ME 213

Python
for
C & Matlab programmer

- a VERY brief introduction –
- Inspired by many (see next slide) -

Some references

- https://www.w3schools.com/python/
- https://docs.python.org/3.7/index.html
- https://docs.python.org/3.7/tutorial/index.html
- https://en.wikipedia.org/wiki/Python_(programming_language)
- https://realpython.com/
- https://www.lancaster.ac.uk/staff/drummonn/PHYS281/demo-data-types/

You know matlab and C

-> you won’t be lost in python

You need to learn the syntaxe and other idiosyncrasy

What is python

- Python is an interpreted textual language
- CPython is the reference implementation of the Python programming language, it is written in C and Python.
- Pypy and conda are other python implementations.
- Python (CPython) is compiled as bytecode and then interpreted
- Compared to C, Python is very-high-level language, similar to Matlab
- Python can be treated in a procedural way, an object-oriented way or a functional way
- Python can be extended with other language, ex. C,

  it can also be embedded in other environments like Matlab or LabVIEW
- The original author is Guido van Rossum, initial release 1994

Plan – covers some* aspects of

- Interactive python
- Variables
- Types
- Loop, if, etc.
- Functions
- Modules
- other

** very far from being exhaustive*

Python – interactive mode

Interactive mode similar to matlab command window

```
% python3                    Launch python in the terminal

Python 3.10.11 (v3.10.11:7d4cc5aa85, Apr  4 2023, 19:05:19) [Clang 13.0.0 (clang-1300.0.29.30)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> print("Hello, World!")
Hello, World!

>>> 3+2                       Simple command
5

>>> a=4                       Declare a variable  and assign 4 to a
>>> a
4

>>> b=6
>>> a+b                       Simple calculation
10
```

Python – built-in types

Variables have type

```
>>> type(b)
<class 'int'>

>>> type(3.14)
<class 'float'>

>>> s = 'abc'
>>> s
'abc'

>>> z = complex(1, 2)
>>> z
(1+2j)

>>> print(s)
abc
```

Ask for variable type

Numbers also have type

String can be delimited with ‘ or “

Use print() to display variable value

More

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

More types will be available via NumPy

Python – variable assignation

Variables have type

```
>>> a = 4          a is an int
>>> type(a)
<class 'int'>

>>> a = "bcd"      a is now a str
>>> type(a)
<class 'str'>


>>> x = int(1)     force a type to int
>>> type(x)
<class 'int'>

>>> x = str(1)     force a type to str
>>> x
'1'
```

```
>>> x = float(1)
>>> x
1.0

>>> x = bool(1)          0 -> false, other is '1'
>>> x
True


>>> del(s)                    Remove a var
>>> s
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 's' is not defined
```

Python – multiple assignations

You can assign multiple variables at the same time

```
>>> a, b, c  =  11, 12, 13
>>> a
11
>>> b
12
>>> c
13


>>> a = b = c = 11
>>> a
11
>>> b
11
>>> c
11
```

```
>>> y = [11, 22, 33]
>>> y
[11, 22, 33]

>>> type(y)
<class 'list'>


>>> s = 3 * 'x'
>>> s
'xxx'
```

y is a list

One can do funny things in python

Python – variable scope

Similar to Matlab

This is a function
(next slide)

```
x = 300
def myfunc():
  x = 200

>>> myfunc()

>>> x
300
```

this x is global

*this is **another x local** to myfunc()*

```
x = 300
def myfunc():
  global x
  x = 200

>>> myfunc()

>>> x
200
```

this x is global

make var global as in ML

*this is **global x***

Python – compound types - collections

Collections are similar array in C but can contain different data types like cell array in Matlab

*There is 4 types of collections: **List**, **Tuple**, **Set**, **Dictionaries**, each types has its own set of characteristics*

	Sep.	Ordered	Indexed	Dup. values	Changeable	Analogy
List	**[ ]**	V	V	V	V	*ML cell array*
Tuple	**( )**	V	V	V	X	*ML cell array*
Set	**{ }**	X	X	X	X*	*bag*
Dictionaries	**{k:v}**	V	X	X	V	*DB / JSON*

Python – list

List is similar to array/cell array

```
>>> mylist = [1, 2, 'a', 3.14]
>>> mylist
[1, 2, 'a', 3.14]

>>> mylist[0]
1

>>> mylist[2]
'a'

>>> mylist[-1]
3.14

>>> mylist[1:3]
[2, 'a']
```

