```
[[[ 0,  1,  2]
  [ 3,  4,  5]
  [ 6,  7,  8]
  [ 9, 10, 11]]
  [[12, 13, 14]
   [15, 16, 17]
   [18, 19, 20]
   [21, 22, 23]]]
```

```
>>> a1=np.array([1,2,3])
>>> a2=np.array([4,5,6])
>>> a1+a2
array([5, 7, 9])
>>> a2-a1
array([3, 3, 3])
>>> a1*a2
array([ 4, 10, 18])
>>> a1/a2
array([ 0.25,  0.4 ,  0.5 ])
>>>
```

NumPy

```
# convert list to ndarray
import numpy as np

x = [1,2,3]
a = np.asarray(x)
print a
```

```
a[0] = 5
print a[0], a[1], a[2]
print a.shape
print type(a)
type 'numpy.ndarray
```

```
# dtype is set
import numpy as np

x = [1,2,3]
a = np.asarray(x, dtype = float)
print a
```

```
a = np.zeros((2,2))
print(a)

[[ 0.  0.]
 [ 0.  0.]]
```

```
a = np.ones((2,3))
print(a)

[[ 1.  1.  1.]
 [ 1.  1.  1.]]
```

NumPy

```
# create list object using range function
import numpy as np
list = range(5)
print list
```

```
# obtain iterator object from list
import numpy as np
list = range(5)
it = iter(list)

# use iterator to create ndarray
x = np.fromiter(it, dtype = float)
print x
```

```
import numpy as np
# dtype set
x = np.arange(5, dtype = float)
print x
```

```
>>> np.matmul(a, b)
array([16, 6, 8])
```

```
>>> a = np.array([[ 5, 1 ,3],
                  [ 1, 1 ,1],
                  [ 1, 2 ,1]])
>>> b = np.array([1, 2, 3])
>>> print a.dot(b)
array([16, 6, 8])
```

NumPy

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print c
```

```
[10   40   90   160]
```

a[0]

a[1]

less memory

cf. list *vs* numpy array

```
import numpy as np
import time
import sys

l = range(1000)
print(sys.getsizeof(5)*len(l))

array = np.arange(1000)
print(array.size*array.itemsize)
```

```
14000
4000
```

NumPy

```
import numpy as np
import time
import sys

SIZE = 1000000

l1 = range(SIZE)
l2 = range(SIZE)

a1=np.arange(SIZE)
a2=np.arange(SIZE)
```

```
# python list
start = time.time()
result = [(x+y) for x,y in zip(l1,l2)]
print("python list took: ",(time.time()-start)*1000)
# numpy array
start= time.time()
result = a1 + a2
print("numpy took: ", (time.time()-start)*1000)
```

```
python list took:  116.33777618408203
numpy took:  11.529922485351562
```

a[0]

a[1]

fast

cf. list *vs* numpy array

NumPy

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

NumPy

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print 'Original array is:'
print a
print '\n'

print 'Sorted in C-style order:'
for x in np.nditer(a, order = 'C'):
   print x,
print '\n'

print 'Sorted in F-style order:'
for x in np.nditer(a, order = 'F'):
   print x,
```

```
>>> a = np.arange(6).reshape(2,3)
>>> for x in np.nditer(a.T):
...     print x,
...
0 1 2 3 4 5
```

```
>>> for x in np.nditer(a.T.copy(order='C')):
...     print x,
...
0 3 1 4 2 5
```

https://docs.scipy.org/doc/numpy-1.14.0/reference/arrays.nditer.html#arrays-nditer

NumPy

```
>>> from numpy.random import rand
>>> from numpy.linalg import solve, inv
>>> a = np.array([[1, 2, 3], [3, 4, 6.7], [5, 9.0, 5]])
>>> a.transpose()
array([[ 1. ,  3. ,  5. ],
       [ 2. ,  4. ,  9. ],
       [ 3. ,  6.7,  5. ]])
>>> inv(a)
array([[-2.27683616,  0.96045198,  0.07909605],
       [ 1.04519774, -0.56497175,  0.1299435 ],
       [ 0.39548023,  0.05649718, -0.11299435]])
>>> b =  np.array([3, 2, 1])
>>> solve(a, b)  # solve the equation ax = b
array([-4.83050847,  2.13559322,  1.18644068])
>>> c = rand(3, 3) * 20  # create a 3x3 random matrix of values within [0,1] scaled by 20
>>> c
array([[  3.98732789,   2.47702609,   4.71167924],
       [  9.24410671,   5.5240412 ,  10.6468792 ],
       [ 10.38136661,   8.44968437,  15.17639591]])
>>> np.dot(a, c)  # matrix multiplication
array([[  53.61964114,   38.8741616 ,   71.53462537],
       [ 118.4935668 ,   86.14012835,  158.40440712],
       [ 155.04043289,  104.3499231 ,  195.26228855]])
>>> a @ c # Starting with Python 3.5 and NumPy 1.10
array([[  53.61964114,   38.8741616 ,   71.53462537],
       [ 118.4935668 ,   86.14012835,  158.40440712],
```

NumPy

```
>>> vector1 = array((1,2,4,5))
>>> print vector1
[1 2 3 4 5]
>>> matrix1 = array(([0,1],[1,3]))
>>> print matrix1
[[0 1]
 [1 3]]
>>> print vector1.shape, matrix1.shape
(5,) (2,2)
>>> print vector1 + vector1
[ 2  4  6  8  10]]
>>> print matrix1 * matrix1
[[0 1]                          # note that this is not the matrix
 [1 9]]                         # multiplication of linear algebra
```

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.array.html

NumPy

```
>>> print sin([pi/2., pi/4., pi/6.])
[ 1.         ,  0.70710678,  0.5        ]
>>> print greater([1,2,4,5], [5,4,3,2])
[0 0 1 1]
>>> print add([1,2,4,5], [5,4,3,2])
[6 6 7 7]
>>> print add.reduce([1,2,4,5])
12                                  # 1 + 2 +     4 + 5
```

```
>>> import numpy as np
>>> a=np.array([5,6,9])
>>> a[0]
5
>>> a[1]
6
>>> a=np.array([[1,2],[3,4],[5,6]])
>>> a.ndim
2
```

NumPy

```
>>> data = arange(10)                    # convenient homolog of builtin
range()
>>> print data
[0 1 2 3 4 5 6 7 8 9]
>>> print where(greater(data, 5), -1, data)
[ 0  1  2  3  4  5 -1 -1 -1 -1]  # selection facility
>>> data = resize(array((0,1)), (9, 9))
>>> print data
[[0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]]
```

```
>>> a=np.array([[0,1],[2,3]])
>>> np.resize(a,(2,3))
array([[0, 1, 2],
       [3, 0, 1]])
>>> np.resize(a,(1,4))
array([[0, 1, 2, 3]])
>>> np.resize(a,(2,4))
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.resize.html

NumPy

