The Python language

http://www.python.org

Python is a portable, dynamic, extensible, open source language.
Modular approach and object oriented programming.
Python is developed since 1989 by Guido van Rossum & PSF.

christophe.pradal@cirad.fr

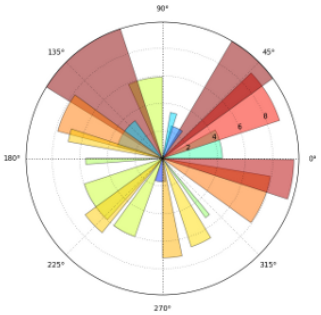
cirad

informatics mathematics
Inria

Python in Science

Scientific Computing

- NumPy,
- SciPy (www.scipy.org)
- Matplotlib (2D plot)



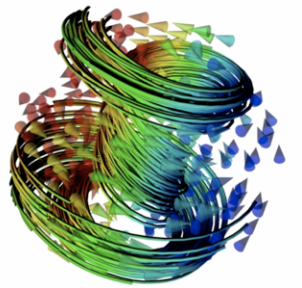
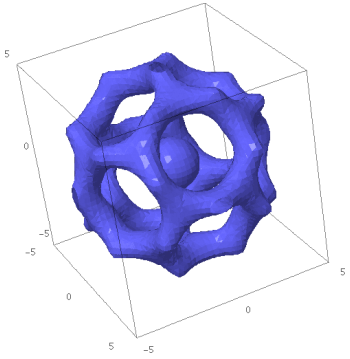