*array of **mixed** types*

*'-' -> from the **end-1***

*range **begin : end-1***

Python – list

List is similar to array/cell array

```
>>> mylist[2] = 5                    modify item at position 2
>>> mylist
[1, 2, 5, 3.14]


>>> mylist[4] = 8                    Δ array expansion not like Matlab!!
Traceback (most recent call last):
  File "<python-input-17>", line 1, in <module>
    mylist[4]=8


>>> mylist.append(8)                 Add '8' at the end
>>> mylist
[1, 2, 5, 3.14, 8]
```

Python – list

List is similar to array/cell array

```
>>> len(mylist)
5
```

list knows its size

```
>>> del(mylist[2])
>>> mylist
[1, 2, 3.14, 8]
```

remove item at position x

```
>>> mylist.remove(8)
>>> mylist
[1, 2, 3.14]
```

remove item which as value v
If there are more than one item with the specified value, the remove() method removes the first occurrence

```
>>> mylist.pop()
3.14
```

remove last item

Python – list

List is similar to array/cell array

```
>>> mylist = [1, 2, 3.14]

>>> mylist2 = mylist
>>> mylist2
[1, 2, 3.14]

>>> mylist[0] = 0
>>> mylist2
[0, 2, 3.14]

>>> mylist3 = mylist.copy()
>>> mylist[0] = -3
>>> mylist3
[0, 2, 3.14]

>>> mylist4 = list(mylist)
>>> mylist5 = mylist[:]
```

*copy **a *reference*** to the **mylist***
like pointer copy in C !

*copy the list ***items***
In C you should use a loop to copy items one by one

other ways to copy the list items

Python – list methods

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python – dictionnary

Dictionaries are used to store data values in ***key : value*** *pairs*

```
>>> myDict= { 'name':'John',
              'age': 3,
              'color':'blue' }

>>> myDict
{'name': 'John', 'age': 3, 'color': 'blue'}


>>> myDict['name']                    Get value from key
'John'

>>> myDict.get('name')                Get value from key, method way
'John'


>>> myDict.keys()                     Get all keys
dict_keys(['name', 'age', 'color'])
```

Python – dictionnary

Dictionaries are used to store data values in ***key : value*** *pairs*

```
>>> myDict['size'] = 'M'
>>> myDict
{'name': 'Jane', 'age': 3, 'color': 'blue', 'size': 'M'}

>>> myDict['size'] = 'L'
>>> myDict
{'name': 'Jane', 'age': 3, 'color': 'blue', 'size': 'L'}

>>> myDict.pop('color')
'blue'
>>> myDict
{'name': 'Jane', 'age': 3, 'size': 'L'}

>>> del myDict['age']
{'name': 'Jane', 'size': 'L'}
```

add a new ***key:value*** *pair if key does not exist*

modify a ***key:value*** *pair if key exist*

remove an existing ***key:value*** *pair*

remove an existing ***key:value*** *pair*

Python – nested dictionnaries

Dictionaries are used to store data values in ***key : value*** *pairs*

```
>>> myFriends = { 'friend1' : {'name': 'Jane', 'age' : 3},
                  'friend2' : {'name': 'Jule', 'age' : 4}}
>>> myFriends
{'friend1': {'name': 'Jane', 'age': 3}, 'friend2': {'name':
'Jule', 'age': 4}, 'friend3': {'name': 'Jimy', 'age': 3}}

>>> myFriends['friend2']['name']
'Jule'

>>> friend3 = {'name': 'Jimy', 'age': 3}

>>> myFriends.update({'friend3':friend3})

>>> myFriends
{'friend1': {'name': 'Jane', 'age': 3}, 'friend2': {'name':
'Jule', 'age': 4}, 'friend3': {'name': 'Jimy', 'age': 3}}
```

access nested ***value***

create a new ***key:value*** *pair*

add it to the existing dictionary

Python – dictionary methods

clear()	Removes all the elements from the dictionary
copy()*	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Similarly to LIST, myDict2 = myDict copies **a *reference* to **myDict ->** use the **copy()** method to copy key:value pairs into myDict2*

Python – basic arithmetic operations

Very similar to C and Matlab

```
>>> # this is a comment
>>> 2 ** 3   # ^ is XOR as in C
8
>>> 8 % 3    # % is modulo as in C
2
>>> 9 // 2   # // is the integer division
4
>>> 3 != 4   # != is ≠ as in C
True

>>> pi       # not defined here
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
```