```
>>> from RandomArray import random, uniform, randint, permutation
>>> print random((5,5))
[[ 0.45456091  0.53438765  0.72412336  0.12156525  0.79255972]
 [ 0.14763653  0.93401444  0.38913983  0.97293309  0.45860398]
 [ 0.57528652  0.9801351   0.19893601  0.3396503   0.12224415]
 [ 0.9067847   0.37667559  0.71613152  0.24334284  0.68907028]
 [ 0.9655151   0.29746972  0.42734603  0.72314573  0.66344323]]
>>> print uniform(-1.0,1.0, (5,))
[-0.2637264   0.12331069  0.11497829 -0.25969645  0.36571342]
>>> print randint(10, 20, (4,2))
[[19 14]
 [14 11]
 [13 11]
 [13 11]]
>>> print permutation(10)
[0 5 9 4 2 1 6 8 3 7]
>>> print permutation(10)
[3 7 1 2 9 0 4 8 5 6]
```

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],
       [ 0.37601032,  0.25528411],
       [ 0.49313049,  0.94909878]])
```

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.random.rand.html

NumPy

```
>>> from FFT import fft, inverse_fft
>>> data = array((1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0))
>>> print data
[ 1.  0.  0.  0.  1.  0.  0.  0.]
>>> print fft(data)
[ 2.+0.j  0.+0.j  2.+0.j  0.+0.j  2.+0.j  0.+0.j  2.+0.j  0.+0.j]
>>> print inverse_fft(fft(data))
[ 1.+0.j  0.+0.j  0.+0.j  0.+0.j  1.+0.j  0.+0.j  0.+0.j  0.+0.j]

>>> from LinearAlgebra import inverse
>>> data = array(((1.0,2), (4,5)))
>>> print data
[[ 1.  2.]
 [ 4.  5.]]
>>> print inverse(data)
[[-1.66666667  0.66666667]
 [ 1.33333333 -0.33333333]]
>>> print inverse(inverse(data))
[[ 1.  2.]
 [ 4.  5.]]
```

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.fft.fft.html

Matrix diagonalization

```
>>> w, v = LA.eig(np.array([[1, -1], [1, 1]]))
>>> w; v
array([ 1. + 1.j,  1. - 1.j])
array([[ 0.70710678+0.j        ,  0.70710678+0.j        ],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]])
```

```
>>> a = np.array([[1, 1j], [-1j, 1]])
>>> w, v = LA.eig(a)
>>> w; v
array([  2.00000000e+00+0.j,   5.98651912e-36+0.j]) # i.e., {2, 0}
array([[ 0.00000000+0.70710678j,  0.70710678+0.j        ],
       [ 0.70710678+0.j        ,  0.00000000+0.70710678j]])
```

https://sites.google.com/a/aims-senegal.org/scipy/the-numpy-linear-algebra-module-linalg

Matrix diagonalization

```
>>> from numpy import linalg as LA
```

(Almost) trivial example with real e-values and e-vectors.

```
>>>
>>> w, v = LA.eig(np.diag((1, 2, 3)))
>>> w; varray([ 1.,  2.,  3.])array([[ 1.,  0.,  0.],       [ 0.,  1.,  0.],       [ 0.,  0.,  1.]])
```

Real matrix possessing complex e-values and e-vectors; note that the e-values are complex conjugates of each other.

```
>>>
>>> w, v = LA.eig(np.array([[1, -1], [1, 1]]))
>>> w; varray([ 1. + 1.j,  1. - 1.j])array([[ 0.70710678+0.j        ,  0.70710678+0.j        ],       [ 0.00000000-0.70710678j,  0.00000000+0.70710
```

Complex-valued matrix with real e-values (but complex-valued e-vectors); note that a.conj().T = a, i.e., a is Hermitian.

```
>>>
>>> a = np.array([[1, 1j], [-1j, 1]])
>>> w, v = LA.eig(a)
>>> w; varray([  2.00000000e+00+0.j,   5.98651912e-36+0.j]) # i.e., {2, 0}array([[ 0.00000000+0.70710678j,  0.70710678+0.j        ],       [ 0.7071
```

Be careful about round-off error!

```
>>>
>>> a = np.array([[1 + 1e-9, 0], [0, 1 - 1e-9]])
>>> # Theor. e-values are 1 +/- 1e-9>>> w, v = LA.eig(a)
>>> w; varray([ 1.,  1.])array([[ 1.,  0.],       [ 0.,  1.]])
```

http://incredible.egloos.com/7193290

Files and directories

```
import os
directory = 'c:/users/heinold/desktop/'
files = os.listdir(directory)
for f in files:
    if os.path.isfile(directory+f):
        s = open(directory+f).read()
        if 'hello' in s:
            print(f)
```

```
import os
import shutil
directory = 'c:/users/heinold/desktop/'
files = os.listdir(directory)
for f in files:
    if os.path.isfile(directory+f):
        shutil.copy(directory+f, directory+'Copy of '+f)
```

http://incredible.egloos.com/2950369

File existence

```
import os.path
os.path.exists(file_path)


os.path.isfile(file_name)
```

```
with open('seven.txt') as f:
    for line in f:
        print(line)
```

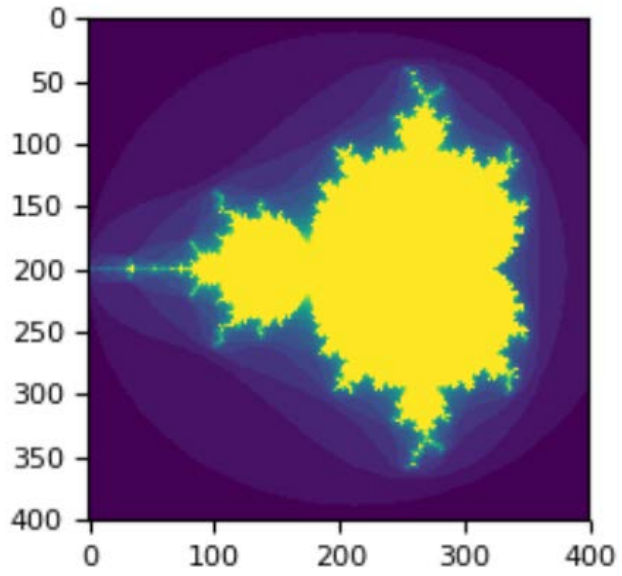
```
#!/usr/bin/python
import os

# Delete file test2.txt
os.remove("text2.txt")
```

```
with open('seven.txt') as f:
    while True:
        line = f.readline()
        if not line:
            break
        print(line)
```

```
#!/usr/bin/python
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

NumPy

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> def mandelbrot( h,w, maxit=20 ):
...     """Returns an image of the Mandelbrot fractal of size (h,w)."""
...     y,x = np.ogrid[ -1.4:1.4:h*1j, -2:0.8:w*1j ]
...     c = x+y*1j
...     z = c
...     divtime = maxit + np.zeros(z.shape, dtype=int)
...
...     for i in range(maxit):
...         z = z**2 + c
...         diverge = z*np.conj(z) > 2**2            # who is diverging
...         div_now = diverge & (divtime==maxit)  # who is diverging now
...         divtime[div_now] = i                  # note when
...         z[diverge] = 2                        # avoid diverging too much
...
...     return divtime
>>> plt.imshow(mandelbrot(400,400))
>>> plt.show()
```




```
>>> 1j
1j
>>> 1J
1j
>>> 1j * 1j
(-1+0j)
```

```
>>> complex(2,3)
(2+3j)
```

```
>>> z = 2+3j
>>> z.real
2.0
>>> z.imag
3.0
>>> z.conjugate()
(2-3j)
```