Image Analysis

- MayaVi (vtk), OpenCV, scikits.image

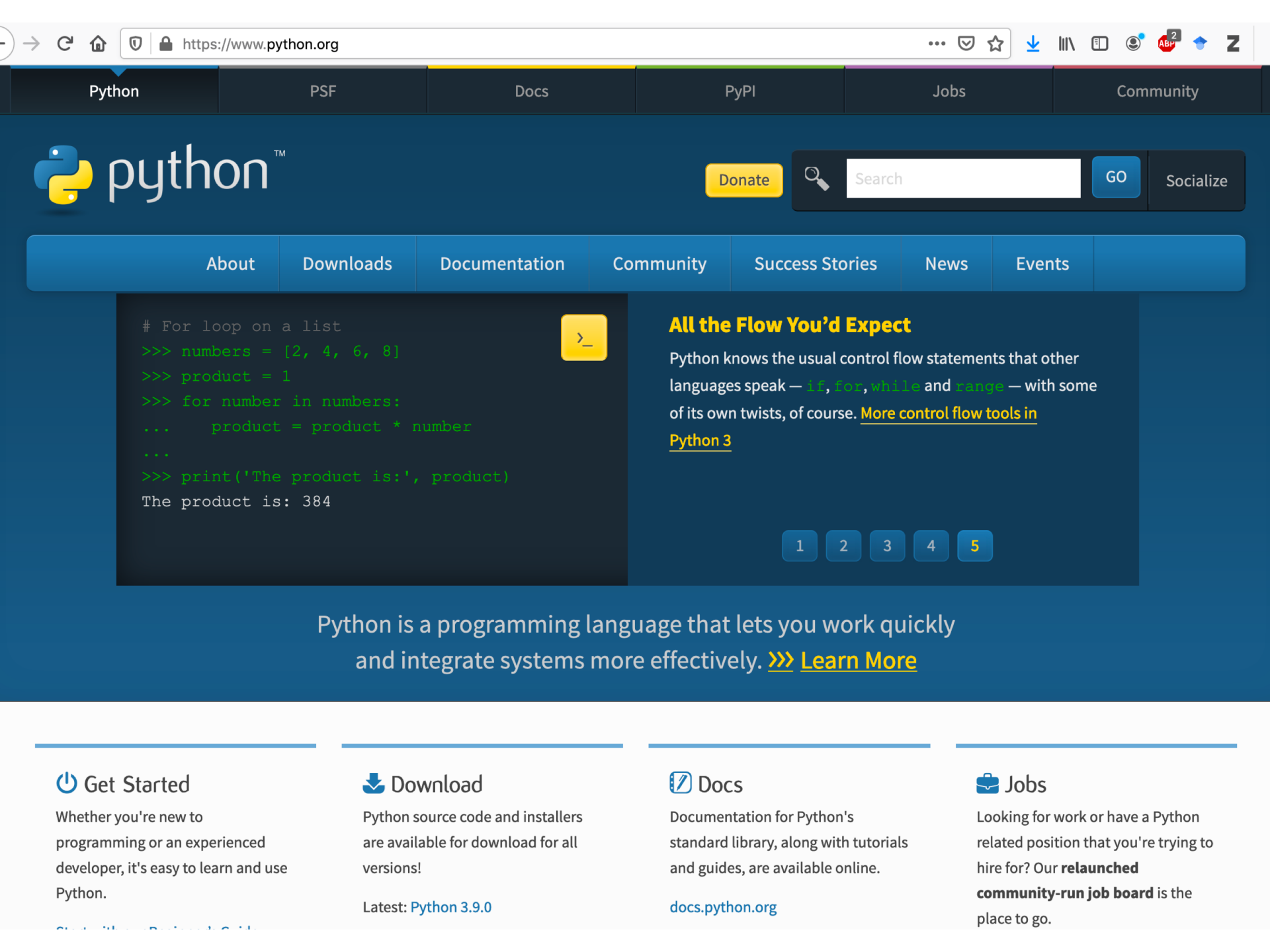


Data Science

- Pandas, scikits.learn
- PyTorch, Keras, TensorFlow

Others

- SAGE (Mathematics)
- Jupyter


https://www.python.org
Python
PSF
Docs
PyPI
Jobs
Community
python™
Donate
Search
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
# For loop on a list
>>> numbers = [2, 4, 6, 8]
>>> product = 1
>>> for number in numbers:
...   product = product * number
...
>>> print('The product is:', product)
The product is: 384
All the Flow You'd Expect
Python knows the usual control flow statements that other languages speak — if, for, while and range — with some of its own twists, of course. More control flow tools in Python 3
1
2
3
4
5
Python is a programming language that lets you work quickly and integrate systems more effectively. »» Learn More
Get Started
Whether you're new to programming or an experienced developer, it's easy to learn and use Python.
Download
Python source code and installers are available for download for all versions!
Latest: Python 3.9.0
Docs
Documentation for Python's standard library, along with tutorials and guides, are available online.
docs.python.org
Jobs
Looking for work or have a Python related position that you're trying to hire for? Our relaunched community-run job board is the place to go.

Outline

Python language

- Type and Data Structure
- Control Flow
- Function & Class
- Module & Package

Scientific Python Libraries

- NumPy
- Matplotlib
- Pandas

Python – Numeric Types

Integer

```
>>> 2 + 3
5
>>> type(5)
int
```

Float

```
3.14 10. .001 1e100 3.14e-10
```

Arithmetic Operators

```
+,-,*,/
>>> 5 + 3
8
>>> 7 + 3*4
>>> 20 / 3 # float division
>>> 20 // 3 # int division
```

Operators

```
x % y
divmod(x,y) # (x/y, x%y)
x ** y
pow(x,y)    # x**y

abs(x)
int(x), float(x),
complex(re,im)

float('inf'), float('nan')
```

Bit-string Operators

```
x | y  # or
x ^ y  # xor
x & y  # and
x << n # n bits left 2<<1==4
x >> n #n bits right 2>>1==1
~x     #bits inverted ~2==-3
```

Python – Boolean

bool

```
True, False
```

False

```
None
False
0, 0., 0j
"", (), []      # empty sequence
{}              # empty mapping
```

