Hands on Pyhton

Carlos Hernández-García, Luis Plaja

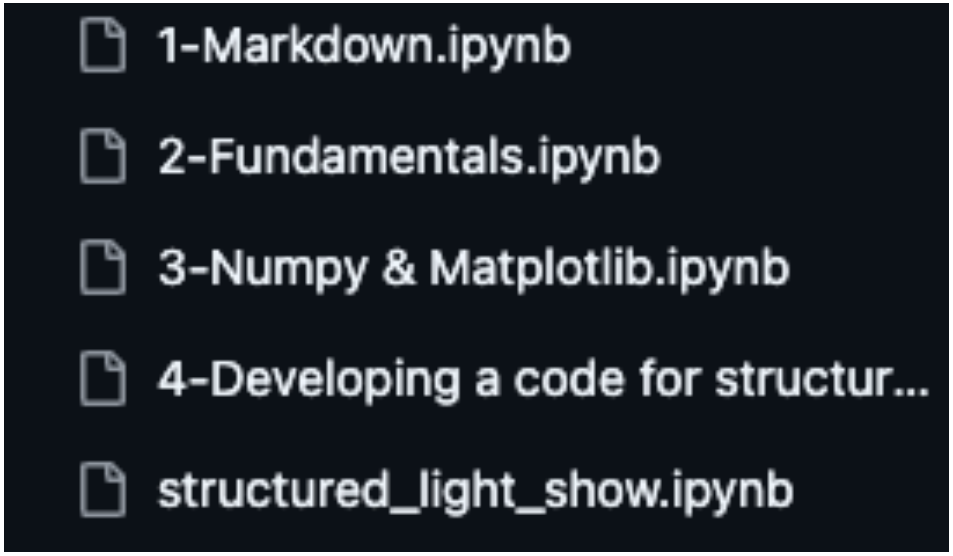
VNiVERSiDAD Đ SALAMANCA
Unidad de Excelencia en
Luz y Materia Estructuradas (LUMES)

ALF USAL

GitHub repository

chrome

https://lplaja.github.io/structured_light/lab/index.html



Introduction to markdown

Fundamentals of Pyhton

Numpy and Matplotlib

Your structured light code

A tool for this week (and beyond..)

First a poll

Who has some previous knowledge on Python coding? (No no need of an intro to Python)

…. go directly to

Developing a code for structured light.

Crash course on python

Almost everything in Python is an object

Objects: a minimal introduction

pneumatics

steering wheel

lights

seats

Attributes (properties)

accelerate

brake

turn riht or left

go backwards

Methods (functionalities)

Objects: a minimal introduction

Class definition and object instantiation

```
class car:
    def __init__(self,pneumatics_list):
        self.pneumatics=pneumatics_list

    def accelerate(self):
        print('Brum, brum, bruuuuum!!!')
```

Class definition (General car structure)

The __init__ method allows to create objects from the class by setting the attributes

Creation of a particular object of the class (object instantiation)

```
# Instantiate an object of class "car" specifying the models of the four pneumatics
mycar=car(['firestone10M6','firestone10M6','continentalvcm','continentalvcm'])
```

Objects: a minimal introduction

Using an object

Access to the attributes:

```
# What pneumatics uses my car?
print('My car has the following phenumatics: ',mycar.pneumatics)
```

Use the methods:

```
mycar.accelerate()
```

Objects: a minimal introduction

Some pyhton objects:

- Numbers:

```
[11]
1  i=1
2  print(type(i))

··  <class 'int'>
```

Despite being objects, we treat them as in any other language:

```
i=-10
print(abs(i)) # abs() brings the absolute value
```

- characters, strings…

Containers

Other python objects are very interesting structures. They are called *containers* , because the contain *things* (other objects, numbers, letters, functions… anything)