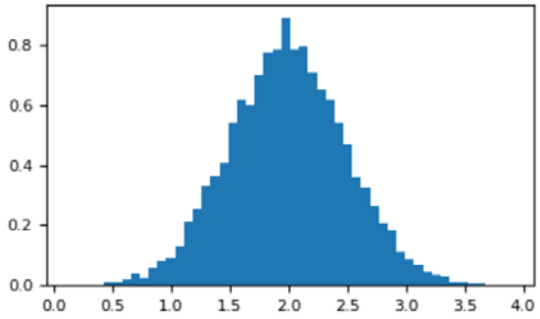
```
>>> import cmath
>>> cmath.sin(2 + 3j)
(9.15449914691143-4.168906959966565j)
```

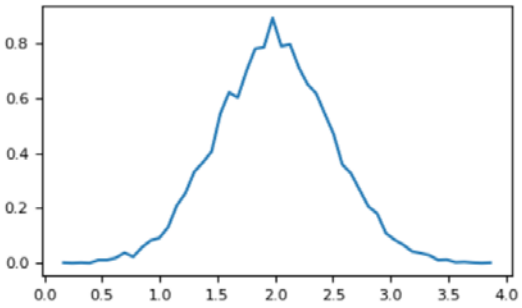
https://docs.scipy.org/doc/numpy/user/quickstart.html

NumPy

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> # Build a vector of 10000 normal deviates with variance 0.5^2 and mean 2
>>> mu, sigma = 2, 0.5
>>> v = np.random.normal(mu,sigma,10000)
>>> # Plot a normalized histogram with 50 bins
>>> plt.hist(v, bins=50, normed=1)       # matplotlib version (plot)
>>> plt.show()
```

```
>>> # Compute the histogram with numpy and then plot it
>>> (n, bins) = np.histogram(v, bins=50, normed=True)  # NumPy version (no plot)
>>> plt.plot(.5*(bins[1:]+bins[:-1]), n)
>>> plt.show()
```





https://docs.scipy.org/doc/numpy/user/quickstart.html

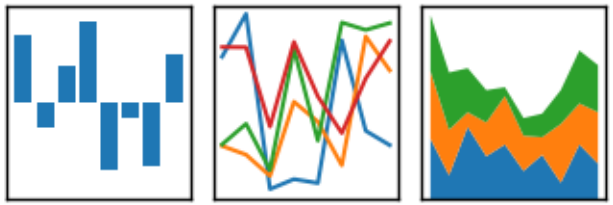
Pandas: The Swiss Army Knife for Your Data

pip install pandas

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



https://pandas.pydata.org/

https://code.tutsplus.com/tutorials/pandas-the-swiss-army-knife-for-your-data-part-1--cms-29523

Cython

RPython(←**R**estricted **Python**) → PyPy

```python
# python_functions.py
# -----------------------
import math

def f(x):
    return math.exp(-(x ** 2))

def integrate_f(a, b, N):
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

```python
# cython_functions.pyx
# -----------------------
from libc cimport math

cdef double f(double x):
    return math.exp(-(x ** 2))

def integrate_f(double a, double b, int N):
    cdef double s = 0
    cdef double dx = (b - a) / N
    cdef int i
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

- CPython: The pure Python implementation running in the standard CPython interpreter
- CPython + Cython: The Cython implementation running in the standard CPython interpreter
- PyPy: The pure Python implementation running in the PyPy interpreter

https://cardinalpeak.com/blog/faster-python-with-cython-and-pypy-part-1/
https://cardinalpeak.com/blog/faster-python-with-cython-and-pypy-part-2/

Cython

```
import time
t0 = time.clock()
p = dice6_py(N, ndice, nsix)
t1 = time.clock()
print 'CPU time for loops in Python:', t1-t0
```

https://medium.com/@shamir.stav_83310/making-your-c-library-callable-from-python-by-wrapping-it-with-cython-b09db35012a3

http://hplgit.github.io/teamods/MC_cython/main_MC_cython-solarized.html

Cython

pip install cython
pip3 install cython
python setup.py install
cython -V

Cython is a programming language.

C is **fast** whereas Python is **slow**.
You take the best of two worlds and combine them to fit **your needs**.

Cython is an extension to the Python language that allows explicit type declarations and is compiled directly to C.

Reduces program runtime by 50–100 times if done correctly, depending on the complexity of the code.

Requires very little additional work: **90% of the job is writing the python code!**
Compilation is automatic, all you need to specify is a name.

Cython is…
an Open Source project
a Python compiler (almost)
an extended Python language for
writing fast Python extension modules
interfacing Python with C (C++, etc.) libraries

static type declarations

F calls C.
C calls F.

One of main uses of Cython is to call external C libraries from Python.

Cython

Copy your Python code and rename it with **.pyx**
Copy-paste the import statements.
Assuming your Python code is bug-free, turn off array boundschecks.
Declare variables used in the **slow regions of the code.**
Remember: Profile your code. No need to cythonize everything.

- `cython -a NAME.pyx`
- `open NAME.html`
- Lines are colored according to the level of "typedness" – white lines translates to pure C without any Python API calls.

Cython

helloworld.pyx

```
print "Hello World"
```

setup.py

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    ext_modules = cythonize("helloworld.pyx")
)
```

```
$ python setup.py build_ext --inplace
```

helloworld.so (Unix) helloworld.pyd (Windows) **Shared Objects**

test.py

```
>>> import helloworld
Hello World
```

python setup.py build_ext -i --compiler=mingw32 -DMS_WIN64
python setup.py build_ext -i

Cython

fib.pyx

```
def fib(n):
    """Print the Fibonacci series up to n."""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b
```

setup.py

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    ext_modules=cythonize("fib.pyx"),
)
```

```
$ python setup.py build_ext --inplace
```

test.py

```
>>> import fib
>>> fib.fib(2000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Cython

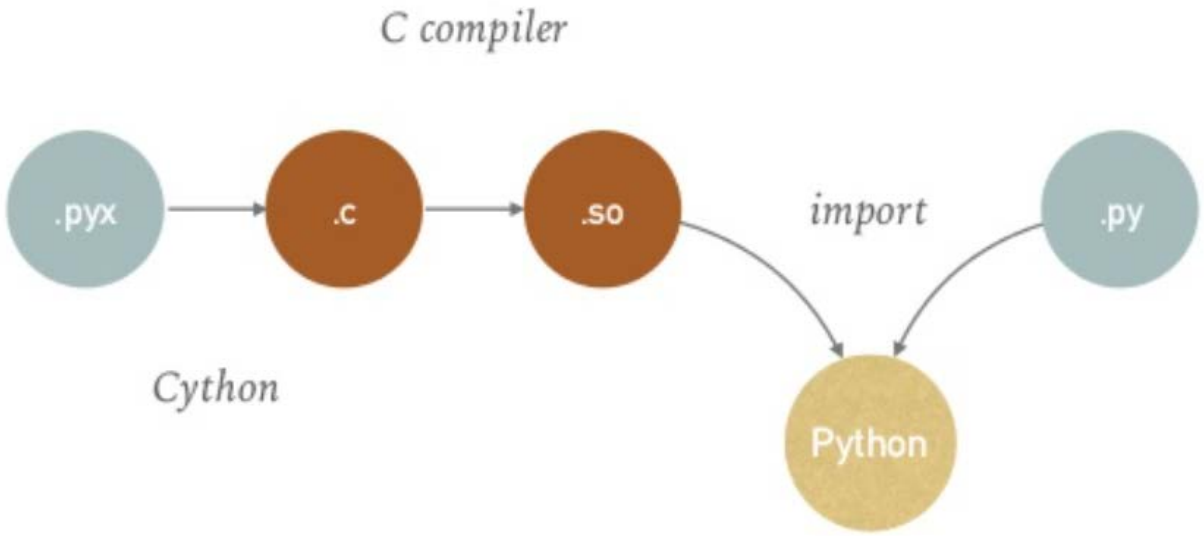
Copy-paste the setup.py file:

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

numpy = "/local/lib/python2.5/site-packages/numpy/core/include/"

setup(
    cmdclass = {'build_ext': build_ext},
    ext_modules = [Extension("my_lib", ["superfast.pyx"],
                             include_dirs = [numpy])]
)
```

Create the linked library by running the setup.py file. Then you may simply import it:

```
$ python setup.py build_ext --inplace
$ python -c "import my_lib"
```

Cython



cdef-ed mode?

cdef int x=0

.pyx → .c → .so → python

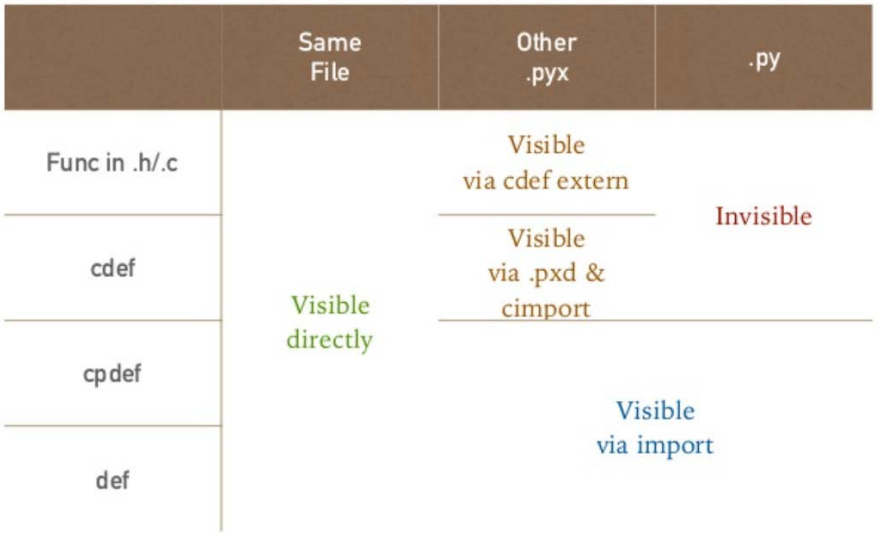
cython inputfile.pyx

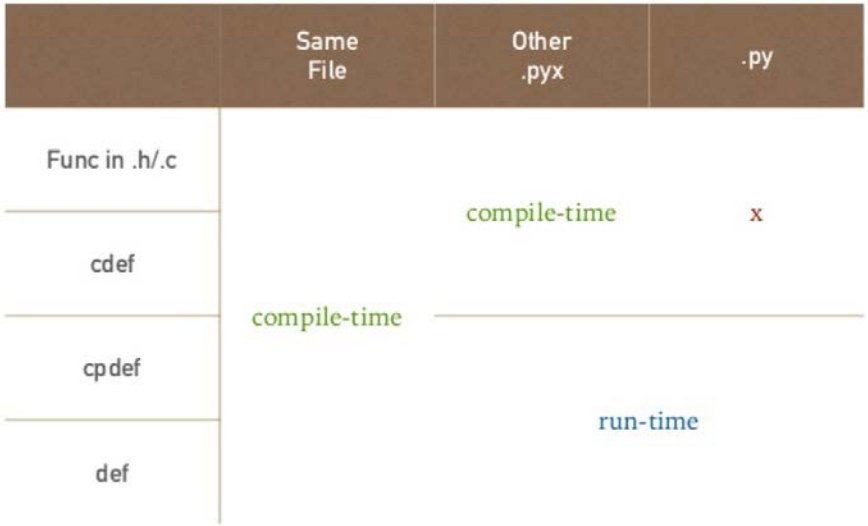
python setup.py build_ext --inplace

Cython

```
cdef int I,j,k
cdef float f, g[44], *h
```

```
cdef  struct grail:
   int age
   float volume
```





Cython

```
def f(x):
    return x**2-x

def integrate_f(a, b, N):
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

```
def f(double x):
    return x**2-x

def integrate_f(double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

```
cdef double f(double x) except? -2:
    return x**2-x
```

Cython

mathx.pyx

```
def inverse(float x):
    cdef float y
    y = 1.0 / x
    return y
```

setup.py

```
from distutils.core import setup
from Cython.Build import cythonize

setup(name = 'Cython Library',
      ext_modules = cythonize("*.pyx"))
```

scix.pyx

```
import math

def kmc( float x):
    cdef float t
    t = math.log( x)
    return t
```

```
python setup.py build_ext --inplace
```

```
>>> import mathx, scix
>>> mathx.inverse( 3.0)
>>> scix.kmc( 5.0)
```

Cython

scix_c.pyx

```
cdef extern from "math.h":
    double log(double x)

def kmc( float x):
    cdef float t
    t = log( x)
    return t
```

scix.pyx

```
import math

def kmc( float x):
    cdef float t
    t = math.log( x)
    return t
```

Speeding Up Python With Cython 1/5

```python
import time


def count(limit):
    result = 0
    for a in range(1, limit + 1):
        for b in range(a + 1, limit + 1):
            for c in range(b + 1, limit + 1):
                if c * c > a * a + b * b:
                    break

                if c * c == (a * a + b * b):
                    result += 1
    return result


if __name__ == '__main__':
    start = time.time()
    result = count(1000)
    duration = time.time() - start
    print(result, duration)
```

```
881 13.883624076843262
```

```
>>> import timeit
>>> timeit.timeit('count(1000)', setup='from pythagorean_triples import count', number=1)
0.05357028398429975

# Running 10 times
>>> timeit.timeit('count(1000)', setup='from pythagorean_triples import count', number=10)
0.5446877249924
```

Speeding Up Python With Cython 2/5

pythagorean_triples.pyx

compiles your Cython code on the fly

```
import time
import pyximport; pyximport.install()
import pythagorean_triples


def main():
    start = time.time()
    result = pythagorean_triples.count(1000)
    duration = time.time() - start
    print(result, duration)


if __name__ == '__main__':
    main()
```

```
881 9.432806253433228
```

Speeding Up Python With Cython 3/5

setup.py

to avoid dynamically compiling on every run
to build it before running

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    ext_modules = cythonize("pythagorean_triples.pyx")
)
```