Boolean Operators

```
x and y
x or y
not x
```

Comparisons

```
<, <=, >, >=,
==, !=
is, is not

Supported by all objects
Same priority> bool operation

x < y <= z ⇔ x < y and y <= z

>>> x = 3

>>> if x<5 or (x>10 and x<20):
...    print("OK”)
OK

>>> if x < 5 or 10 < x < 20:
...    print("OK”)
OK
```

Python – Variables and so on

Value & Type

```
value: 2          type: integer
value:'hello'     type: string
>>> type('hello')
<type string>
```

Variables

```
Variables refer a value
>>> x = 3
# name != keywords
>>> class = 3 # error
# Aliasing
>>> y = x
```

Keywords

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

Assignement

```
Assignement statement (=)
    • create new variables
    • give them values
>>> width = 20
# multiple values
>>> x = y = z = 0
# swapping values (tuple)
>>> x, y = y, x
```

Del

```
# suppress binding name/value
>>> del x # delete x
```

Python – String (immutable)

Definition

```
string= "..." or '...'
'spam eggs‘,"spam eggs"

long string: """ or '''
"""
This is a long string
Containing several lines
"""
```

Accessing elts

```
>>> name= ‘FSPM’
>>> print(name[0], name[3])
F M
```

Sub-Strings (slicing)

```
>>> name[0:2]
FS
```

Usual Operators

```
# Concatenation
>>> s=‘little'+' '+‘fish‘
>>> s
little fish

# std function length
>>> len(s)
11

# convert str -> int
>>> ch=’15’
>>> ch + 10 # error
>>> int(ch) + 10
25

>>>(‘little ‘)*3 + ‘fish‘
little little little fish
```

Python – List

Definition

```
list= [x,'y',z]
>>> a = ['spam', 100, 1234]
```

Mutable

```
>>> a[2]= 23
>>> print(a)
['spam', 100, 23]
```

Standard functions

```
>>> del a[2] # remove a[2]
>>> len(a)
2
```

Accessing elts

```
# Like string indices
>>> a.append(1234); a[0]
'spam'
>>> a[-2] # a[len(a)-2]
100
```

Sub-List (slicing)

```
Slices : copy
[i:j] ⇔ [i,j[
>>> a[0:1]
["spam"]

[:j]  ⇔ [0,j[
>>> a[:2]
['spam', 100]

[i:]  ⇔ [i,len(a)[
>>> a[0:2]
['spam', 100, 1234]

>>> a[0:-1]
['spam', 100]

>>> 2*a[:2] + [3]
['spam',100,'spam',100,3]
```

Python – Tuple

Definition

A tuple is a sequence like a string or a list

(), (x,y,...)
(x,) # type((1)) -> int
tuple= x,y,z

Assignement

```
# pack
>>> t=1,2, "FSPM"
>>> t
(1, 2, "FSPM")
# unpack
>>> x,y,z= t
```

a, b = b, a ⇔ (a,b) = (b,a)

Assignement: variable= value
tuple (variable)= tuple(value)

Immutable

String and tuple are immutable

```
>>> t = 1,2,3
>>> del t[2] # error
>>> s=str(123)
>>> del s[2] # error
```

List are mutable

```
>>>l = [1,2,3]
>>>del l[:]
>>>l
[]
```

Sequence Operators

- len
- slicing (t[i:j])
- indices (t[i])
- +
- *

Python – Sequences (Sets, arrays)

Sets

```
set= set(sequence)
>>>basket = ['apple', 'pear', 'apple']
>>> fruit = set(basket)
>>> fruit
set(['apple', 'pear'])
'apple' in fruit # fast test
```

Operation

```
>>> a = set('abracadabra')
>>> b = set('alacazam')

>>> a # unique letters
>>> a - b # in a but not in b
>>> a | b # in either a or b
>>> a & b # in both a and b
>>> a ^ b # in a or b but not both
```

Arrays

```
array(typecode, list|str)
Typecode :'i','l','f','d'

import array
s = 'This is the array.'
? : documentation on function
>>>array.array?
```

heapq

```
from heapq import heappush
heap = []
>>> data = [(1, 'J'), (4,
'N'), (3, 'H'), (2, 'O')]
>>> for item in data:
...     heappush(heap, item)
```