Among them, there are **two** important ones

Lists

```
ex=[0.2, 3, 4.5,'+','+', 0.3, -4, 8, '-', '-', 1]
```

List construction

```
print(ex[0],ex[2]) # the first (position 0) and third data (position 2)
print(ex[2:5])     # data from position 2 to 4, 5 is not inluded
print(ex[2:])      # from position 2 to the final of the list
print(ex[:5])      # from the beggining to position 4
print(ex[-1])      # last element
print(ex[3::2])    # form position 3 to the end in steps of two
```

Accessing the elements

Containers

Lists

Important things to remember

- List are not arrays (mathematical objects composed with numbers)

```
ex1=[0.2,3,4.5,'+','+',0.3,-4,8, '-', '-', 1]
ex2=[3,2,1,0]
print(ex1+ex2)  # concatenates strings
print(2*ex2)    # concatenates a string with itself

# [0.2, 3, 4.5, '+', '+', 0.3, -4, 8, '-', '-', 1, 3, 2, 1, 0]
# [3, 2, 1, 0, 3, 2, 1, 0]
```

Containers

Lists

Important things to remember

- There are no two- three- dimensional lists. Only 1D

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

- However you can define a list with aother lists as elements

```
a=[[1,-1],2,3,4]
```

Containers

Dictionaries

Dictionaries contain a series of pairs “key”/value

Key : Value

```
vehicles={'car':'a vehicle moving on wheels',
          'bus':'a large motor vehicle designed to carry passengers',
          'truck':'a wheeled vehicle for moving heavy articles'}


  Prices={‘oranges':5,
           ‘bananas’:4,
           ‘apples’:3.5}
```

Containers

Looping over the elements of a container

Looping over a list

You don't have to use an index: `For n=1 to 10`

```
ex1=[0.2,3,4.5,'+','+',0.3,-4,8, '-', '-', 1]
for n in ex1:  # n is not an index, loops directly the values stored in the list
    print(n)
```

It is possible to enumerate the list

```
ex1=[0.2,3,4.5,'+','+',0.3,-4,8, '-', '-', 1]
for i, n in enumerate(ex1):
    print(i, n)
```

Containers

Looping over the elements of a container

Looping over a dictionary

You don't have to use an index: `For n=1 to 10`

```
ex1=[0.2,3,4.5,'+','+',0.3,-4,8, '-', '-', 1]
for n in ex1:  # n is not an index, loops directly the values stored in the list
    print(n)
```

It is possible to enumerate the list

```
ex1=[0.2,3,4.5,'+','+',0.3,-4,8, '-', '-', 1]
for i, n in enumerate(ex1):
    print(i, n)
```

Functions

```
def my_function(a,b):
    type_a=type(a)
    type_b=type(b)
    result=a+b
    return result, type_a, type_b

a=2; b=3.2
r, ta, tb= my_function(a,b)
print('The first paramater is of type ', ta, ', the second is o type ',tb,'
and their sum is ', r )

a=[2,2]; b=[3.2,1]
r, ta, tb= my_function(a,b)
print('The first paramater is of type ', ta, ', the second is o type ',tb,'
and their sum is ', r )

a='first'; b='second'
r, ta, tb= my_function(a,b)
print('The first paramater is of type ', ta, ', the second is o type ',tb,'
and their sum is ', r )
```

Mathematical Python: Numpy

To make Python understand advanced mathematics we need to wempower it loading the **module numpy**.

```
import numpy as np
```

A module usually is a library of functions and constants. In Python a module is an **object**.

Mathematical constants are attributes

```
print(np.pi)
print(np.e)
```

Mathematical functions are methods

```
print(np.cos(np.pi))
```

Numpy arrays

Lists can be converted to arrays.

- An array is a mathematical object (you can do mathematical operations)

```
my_array1=np.array([1,3,5,7]); print(my_array1)          [1 3 5 7]
my_array2=np.array([2,4,6,8]); print(my_array1)          [2 4 6 8]

my_array3=my_array1+my_array2; print(my_array3)          [ 3  7 11 15]
```

- Multidimensional arrays are constructed from lists os lists

```
my_array2D=np.array([[1,2,3], [4,5,6], [7,8,9]])

print(my_array2D)                              [[1 2 3]
                                                [4 5 6]
                                                [7 8 9]]
```

Grids

A grid is an array of equally spaced incrementing/decrementing numbers. We use them as coordinates for plots.