$ python setup.py build_ext --inplace

```
$ python setup.py build_ext --inplace
Compiling pythagorean_triples.pyx because it changed.
[1/1] Cythonizing pythagorean_triples.pyx
running build_ext
building 'pythagorean_triples' extension
creating build
creating build/temp.macosx-10.7-x86_64-3.6
gcc -Wno-unused-result -Wsign-compare -Wunreachable-code
-DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
-I/Users/gigi.sayfan/miniconda3/envs/py3/include
-arch x86_64 -I/Users/gigi.sayfan/miniconda3/envs/py3/include
-arch x86_64
-I/Users/gigi.sayfan/miniconda3/envs/py3/include/python3.6m
-c pythagorean_triples.c
-o build/temp.macosx-10.7-x86_64-3.6/pythagorean_triples.o
gcc -bundle -undefined dynamic_lookup
-L/Users/gigi.sayfan/miniconda3/envs/py3/lib
-L/Users/gigi.sayfan/miniconda3/envs/py3/lib
-arch x86_64
build/temp.macosx-10.7-x86_64-3.6/pythagorean_triples.o
-L/Users/gigi.sayfan/miniconda3/envs/py3/lib
-o pythagorean_triples.cpython-36m-darwin.so
```

← pythagorean_triples.c

Speeding Up Python With Cython 4/5

setup.py

```
import time
import pythagorean_triples


def main():
    start = time.time()
    result = pythagorean_triples.count(1000)
    duration = time.time() - start
    print(result, duration)


if __name__ == '__main__':
    main()
```

```
881 9.507064819335938
```

Speeding Up Python With Cython 5/5

pythagorean_triples.pyx; modified version

```
# pythagorean_triples.pyx
def count(limit):
    cdef int result = 0
    cdef int a = 0
    cdef int b = 0
    cdef int c = 0

    for a in range(1, limit + 1):
        for b in range(a + 1, limit + 1):
            for c in range(b + 1, limit + 1):
                if c * c > a * a + b * b:
                    break

                if c * c == (a * a + b * b):
                    result += 1
    return result
```

```
# main.py

import time
import pyximport; pyximport.install()
import pythagorean_triples


def main():
    start = time.time()
    result = pythagorean_triples.count(1000)
    duration = time.time() - start
    print(result, duration)


if __name__ == '__main__':
    main()
```

```
881 0.05641412734985352
```

to call C functions from Python

add.c

```c
//sample C file to add 2 numbers - int and floats

#include <stdio.h>

int add_int(int, int);
float add_float(float, float);

int add_int(int num1, int num2){
    return num1 + num2;
}

float add_float(float num1, float num2){
    return num1 + num2;
}
```

```
#For Linux
$  gcc -shared -Wl,-soname,adder -o adder.so -fPIC add.c

#For Mac
$ gcc -shared -Wl,-install_name,adder.so -o adder.so -fPIC add.c
```

```python
from ctypes import *

#Load the shared object file
adder = CDLL('./adder.so')

#Find sum of integers
res_int = adder.add_int(4,5)
print "Sum of 4 and 5 = " + str(res_int)

#Find sum of floats
a = c_float(5.5)
b = c_float(4.1)

add_float = adder.add_float
add_float.restype = c_float
print "Sum of 5.5 and 4.1 = ", str(add_float(a, b))
```

add.so (DLL in Windows)

Simplified Wrapper and Interface Generator(SWIG)

This is a great method when you have a C/C++ code base, and you want to interface it to many different languages.

http://book.pythontips.com/en/latest/python_c_extension.html

F2Py

```
! file: helloworld.f90
subroutine sayhello(comm)
  use mpi
  implicit none
  integer :: comm, rank, size, ierr
  call MPI_Comm_size(comm, size, ierr)
  call MPI_Comm_rank(comm, rank, ierr)
  print *, 'Hello, World! I am process ',rank,' of ',size,'.'
end subroutine sayhello
```

```
$ f2py -c --f90exec=mpif90 helloworld.f90 -m helloworld
```

```
>>> from mpi4py import MPI
>>> import helloworld
>>> fcomm = MPI.COMM_WORLD.py2f()
>>> helloworld.sayhello(fcomm)
Hello, World! I am process 0 of 1.
```

F2Py

build f2py modules.py

```
# The content of build_f2py_modules.py

from numpy.distutils.core import Extension

ext = Extension(name     = 'chosen_module_name',
                sources = ['fortran_file_name.f90'] )

if __name__ == "__main__":
    from numpy.distutils.core import setup
    setup(name = 'f2py_example',
          ext_modules = [ext]
          )
```

```
f2py -c -m chosen_module_name fortran_file_name.f90
```

```
>>> import numpy
>>> import chosen_module_name
>>> print( chosen_module_name.__doc__ )

>>> a = numpy.linspace(0,1,10)
>>> A = chosen_module_name.fortran_sum(1.2,4.2)
>>> print(A)
 5.4
>>>
```

to call F functions from Python

my_lib.f90

```
subroutine threshold_image(image, n, threshold, output)
    ! Inputs: image, n, threshold.
    ! Output: output
    !   output(i,j) is 1 if image(i,j) > threshold and 0 otherwise.

    integer n
    real(8) threshold
    real(8), dimension(n,n) :: image, output

    !f2py intent(in) :: image, threshold
    !f2py intent(hide), depend(image) :: n = shape(image, 0)
    !f2py intent(out) output

    write(*,*) "Hello from the Fortran subroutine!"

    ! Loop through columns and rows and threshold the image.
    do j=1,n
        do i=1,n
            if (image(i,j) > threshold) then
                output(i,j) = 1.0
            else
                output(i,j) = 0.0
            end if
        end do
    end do
end subroutine
```

```
import numpy as np
import matplotlib.pyplot as plt

import my_lib as ml

# Read matrix from text file as double precision matrix.
I = np.genfromtxt('./image.txt', dtype=np.float64)

# Threshold value.
t = 0.3

# Call the fortran routine.
T = ml.threshold_image(image=I, threshold=t)

# Plot the images.
fig, axes = plt.subplots(1, 2)

axes[0].imshow(I, cmap=plt.cm.gray)
axes[0].axis('off'); axes[0].set_title('Original')

axes[1].imshow(T, cmap=plt.cm.gray)
axes[1].axis('off'); axes[1].set_title('Thresholded at %.2f' % t)

plt.tight_layout()
plt.show()
```

f2py −c −m my_lib my_lib.f90

my_lib.so

https://notmatthancock.github.io/2017/02/10/calling-fortran-from-python.html

http://www.legi.grenoble-inp.fr/people/Pierre.Augier/how-to-call-fortran-from-python.html

RPython

RPython(←**R**estricted **Python**) → PyPy

Explicit typing in RPython

RPython is a translation and support framework for producing implementations of dynamic languages, emphasizing a clean separation between language specification and implementation aspects.

```
pypy <PyPysourcedirectory>/rpython/bin/rpython fib.py
python <PyPysourcedirectory>/rpython/bin/rpython fib.py
```

https://rpython.readthedocs.io/en/latest/
https://ko.wikipedia.org/wiki/RPython

PyPy, RPython

RPython(←**R**estricted **Python**) → PyPy

PyPy is running the exact same pure Python code as the CPython implementation.

PyPy is an alternative implementation of the Python programming language which often runs faster than the standard implementation of Python.
In computing, **just-in-time** (**JIT**) **compilation**, (also **dynamic translation** or **run-time compilation**), is a way of executing computer code that involves compilation during execution of a program – at run time – rather than prior to execution.

Most Python code runs well on PyPy, except for code that relies on Cpython extensions.