Python – Dictionary

Definition

```
dict= { key:value,
        ...,
        key:value }

Keys can be any immutable type:
       int, string, tuple

>>> dict= {} # empty
>>> tree= {'a':['b','c'],
           'b':['d','e'] }

store value:    d[key]=value
extract value:  d[key]
delete pair:    del d[key]

>>> tree['c']=['f']
>>> tree['b']
['d','e']
```

Operations & methods

```
>>> del tree['c']
>>> print(tree)
{'a':['b','c'],'b':['d','e']}
>>> len(tree)

dict.keys() -> [key]
dict.values() -> [value]
dict.items() -> [(key,value)]
>>> tree.keys()
['a','b']

key in dict
>>>'a' in tree
```

Aliasing & Copy

```
alias= tree
copy= tree.copy()
```

Python – Control Flow

If

```
>>>if x == 5:
...    print("value < 5”)
...elif 10 < x < 20:
...    print("value in ]10,20[“)
...else:
...    print("more than 20")
```

LOOP: while, for

```
# Fibonacci
>>> a, b = 0, 1
>>> while b < 10:
...    a, b = b, a+b
...    print(b,)

>>> a = ['cat', 'window']
>>> for x in a:
...     print(x, len(x),)
cat 3 window 6
```

List Comprehension

```
[f(x) for x in seq] ⇔ Foreach
[f(x,y) for x in dict.items()]
for if for if for if ...

>>> vec = [2, 4, 6]
>>>[3*x for x in vec if x > 3]
[12,18]
```

Range

```
range(i) ⇔ Interval(0,i)
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(5,10)
[5,6,7,8,9]
>>> range(0,8,3)
[0,3,6]

for(i=0; i<10; i++)
>>> for i in range(10): pass
```

Python – Control Flow

CONTINUE

```
# continue with the next
# iteration
>>> for i in range(8):
...     if i % 2:
...         continue
...     print(i)
0 2 4 6
```

BREAK

```
# break out the smallest
# enclosing for or while loop

>>> for i in range(8):
...     if i % 2:
...         print('the first par:')
...         print(i)
...         break
1
```

ELSE on LOOP

```
# Loop may have a else clause
# Not executed by a break

# Search for prime number
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            break
    else:
        print(n,)
        print('is prime number')

2 is prime number
3 is prime number
5 is prime number
7 is prime number
```

Python – Function

Definition

```
# function definition
def function (arg1,..., argn):
    """docstring"""
    <block>
def f(*args, **kwds): pass
```

Return

```
# function: return
>>> def sum(a,b): return a+b

# procedure: no return
>>> def sum(a):print(a+b)
```

Default Arguments

```
>>> def sum(a= 1,b= 2):
...     return a+b
>>> sum()
3
```

Default Arguments

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

# default value evaluation
>>> i = 5
>>> def f(arg=i):
...     print(arg)

>>> i = 6
>>> f()
5
```

Keyword Arguments

```
>>> sum(b= 1,a= 2):
3
```

Python – Classes

Class definition

```
class ClassName:
   <statement-1>
   . . .
   <statement-N>
```

Class object

```
Support 2 operations:
  instantiation
    obj = Class()
  attribute reference
    obj.attribute

class MyClass(object):
  "A simple example class"
  i = 12345
  def f(self):
    return 'hello world'
```

Instantiation

```
# Create an empty object
>>> x= MyClass()
>>> print(x.i)

# Define a constructor
def __init__(self):
    self.data=[]

# Ctor with arguments
class Complex:
  def __init__(self, r, i):
    self.r= r
    self.i= i

>>> x= Complex(1.,2.)
>>> print(x.r,x.i)
(1.,2.)
```

Python – Classes

Data attribute

Not declared, like any other variables.
Exist when first assigned to.

