Short introduction to Python for Matlab users

Claus Führer, Numerical Analysis, Matematikcentrum

Najmeh Abiri, Computational Biology, Fysikum

Python Compact - Part 1

What do we cover in this introduction lecture?

What is also interesting to know?

- Testing, Profiling
- Iterators

Preparation

Everywhere in this course: in files, in shells we start with

In [1]:
```
from scipy import *
from matplotlib.pyplot import *
```

Tools you will need in this course:

- Ipython
- Spyder
- jupyter-notebook

Elementary datatypes

Numeric Types

In [2]:
```
i = 1           #  an integer
x = 2.          # a float
z = y = 3.e-8
c = 1.-3        # automatic type conversion
d = 1.+3.j      # a complex number
```

Elementary datatypes

Strings

In [3]:
```
text = 'We learn Python'
speed = 'fast'
how = 'fast. ' if speed else 'slow. '   # note the conditioned assignment
now = text + ' ' + how                  # concatination by "+"
2 * now
```

Out[3]:
```
'We learn Python fast. We learn Python fast. '
```

Summary: Elementary Datatypes

- we saw integers, floats, complex numbers, Booleans and strings
- we saw different ways to assign values to a variable
- we saw concatination of strings
- we saw `print()` and `type()`

Program Flow: Block commands

The flow is controlled by loops, conditional statements and functions

In [4]:
```
for i in range(3):
    print(i)
    j = i**2
    print(j)
```

```
0
0
1
1
2
4
```

Mind the intendation!

Nested Loop:

In [5]:
```
for i in range(5):
    for j in range(i + 1):
        print( (i,j), end = ' ')
    print('\n')      # Makes a new line
```

```
(0, 0)

(1, 0) (1, 1)

(2, 0) (2, 1) (2, 2)

(3, 0) (3, 1) (3, 2) (3, 3)

(4, 0) (4, 1) (4, 2) (4, 3) (4, 4)
```

Conditional statement

In [6]:
```
b = False
if b :
    j = 2 * j
else:
    j = None
print(j)
```

```
None
```

Loop constructions: `break`, `else`

In [7]:
```
j = 30
for i in range(100):
    if i == j:
        break
else:
    raise Exception('j not found')
print('j found')
```

```
j found
```

Container types

Lists

In [8]:
```
a = [1,2,'aha']
print(a)
b = [1,a]
a[0]
print(a[0])
print(b[1][2])
```

```
[1, 2, 'aha']
1
aha
```

In [9]:
```
a+b
```

Out[9]:
```
[1, 2, 'aha', 1, [1, 2, 'aha']]
```

In [10]:
```
c = list(range(0,8))
print(c[-1])
print(c[-4:])
print(c[0:2]) # Note the "hotel booking" rule.
```

```
7
[4, 5, 6, 7]
[0, 1]
```

List comprehension

In [11]:
```
base_list = [5,9,22,38]

new_list = [l**2+1 for l in base_list]

other_list=[l+1 for l in base_list if l%2 == 0 ]

print(new_list,end=3 * '\n')

print(other_list)
```

```
[26, 82, 485, 1445]


[23, 39]
```

List Operations

note the *inplace* operations

In [12]:
```
L=[1,2,3,29,-10]
L.sort()
print('sorted list L=', L)
L.reverse()
print('reversed list L=',L)
L.append(70)
print('extended list L=', L)
L.pop()
print('last element removed L=',L)
```

```
sorted list L= [-10, 1, 2, 3, 29]
reversed list L= [29, 3, 2, 1, -10]
extended list L= [29, 3, 2, 1, -10, 70]
last element removed L= [29, 3, 2, 1, -10]
```

Summary

- Lists and lists of lists
- indexing and slices
- negative indexing
- 'Hotel'booking rule
- List comprehension
- `list(range(...))`

Dictionaries

In [13]:
```
truck={'bodies':['frWheel','reWheel','Chassis','Cabine','Load'],
       'total length':15,'total weight':30000}
```

In [14]:
```
truck.keys()
```

Out[14]:
```
dict_keys(['total length', 'bodies', 'total weight'])
```

Tuples

In [15]:
```
a=2,3
print(a)
b=('anna',23)
print(b)
print(a+b)
print(b[1])
```

```
(2, 3)
('anna', 23)
(2, 3, 'anna', 23)
23
```

A tuple is an immutable list. What means immutable?

In [16]:
```
b[1]=-2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-74c169ca2cd8> in <module>()
----> 1 b[1]=-2

TypeError: 'tuple' object does not support item assignment
```

Warning: Changing in sub list has side effects!