C Extension Modules are the single biggest barrier to use PyPy.

https://en.wikipedia.org/wiki/PyPy
https://cardinalpeak.com/blog/faster-python-with-cython-and-pypy-part-1/
https://cardinalpeak.com/blog/faster-python-with-cython-and-pypy-part-2/

mpi4py

MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface

Focus is in translating MPI syntax and semantics: If you know MPI, MPI4Py is "obvious"

You can communicate Python objects!!

What you lose in performance, you gain in shorter development time

mpi4py

There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, most commonly used are

No need to call MPI_Init() or MPI_Finalize()

–MPI_Init() is called when you import the module

–MPI_Finalize() is called before the Python process ends

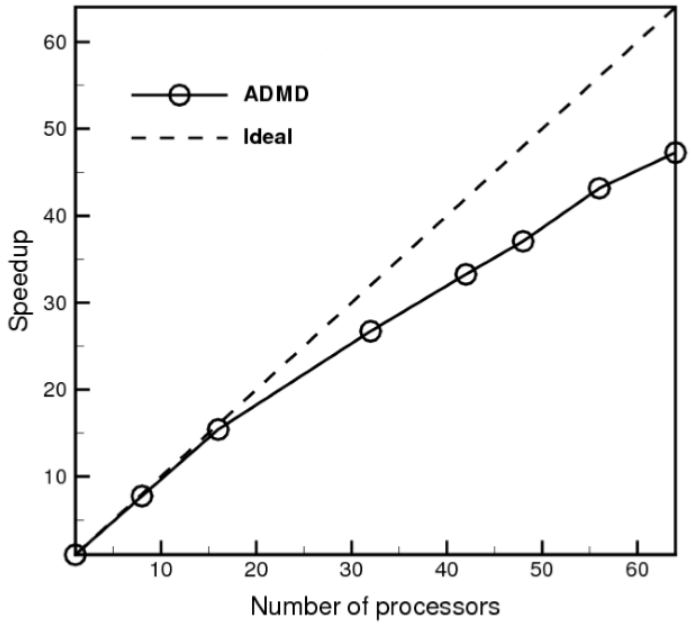
- petsc4py
- mpi4py
- NumPy

speedup

$$S_p = \frac{T_1}{T_p}$$

p is the number of processors.
T_p is the execution time of the algorithm with p processors.

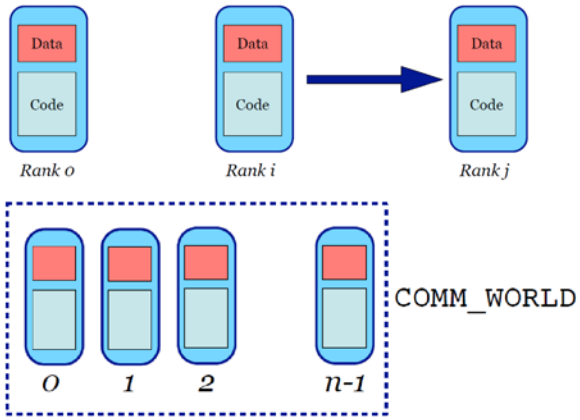
Not CPU time, but wall-clock time reference



Y. Aoyama and J. Nakano, *RS/6000 SP: Practical MPI Programming*

Python with MPI

- All processors execute the same program
- Every process has a unique rank
- Branch on the rank
- Point-to-point, collective communications
- Blocking, nonblocking communications



install

Mpi4py INSTALL

https://bitbucket.org/mpi4py/mpi4py/downloads

- $ tar -zxf mpi4py-1.3.1.tar.gz
- $ cd mpi4py-1.3.1
- $ python setup.py build
- $ python setup.py install

from mpi4py import MPI

mpi4py-1.3.1

```
[ihlee@tucana-master mpi4py_tuto]$ ls /usr/local/mpi4py-1.3.1
total 80
4 build/   4 demo/   8 HISTORY.txt   8 mpi.cfg    4 README.txt  20 setup.py   4 test/
4 conf/    4 docs/   4 LICENSE.txt   4 PKG-INFO   4 setup.cfg    4 src/       4 THANKS.txt
[ihlee@tucana-master mpi4py_tuto]$ ls /usr/local/mpi4py-1.3.1/demo/
total 116
4 compute-pi/     4 helloworld.f90  4 mpe-logging/     4 osu_bw.py          4 sequential/   4 wrap-c/
4 cython/         4 helloworld.py   4 mpi-ref-v1/      4 osu_latency.py     4 spawning/     4 wrap-cython/
4 embedding/      4 init-fini/      4 nxtval/          4 osu_multi_lat.py   4 threads/      4 wrap-f2py/
4 helloworld.c    4 makefile        4 osu_alltoall.py  4 README.txt         4 vampirtrace/  4 wrap-swig/
4 helloworld.cxx  4 mandelbrot/     4 osu_bibw.py      4 reductions/        4 wrap-boost/
[ihlee@tucana-master mpi4py_tuto]$
```

Hello World

```
[ihlee@tucana-master mpi4py_tuto]$ cat /usr/local/mpi4py-1.3.1/demo/helloworld.py
#!/usr/bin/env python
"""
Parallel Hello World
"""

from mpi4py import MPI
import sys

size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()

sys.stdout.write(
    "Hello, World! I am process %d of %d on %s.₩n"
    % (rank, size, name))
[ihlee@tucana-master mpi4py_tuto]$
```

test

```
[ihlee@tucana-master mpi4py_tuto]$ cp /usr/local/mpi4py-1.3.1/demo/helloworld.py .
[ihlee@tucana-master mpi4py_tuto]$ mpirun -np 5 python helloworld.py
Hello, World! I am process 2 of 5 on master.hpc.
Hello, World! I am process 3 of 5 on master.hpc.
Hello, World! I am process 4 of 5 on master.hpc.
Hello, World! I am process 0 of 5 on master.hpc.
Hello, World! I am process 1 of 5 on master.hpc.
[ihlee@tucana-master mpi4py_tuto]$
```

Issuing at the command line:

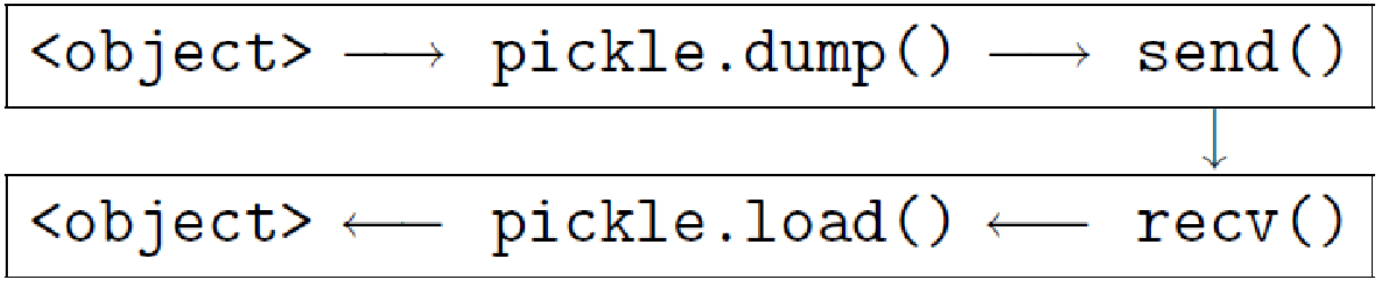
```
$ mpiexec -n 5 python demo/helloworld.py
```

or (in the case of older MPI-1 implementations):

```
$ mpirun -np 5 python demo/helloworld.py
```

mpiexec *vs* mpirun

- Communication of Python objects.
  - high level and very convenient, based in `pickle` serialization
  - can be slow for large data (CPU and memory consuming)