```
class A:
    def __init__(self):
        self.r= 0
>>> a= A()
>>> print(a.r)
>>> a.t= 2
>>> del a.t
```

Method attribute

Method : function of an object

x.f() ⇔ X.f(x)
x.f(a,b,c) ⇔ X.f(x,a,b,c)

Inheritance

```
class Derived(Base):
    <statement-1>
    ...
    <statement-N>
```

All methods are virtual

```
class Base(object):
  def f(self): print("Base")
class Derived(Base):
  def f(self): print("Derived")
>>> d=Derived(); d.f()
Derived
>>> Base.f(d)
Base
```

Multiple Inheritance

```
class Derived(Base1, Base2):
    <block>
```

Python – Modules

Definition

File containing Python code

Import

```
import
>>> import os
>>> dir(os)
>>> os.name
posix
from module import *
>>> from os import *
>>> name
posix
```

Name access

```
module.__name__: module name
>>> if __name__ = "__main__":
...    run_test()
A way to add tests in a module
```

import (cont.)

```
from module import f1, f2
>>> from os import name
>>> name
posix
```

Search path

```
PYTHONPATH or
sys.path
>>> import os
>>> 'PYTHONPATH' in os.environ
False
>>> import sys
>>> sys.path
['/Users/pradal/...']
```

Standard modules

See Python Doc (> 100)

Python – Packages

Definition

Directory of Python module

Structure

```
import aa.bb
aa/
   __init__.py
   bb.py
   dd.py
   cc/
      __init__.py
      ee.py
from package import item
```

Importing *

```
__init__.py
__all__ = ['bb', 'dd']
```

Intra package references

```
from . import ee
from .. import bb
from ...aa import dd
```

Pkg in multiple directory

__path__ : list of dirs

```
openalea.core
openalea.visualea
openalea.sconsx
```

```
__init__.py
import pkgutil
__path__ = pkgutil.extend_path(__path__,
                               __name__)
```

Python – Input & Output

Définition

```
print or write in file
```

Format

```
1. Operation on string
2. % operator ⇔ sprintf
%d: int, %f: float, %s:string
'%s %d %f' % tuple
>>> "%s:(%d,%f)" % ("v",1,2)
'name=(1,2.0)'
```

Repr() vs Str()

```
str -> convert to string
repr -> string representation
>>> s= 'FSPM'
>>> str(s)
'FSPM'
>>> repr(s)
"'FSPM'"
```

Files

```
open (filename,x) x in (r,w,)
>>> f= open('toto.txt','w')
>>> fr= open('toto.txt')

readline()
>>> fr.readline()
'this is the first line.\n'
>>> f.write('this is a test')

readlines() -> [lines]
>>> fr.readlines()
['1st line.\n', '2nd line\n']
>>> f.close()
```

Pickle

```
Python object <-> string
>>> pickle.dump(x,f)
>>> x= pickle.load(f)
```

NumPy, Matplotlib, Pandas

SCIENTIFIC PYTHON

NumPy: numerical data

NumPy provides (https://numpy.org/)

- **extension package** to Python for multi-dimensional arrays
- closer to hardware (efficiency)
- designed for scientific computation (convenience)
- Also known as array oriented computing

NumPy: n-dimensional arrays

Import convention

```
>>> import numpy as np
```

Creation

1D

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

2D, 3D, …

```
# 2 x 3 array
>>> b = np.array([[0, 1, 2],
                  [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2,3)
```

operators on array

+,-,*,** : elementwise op.