In [17]:
```
a=[1,2,3]
b=[4,5,a]
print(b)
a[2]=-300
print(b)
```

```
[4, 5, [1, 2, 3]]
[4, 5, [1, 2, -300]]
```

In [18]:
```
b=tuple(b)
print(b)
a[2]=70
print(b)
```

```
(4, 5, [1, 2, -300])
(4, 5, [1, 2, 70])
```

In [19]:
```
b=[1,2]
a=b
b[1]=-234
print(a[1])
```

```
-234
```

Copy removes these side effects

In [20]:
```
a=b.copy()
b[1]=1
a[1]
```

Out[20]:
```
-234
```

Unpacking , Packing

In [21]:
```
a=1
b=b
c,d=a,b
```

or even

In [22]:
```
b,a = a,b  # the swapping trick
```

Summary:

- lists, list indexing, slices
- dictionaries, keys, values, items
- tuples, immutable
- copy
- packing, unpacking

Functions

In [23]:
```
def my_func(p1, p2, p3 = None):
    a = p1*p2 + p3 if p3 != None else p1*p2
    b = p1**2
    return a, b
```

In [24]:
```
r1, r2 = my_func(1, 2, 5)
```

`lambda` Functions: Defining functions on the fly

In [25]:
```
f = lambda x: sin(x) * cos(x)

f(pi)
```

Out[25]:
```
-1.2246467991473532e-16
```

Passing arguments

- by position
- by key
- when mixing: **always** first those by position

In [26]:
```
def my_func(p1, p2 ,p3):
    pass                    # a statement which does nothing, just to pleas
e the syntax
```

In [27]:
```
my_func(2, 3, 25)
my_func(p3 = 25, p1 = 2, p2 = 3)     # note, how default values are defined
my_func(2, 3, p3 = 25)
```

Starred Arguments

You can expand a list to form positional arguments and a dictionary to form keyword arguments of a function

In [28]:
```
li = [2, 3, 25]
di={'p1':2, 'p2':3, 'p3':25}
my_func(*li)
my_func(**di)
```

Scope of a variable

In [29]:
```
a = 3; l=[1, 2, 3]
def func1(b):
    return a * b   # a is a global variable

def func2(b):
    a = 17         # a is a local variable
    return a * b
```

In [30]:
```
print(func1(3))
a = -10              # Be aware of side effects
print(func2(3))
print(a)
```

```
9
51
-10
```

Arrays and Linear Algebra

In [31]:
```
from scipy import *
from scipy.linalg import *
```

a vector ...

In [32]:
```
v = array([1.,2.,3.])
w = array([1.,0.,1])
v.reshape((3,))
```

Out[32]:
```
array([1., 2., 3.])
```

Dot product, cross product, outer product

In [33]:
```
a = v @ w  #  or  a=dot(v,w)
A = outer(v,w)
u = cross(v,w)
```

Filling vectors

In [34]:
```
v = linspace(0,3,10)  # filling with equidistant floats
i = arange(0,3)  # filling with integers
o = ones((3,))
z = zeros((3,))
```

Matrices are two dimensional arrays

In [35]:
```
A = array([[1.,2.,7],[7,9,13],[-1,2,6.]])
A.shape
```

Out[35]:
```
(3, 3)
```

a particular element is adressed by a **dubble index**

In [36]:
```
trace = A[0,0] + A[1,1] + A[2,2]
print(trace)
```

```
16.0
```

the same but with list comprehension:

In [37]:
```
trace = sum( [A[i,i] for i in range(A.shape[0])] )
print(trace)
```

```
16.0
```

Filling 2D arrays

In [38]:
```
I = eye(3)
Z = zeros((3,3))        # the argument is a tuple
Z = zeros_like(I)
F = 5. * ones((3,3))    # the argument is a tuple
R = rand(3,3)           # the argument is *not* a tuple
a = pi/3
Rot = array([[cos(a), sin(a)],
             [-sin(a),cos(a)]])
```

Operations on arrays

In [39]:
```
F @ Z
v  = array([3,4])
vr = Rot @ v

from scipy.linalg import solve

abs(solve(Rot,vr) - v) < 1.e-8  # mind round-off
```

Out[39]:
```
array([ True,  True])
```

Views

In [40]:
```
f = arange(0, 10)
F = f.reshape(2, 5)
FT = F.T           # the transpose
F[-1,-1] = 17
print(FT[-1, -1], f[-1]) # everything affected
```

```
17 17
```

Universal functions

Many functions operate directly on arrays -- componentwise

In [41]:
```
sin(F)
abs(F)
```

Out[41]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8, 17]])
```

Addition, multiplication, division are **elementwise** operations on arrays

In [42]:
```
print(F)
print(F + F)
print(F * F)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8 17]]
[[ 0  2  4  6  8]
 [10 12 14 16 34]]
[[  0   1   4   9  16]
 [ 25  36  49  64 289]]
```

Array Examples

A projection matrix P and its eigenvalues and eigenvectors

$$P = \mathrm{I} - \frac{vv^T}{v^T v}$$

In [43]:
```
from scipy.linalg import *

v = array([1., 0., 0., 1., 5.])
P = eye(v.shape[0]) - outer(v, v)/(v @ v)

evalue, evector = eig(P)
print('P v =', P @ v, end=2*'\n')
print('Eigenvalues: ',evalue,
      'Eigenvectors: ', evector,sep=2*'\n')
```