- Communication of array data (e.g. **NumPy** arrays).
  - lower level, slightly more verbose
  - very fast, almost C speed (for messages above 5-10 KB)

```
message = [<object>, (count, displ), datatype]
```

- Broadcasting a Python dictionary:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'key1' : [7, 2.72, 2+3j],
            'key2' : ( 'abc', 'xyz')}
else:
    data = None
data = comm.bcast(data, root=0)
```

- Scattering Python objects:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
    data = [(i+1)**2 for i in range(size)]
else:
    data = None
data = comm.scatter(data, root=0)
assert data == (rank+1)**2
```

- Gathering Python objects:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

data = (rank+1)**2
data = comm.gather(data, root=0)
if rank == 0:
    for i in range(size):
        assert data[i] == (i+1)**2
else:
    assert data is None
```

Importing library:

```
from mpi4py import MPI
```

Getting important information:

```
comm = MPI.COMM_WORLD
rank = MPI.COMM_WORLD.Get_rank()
size = MPI.COMM_WORLD.Get_size()
name = MPI.Get_processor_name()
```

Executing in parallel:

```
mpirun -np <P> python <code>.py
```

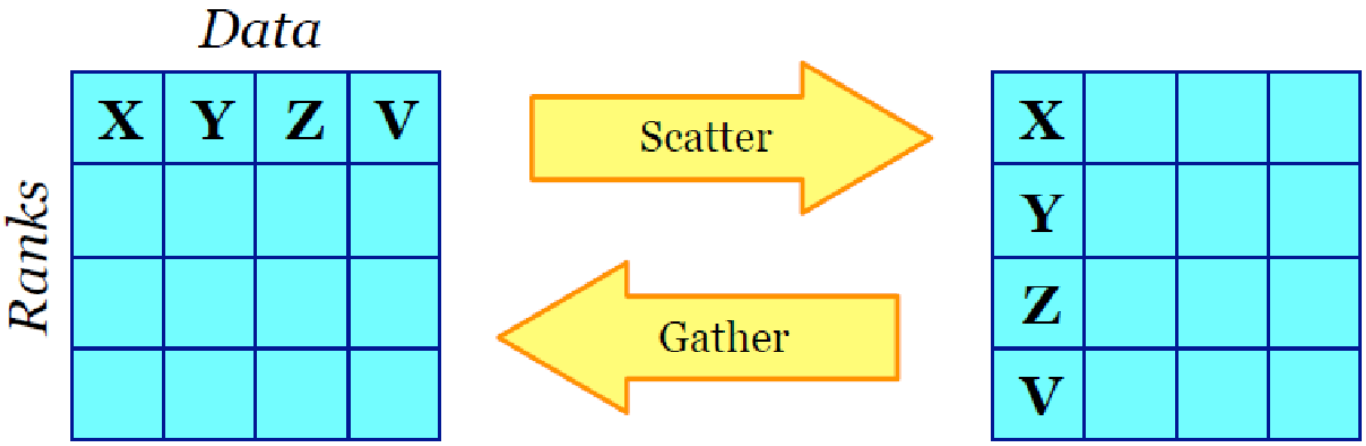
```
import time
t = time.time()
time.sleep(10)
t2 = time.time()

spendtime = t2 - t
print("Before timestemp: ", t)
print("After timestemp: ", t2)
print("Wait {0} seconds".format(spendtime))
```

```
>>>
('Before timestemp: ', 1321780317.218)
('After timestemp: ', 1321780327.218)
Wait 10.0 seconds
```

```
[ihlee@tucana-master ~]$ python
Python 2.6.6 (r266:84292, Jul 10 2013, 22:48:45)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import time
>>> t1=time.time()
>>> t2=time.time()
>>> print t2-t1
7.2504940033
>>>
```

Scatter/Gather

./s_g.py
After Scatter:
[0] [0. 1. 2. 3.]
After Allgather:
[0] [0. 2. 4. 6.]
[ihlee@tucana-master mpi4py_tuto]$ vi s_g.py



```
#!/usr/bin/env python
#--------------
# Loaded Modules
#--------------
import numpy as np
from mpi4py import MPI
#-------------
# Communicator
#-------------
comm = MPI.COMM_WORLD

my_N = 4
N = my_N * comm.size

if comm.rank == 0:
   A = np.arange(N, dtype=np.float64)
else:
   #Note that if I am not the root processor A is an empty array
   A = np.empty(N, dtype=np.float64)
my_A = np.empty(my_N, dtype=np.float64)
#------------------------
# Scatter data into my_A arrays
#------------------------
comm.Scatter( [A, MPI.DOUBLE], [my_A, MPI.DOUBLE] )

if comm.rank == 0:
  print "After Scatter:"

for r in xrange(comm.size):
   if comm.rank == r:
      print "[%d] %s" % (comm.rank, my_A)
   comm.Barrier()
```

```
#-----------------------
# Everybody is multiplying  by 2
#-----------------------
my_A *= 2

#---------------------
# Allgather  data into A again
#---------------------
comm.Allgather(  [my_A, MPI.DOUBLE], [A, MPI.DOUBLE] )

if comm.rank == 0:
   print "After Allgather:"

for r in xrange(comm.size):
    if comm.rank == r:
        print "[%d] %s" % (comm.rank, A)
    comm.Barrier()
```

```
from mpi4py import MPI
import numpy as np

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()
SIZE = COMM.Get_size()
N = 10

if(RANK == 0):
    DATA = np.arange(N*SIZE, dtype='i')
    for i in range(1, SIZE):
        SLICE = DATA[i*N:(i+1)*N]
        COMM.Send([SLICE, MPI.INT], dest=i)
    MYDATA = DATA[0:N]
else:
    MYDATA = np.empty(N, dtype='i')
    COMM.Recv([MYDATA, MPI.INT], source=0)
```

```
S = sum(MYDATA)
print RANK, 'has data', MYDATA, 'sum =', S

SUMS = np.zeros(SIZE, dtype='i')
if(RANK > 0):
    COMM.send(S, dest=0)
else:
    SUMS[0] = S
    for i in range(1, SIZE):
        SUMS[i] = COMM.recv(source=i)
    print 'total sum =', sum(SUMS)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

if comm.rank == 0:
    sendmsg = (7, "abc", [1.0,2+3j], {3:4})
else:
    sendmsg = None

recvmsg = comm.bcast(sendmsg, root=0)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

if comm.rank == 0:
    sendmsg = [i**2 for i in range(comm.size)]
else:
    sendmsg = None

recvmsg = comm.scatter(sendmsg, root=0)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

sendmsg = comm.rank**2

recvmsg1 = comm.gather(sendmsg, root=0)

recvmsg2 = comm.allgather(sendmsg)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

sendmsg = comm.rank

recvmsg1 = comm.reduce(sendmsg, op=MPI.SUM, root=0)

recvmsg2 = comm.allreduce(sendmsg)
```

MPI

- Many examples
- High-level usage of MPI
- Practice

Serial version