```
>>> a**2
0,1,4,9
```

Creation with functions

arange

```
>>> np.arange(10)
```

linspace: start, end, num

```
>>> np.linspace(0, 1, 10)
>>> np.linspace(0, 1, 10,
endpoint=False)
```

random : random numbers

Uniform : rand

```
>>> np.random.rand(4)
array([0.10532385, 0.41072239,
0.53389495, 0.48056411])
```

Gaussian: randn

```
>>> np.random.randn(4)
array([-1.15661474, -0.22848969, -
0.14291618, -1.02244821])
```

Seed: `np.random.seed(64)`

Matplotlib: 2D plotting

Import convention

```
>>> import matplotlib.pyplot as plt
```

Jupyter Notebook

```
>>> %matplotlib inline
```

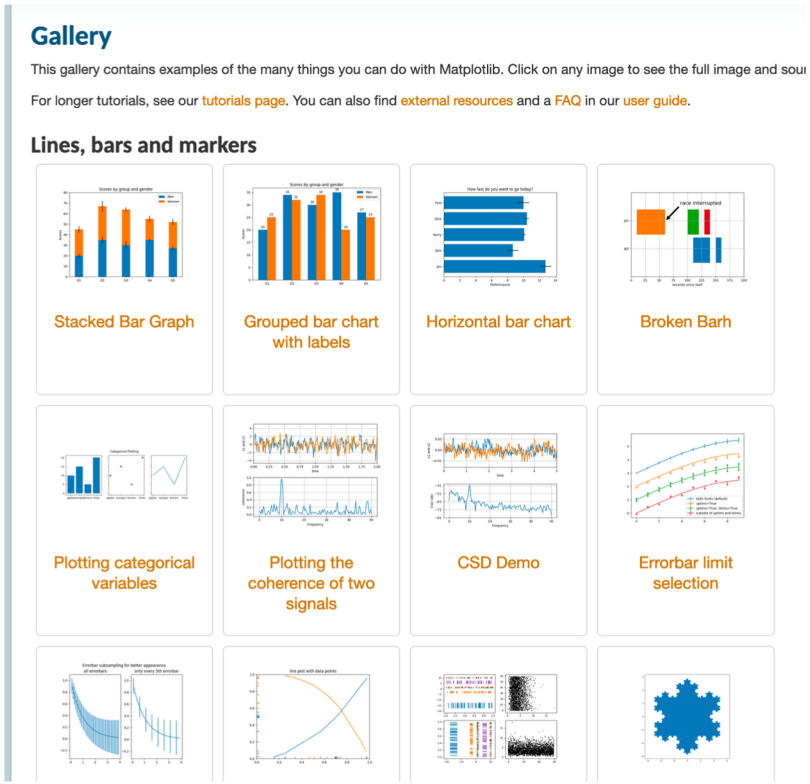
Simple Plot

```
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0, 3, 20)
>>> y = np.linspace(0, 6, 20)

# line plot
>>> plt.plot(x, y)

# dot plot
>>> plt.plot(x, y, 'o')
```

More complex stuff

Look at the matplotlib gallery:

https://matplotlib.org/3.1.1/gallery

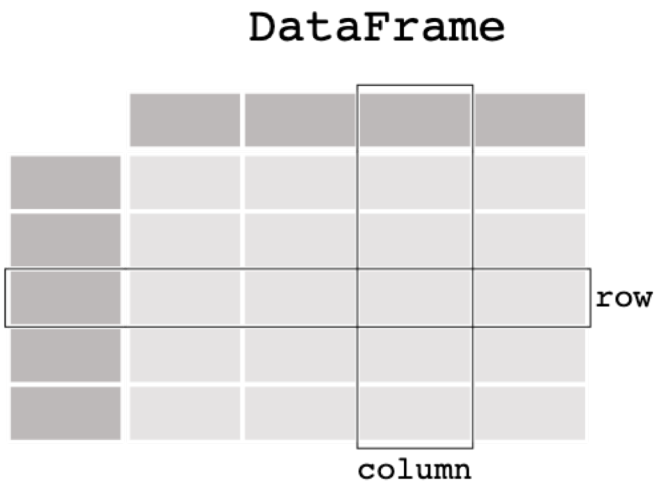


Pandas: tabular data analysis

Import convention

```
>>> import pandas as pd
```

DataFrame



WebSite

https://pandas.pydata.org

Import / Export