```
P v = [2.22044605e-16 0.00000000e+00 0.00000000e+00 2.22044605e-16
 0.00000000e+00]

Eigenvalues:

[1.+0.j 0.+0.j 1.+0.j 1.+0.j 1.+0.j]

Eigenvectors:

[[ 0.98130676 -0.19245009  0.22187058  0.          0.        ]
 [ 0.          0.          0.          1.          0.        ]
 [ 0.          0.          0.          0.          1.        ]
 [-0.03774257 -0.19245009 -0.9637218   0.          0.        ]
 [-0.18871284 -0.96225045  0.14837025  0.          0.        ]]
```

In [44]:
```
for i in range(evalue.shape[0]):
    if allclose(evector[:,i], -v/norm(v,2)) or allclose(evector[:,i],-v/norm
(v,2)):
        print('v is an eigenvector of the projection matrix corresponding to
eigenvalue ',evalue[i].real)
```

```
v is an eigenvector of the projection matrix corresponding to eigenvalue  0.0
```

Boolean Arrays

In [45]:
```
A = rand(4,4)
small = abs(A) < 0.5

C = A.copy()
C[small] = 0
print(A,C,small,sep=2*'\n')
```

```
[[0.79513052 0.2886361  0.09312346 0.39255937]
 [0.36544745 0.09243608 0.46101068 0.18904672]
 [0.03383226 0.39205689 0.04890219 0.57672891]
 [0.42758111 0.02100887 0.20230901 0.3542879 ]]

[[0.79513052 0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.57672891]
 [0.         0.         0.         0.        ]]

[[False  True  True  True]
 [ True  True  True  True]
 [ True  True  True False]
 [ True  True  True  True]]
```

Continuation from eigenvalue example above

(note the decorator `vectorize` to make a universal function)

In [46]:
```
@vectorize
def near(x, y, eps = 1.e-8):
    return (y-eps) <  x.real < (y+eps) and -eps < x.imag < eps

if (near(abs(evalue),1) + near(evalue,0)).all():
    print('All eigenvalues of a symmetric projection matrix are either +- 1
or 0')
```

```
All eigenvalues of a symmetric projection matrix are either +- 1 or 0
```

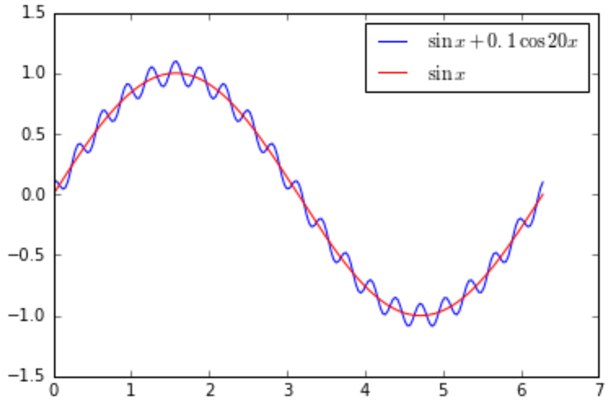
Elementary Plotting

write in IPython as one of the first commands `%matplotlib` when you want to want the plots to be embedded in the current session.

In [47]:
```
%matplotlib inline
```

In [48]:
```
x = linspace(0, 2 * pi, 200)
plot(x, sin(x) + 0.1*cos(20*x), label='$\sin\,x+0.1\,\cos\,20x$')
plot(x, sin(x), 'r', label='$\sin\,x$')
legend(loc = 'best')
```

Out[48]: <matplotlib.legend.Legend at 0x7f7f88d089b0>



Exceptions

In [49]:
```
x_list = [1, 2, 3, 0, 23, 65]
y = []

for x in x_list:
    try:
        y.append(1./x)
    except ZeroDivisionError:
        y.append('no inverse')

print(x_list, y, sep='\n')
```

```
[1, 2, 3, 0, 23, 65]
[1.0, 0.5, 0.3333333333333333, 'no inverse', 0.043478260869565216, 0.01538461
5384615385]
```

Important Exception Types

- IndexError
- ValueError
- NameError
- ZeroDivisionError
- SyntaxError

... or just unspecified.

Task:

1. Generate a vector with random numbers. Check all its elements and count the number of sign changes.
2. write a function which takes an axis-label ('x', 'y', 'z') and an angle as input and returns a rotation matrix, e.g., `rot3D(2*pi,'x')` should return a 3D rotation matrix about the 'x'-axis.
3. Test if the product of three rotation matrices is orthogonal and has determinant 1
4. Compute the eigenvalues of this composed rotation matrix and plot them as small stars in the complex plane.