```
import math

def compute_pi(n):
    h = 1.0 / n
    s = 0.0
    for i in range(n):
        x = h * (i + 0.5)
        s += 4.0 / (1.0 + x**2)
    return s * h

n = 10
pi = compute_pi(n)
error = abs(pi - math.pi)

print ("pi is approximately %.16f, "
       "error is %.16f" % (pi, error))
```

Parallel version

```
from mpi4py import MPI
import math

def compute_pi(n, start=0, step=1):
    h = 1.0 / n
    s = 0.0
    for i in range(start, n, step):
        x = h * (i + 0.5)
        s += 4.0 / (1.0 + x**2)
    return s * h

comm = MPI.COMM_WORLD
nprocs = comm.Get_size()
myrank = comm.Get_rank()
```

```
if myrank == 0:
    n = 10
else:
    n = None

n = comm.bcast(n, root=0)

mypi = compute_pi(n, myrank, nprocs)

pi = comm.reduce(mypi, op=MPI.SUM, root=0)

if myrank == 0:
    error = abs(pi - math.pi)
    print ("pi is approximately %.16f, "
           "error is %.16f" % (pi, error))
```

Serial version

```
def mandelbrot(x, y, maxit):
    c = x + y*1j
    z = 0 + 0j
    it = 0
    while abs(z) < 2 and it < maxit:
        z = z**2 + c
        it += 1
    return it

x1, x2 = -2.0, 1.0
y1, y2 = -1.0, 1.0
w,  h  = 150, 100
maxit  = 127
```

```
import numpy
C = numpy.zeros([h, w], dtype='i')
dx = (x2 - x1) / w
dy = (y2 - y1) / h
for i in range(h):
    y = y1 + i * dy
    for j in range(w):
        x = x1 + j * dx
        C[i, j] = mandelbrot(x, y, maxit)

from matplotlib import pyplot
pyplot.imshow(C, aspect='equal')
pyplot.spectral()
pyplot.show()
```

Parallel version (1/3)

```
def mandelbrot(x, y, maxit):
    c = x + y*1j
    z = 0 + 0j
    it = 0
    while abs(z) < 2 and it < maxit:
        z = z**2 + c
        it += 1
    return it

x1, x2 = -2.0, 1.0
y1, y2 = -1.0, 1.0
w,  h  = 150, 100
maxit  = 127
```

Parallel version (2/3)

```
def mandelbrot(x, y, maxit):
    c = x + y*1j
    z = 0 + 0j
    it = 0
    while abs(z) < 2 and it < maxit:
        z = z**2 + c
        it += 1
    return it

x1, x2 = -2.0, 1.0
y1, y2 = -1.0, 1.0
w,  h  = 150, 100
maxit  = 127
```

```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

# number of rows to compute here
N = h // size + (h % size > rank)

# first row to compute here
start = comm.scan(N)-N

# array to store local result
Cl = numpy.zeros([N, w], dtype='i')
```

Parallel version (3/3)

```
dx = (x2 - x1) / w
dy = (y2 - y1) / h
for i in range(N):
    y = y1 + (i + start) * dy
    for j in range(w):
        x = x1 + j * dx
        C1[i, j] = mandelbrot(x, y, maxit)
```

```
counts = comm.gather(N, root=0)
C = None
if rank == 0:
    C = numpy.zeros([h, w], dtype='i')

rowtype = MPI.INT.Create_contiguous(w)
rowtype.Commit()

comm.Gatherv(sendbuf=[C1, MPI.INT],
             recvbuf=[C, (counts, None), rowtype],
             root=0)

rowtype.Free()
```

```
if comm.rank == 0:
    from matplotlib import pyplot
    pyplot.imshow(C, aspect='equal')
    pyplot.spectral()
    pyplot.show()
```

Parallel

Most MPI programs can be run with the command **mpiexec**.

```
$ mpiexec -n 4 python script.py
```

```
#!/usr/bin/env python
"""
Parallel Hello World
"""
from mpi4py import MPI
import sys
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()
sys.stdout.write(
    "Hello, World! I am process %d of %d on %s.\n"
    % (rank, size, name))
```

Parallel

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
    fd = open("dna_file.txt", "r")
    dna = fd.read()
else:
    dna = None

dna = comm.bcast(dna, root=0)
print rank, dna
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

max = comm.reduce(rank, op=MPI.MAX, root =0)

if rank == 0:
    print "The reduction is %s" % max
```

Master-slave mode

master-slave mode, king-soldier mode

```
if(nproc > 1)then
if(myid == 0)then   ! -----[   process id = 0
call feedin1(nconf,iter)
        else   ! -----    process id /= 0
call predict1(nconf)
        endif  ! -----]   process id /= 0
        else
do ip=1,ncrystals
call grbfnn_predict(ip,tmp)
enddo
        endif
```

http://incredible.egloos.com/3755171

Parallelizing Loops

Block Distribution

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3

Cyclic Distribution

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	1	2	3	0	1	2	3	0	1	2	3	0	1

Block-Cyclic Distribution

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	1	1	2	2	3	3	0	0	1	1	2	2

iblock

Y. Aoyama and J. Nakano, *RS/6000 SP: Practical MPI Programming*
http://incredible.egloos.com/3755171

Parameter range

```
SUBROUTINE para_range(n1, n2, nprocs, irank, ista, iend)
iwork1 = (n2 - n1 + 1) / nprocs
iwork2 = MOD(n2 - n1 + 1, nprocs)
ista = irank * iwork1 + n1 + MIN(irank, iwork2)
iend = ista + iwork1 - 1
IF (iwork2 > irank) iend = iend + 1
END

SUBROUTINE para_range(n1, n2, nprocs, irank, ista, iend)
iwork = (n2 - n1) / nprocs + 1
ista = MIN(irank * iwork + n1, n2 + 1)
iend = MIN(ista + iwork - 1, n2)
END
```

Y. Aoyama and J. Nakano, *RS/6000 SP: Practical MPI Programming*
http://incredible.egloos.com/3755171

References

- John W. Shipman, *A Python 2.7 programming tutorial*
- D. Beazley and B. K. Jones*, Python Cookbook 3rd ed.*
- Tutorialpoint, *Python 3*
- Richard L. Halterman, *Fundamentals of Python Programming*
- Brian Heinold, *A Practical Introduction to Python Programming*
- K. W. Smith*, Cython*
- Stephen Weston, *Parallel Computing in Python using mpi4py*
- Lisandro Dalcin, *MPI for Python*
- Konrad Hinsen, *Optimizing and Interfacing with Cython*
- Stéfan van der Walt, *The Quest for Speed: An Introduction to Cython*
- *http://cs231n.github.io/python-numpy-tutorial/*
- *https://code.tutsplus.com/tutorials/speeding-python-with-cython--cms-29557*
- *https://docs.scipy.org/doc/numpy/user/quickstart.html*
- *http://book.pythontips.com/en/latest/python_c_extension.html*
- *https://notmatthancock.github.io/2017/02/10/calling-fortran-from-python.html*
- Y. Aoyama and J. Nakano, *RS/6000 SP: Practical MPI Programming*
- *https://en.wikibooks.org/wiki/A_Beginner%27s_Python_Tutorial/Classes*