Python – other arithmetic operations

Many arithmetic functions are available via math (and other) module

```
>>> import math    # load the math library as in C

>>> math.pi        # now we have pi ☺
3.141592653589793

>>> math.sqrt(49)
7.0

>>> math.sin(math.pi)
1.2246467991473532e-16        numeric precision


>>> dir(math)                          list all functions in a module
['__doc__', …, 'acos', 'acosh', … 'tanh', 'tau', 'trunc', 'ulp']
```

Python – flow control – indenting and :

A bloc of instructions/statements is defined by its indentation, generally 4 spaces

Failing to indent your code will result in an error

```
a = 3

if a > 3 :                  # : is required
    a = 2 * a
    b = a - 1
    c = 0
    for d in range(b):      # : is required
        c = c + d
        if c < 10:          # : is required
            print(c)
        a = c

e = 3 * a
```

parenthesis are not needed

';' not needed

*no **end** as in Matlab*

Python – flow control - if

Same as in C or Matlab

Conditions expression are the same as in C

```
a = 33
b = 200
c = 0

if b > a:
  c = a
elif a == b:
  c = -2
else:
  c = b


# one line statement
# value_when_true if condition else value_when_false

print(a) if a > b else print(b)
```

*you can have as many **elif** as you like*

Python – flow control - for

For loops are a bit different than in C and are closer to Matlab usage
A for loop is used to increment though a sequence of values (list or tuple)
Much like matlab ***for n= (1:10)***

```
n = [1,2,4,8]  # n is a python list with 4 elements
sum = 0
for i in n:
  sum += n
print(sum)     # sum is 15



s = ['h','e','l','l','o']  # list with 5 char
wrd = ""
for c in s:
    wrd += c
print(wrd)     # wrd is hello
```

Python – flow control - for

range() *returns a sequence of values, much like vector in Matlab, (ML ex.* ***val = 1:2:5*** *and then* ***for n= val*** *…)*
By default the sequence ***starts from 0****, and* ***increments by 1*** *(by default), and* ***ends at a specified number -1***
An increment other than 1 can be specified

```
for x in range(3):          # print from 0 (default) to 3-1
  print(x)
```

```
0
1
2
```

```
for x in range(2,4):         # print from 2 to 4-1
  print(x)
```

```
2
3
```

```
for x in range(3,9,4):       # print from 3 to 9-1 with step 4
  print(x)
```

```
3
7
```

Python – flow control - for

break *and* ***continue*** *work like in C and Matlab*

```
for x in range(9,3,-4):     # print from 9 to 3-1 with step -4
  print(x)
```

```
9
5
```

```
for x in range(-2,2):       # print from -2 to 2-1
  if i >= 0:
    break                   # exit the loop if x ≥ 0
  print(x)
```

```
-2
-1
```

In some cases, ***iterator*** *can replace a for loop (see itertools)*

Python – flow control - While

while loops are similar to C and Matlab
***break** and **continue** work like in C and Matlab*

```
sum = 0
i = 0

while i < 10:
  sum += n
  i += 1                # no i++ in python

print(sum)


i = 1
while i < 6:
  print(i)
  i += 1
  if i == 3:
    break               # loop ends here
else:
  print("i >= 6")       # else … will NOT be executed since break occurred
```

Python – function

Similar to Matlab functions, usage **def** *functionName* **(** *parmeter* **) :**

```
def myFunc():       # no parameter, does not return value
  print("Hello world")

myFunc()


def myFunc2(a,b):   # 2 parameters, does not return value
  print("a+b=", a+b)

myFunc2(2,3)

myFunc2(2)          # will trigger an error, 1 param missign


def myFunc3(a,b):   # 2 parameters, returns 1 value
  return a+b

res = myFunc2(2,3)
```

Python – function

Variable number of parameters are similar to argv[] in C

```
def myFunc4(*myItems):                    # n parameters as tupple
  print("nb params", len(myItems))
  print("first item", myItems[0])


myFunc4("ab")
nb params 1
first item ab

myFunc4("ab","cd","ef")
nb params 3
first item ab

myFunc4()
nb params 0
IndexError: tuple index out of range
```