- 1D grids

```
x=np.linspace(0, 2*np.pi, 10)
        [0.         0.6981317  1.3962634  2.0943951  2.7925268  3.4906585
         4.1887902  4.88692191 5.58505361 6.28318531]

my_sin=np.sin(x); print(my_sin)
        [ 0.00000000e+00  6.42787610e-01  9.84807753e-01  8.66025404e-01
          3.42020143e-01 -3.42020143e-01 -8.66025404e-01 -9.84807753e-01
         -6.42787610e-01 -2.44929360e-16]
```

Grids

A grid is an array of equally spaced incrementing/decrementing numbers. We use them as coordinates for plots.

- 2D grids (meshgrids)

```
x=np.linspace(0,4,4, endpoint=False); print('x=', x) #1d mesh of 4 elements
y=np.linspace(0,3,3, endpoint=False); print('y=', y)  #1d mesh of 2
elements

X,Y=np.meshgrid(x,y); # X and Y compose a 2D mesh of 2x4 elements

Z=X**2+Y**2                          x= [0. 1. 2. 3.]   y= [0. 1. 2.]

print('Z=',Z)                        Z= [[ 0.  1.  4.  9.]
                                         [ 1.  2.  5. 10.]
                                         [ 4.  5.  8. 13.]]
```

Plotting:Matplotlib.pyplot

We can expand Python to make plots from numpy arrays. For this, we have to import de module

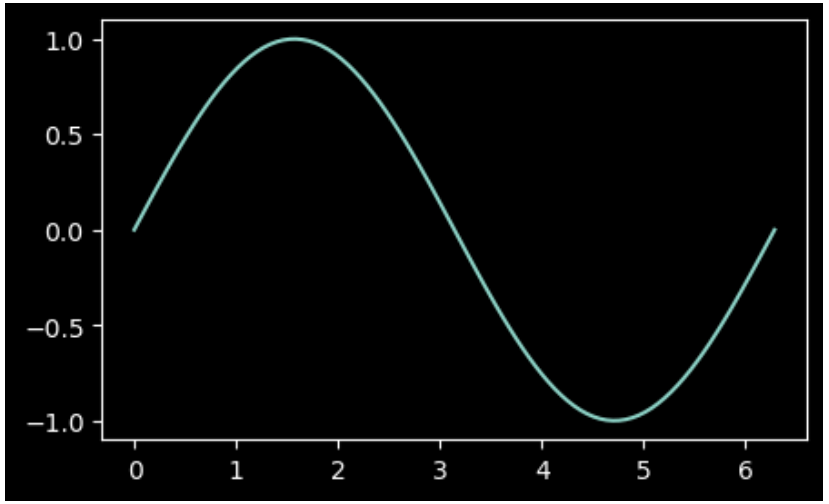
```
import matplotlib.pyplot as plt
```

- 1D plots

```
x=np.linspace(0, 2*np.pi, 100)
my_sin=np.sin(x); print(my_sin)

fig,ax=plt.subplots(1,1,figsize=(5,3))

ax.plot(x, my_sin)
```



Plotting:Matplotlib.pyplot

- 2D plots

```
import matplotlib.pyplot as plt
from matplotlib.colors import LightSource

x=np.linspace(0,4,200)
y=np.linspace(0,3,200)

X,Y=np.meshgrid(x,y)
Z=np.sin(X**2+Y**2)

fig = plt.figure(figsize=(6,6))
ax = plt.axes(projection='3d')

ls = LightSource(azdeg=0, altdeg=65)
# Shade data, creating an rgb array.
rgb = ls.shade(Z, plt.cm.RdYlBu)

ax.plot_surface(X, Y, Z)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                       antialiased=False, facecolors=rgb, linewidth=0)
```