Python – function

*Pass parameters as **key = value,***
*** for a variable number of **key=value** pairs*

```
def myFunc5(first, last):
  print("range",first," - ",last)

myFunc5(last = 10, first = "aa")
range aa - 10



def myFunc6(**range):
  print("range", range["first"]," - ", range.get("middle"))

myFunc6(last = 10, middle ="mm", first = "aa")
```

Python – function

*Default value for parameter as **parameter = defaut value,***
Default value will be used is the parameter is missing

```
def myFunc7(name = "John"):
  print("Hello", name)


myFunc7("Jimmy")
Hello Jimmy


myFunc7()
Hello John
```

Python – function

Return value
Default value will be used is the parameter is missing

```
def myFunc7(name = "John"):
  print("Hello", name)


myFunc7("Jimmy")
Hello Jimmy


myFunc7()
Hello John
```

Python – function

*Add **,/** to force positional parameter*
*Add ***,** to force param as **name=value***

you can mix these two options

```
def myFunc8(name ,/):
  print("Hello", name)

myFunc8("Jimmy")
Hello Jimmy

myFunc8(name = "Jimmy")
TypeError: myFunc8() got some positional-only arguments passed as keyword arguments: 'name'


def myFunc9(*, name):
  print("Hello", name)

myFunc9(name="Jimmy")
Hello Jimmy

myFunc9("Jimmy")
TypeError: myFunc8() takes 0 positional arguments but 1 was given
```

Python – function

*Use **return** keyword to return one or more values*
return value(s) will adapt to the input type

```
def myFunc10(x):
  return 2 * x


ret = myFunc10(3)              # multiply by 2 an int
print(ret)
6

ret = myFunc10("ab")           # duplicate str
print(ret)
abab

ret = myFunc10([4, 5])         # duplicate list
print(ret)
[4, 5, 4, 5]
```

Python – function

*Use **return** keyword to return one or more values*
return value(s) will adapt to the input type

```
def myFunc11(x):
  return x**2, x**3

ret = myFunc11(3)
print(ret)
(9, 27)                          # this is a tuple containing 2 values

a,b = myFunc11(3)                # a and b are two int
print(a,b)
9 27                             # no ',' separator nor ()

_, c = myFunc10(3)               # ignore 1st returned value
print(c)
27

d = myFunc11(3)[0]               # get only the 1st return value
print(d)
9
```

Python – lambda function

lambda function are similar to Matlab anonymous function, format: **lambda** *arguments* **:** *expression*
lambda function can be used inside other function

```
myL = lambda a, b : a * b      # lambda function definition, its name/ref is myL
c = myL(5, 6)
print(c)
30
```

```
def myFunc12(n):
  return lambda a : a * n      # returns a lambda func. which multiply input by n


mul2 = myFunc12(2)             # mul2 is a lamdda func. which multiply input by 2
print(mul2(11))                # a is 11 and n is 2 in mul2
22

mul3 = myFunc12(3)             # mul3 is a lamdda func. which multiply input by 3
print(mul3(12))                # a is 12 and n is 3 in mul3
36
```

Python – Module

Python can be extended via module, similar to library in C or toolbox in Matlab
Python modules are text files with ***.py*** *extension*
Use the **import** *keyword to add a module,* **as** *defines an alias to the module*
To access a specific function, use **from** *module_name* **import** *function/var*

myModule.py

```
def mySquare(x):
  return x**2

def myCube(x):
  return x**2

myVar = 42
```

```
import myModule
print(myModule.mySquare(2))
4
```

```
import myModule as mm          # mm is an alias to myModule

print(mm.myCube(3))
27
print(mm.myVar)
42
```

Python – Module

Python modules can contain functions and variables but can also contain the "main" program similarly to C main() or as a script (vs function) file in Matlab and run as a stand-alone script
*Python set the variable __**name**__ to __**main**__ when calling the script directly not importing it as a module*

```
import myModule as mm

print(mm.mySquare(2))
4
print(mm.myName)
myModule
```

```
In the terminal

>> python3 myModule

name: __main__
demo: 3**2= 9
```

myModule.py

```
def mySquare(x):
  return x**2

myName = __name__

if __name__ == "__main__":
    print("name:", __name__ );
    print("demo: 3**2=",mySquare(3))
```

Use the following to add a path not in current python paths

```
import sys
sys.path.append('/path/to/myModule/')
import myModule
```

Python – Module

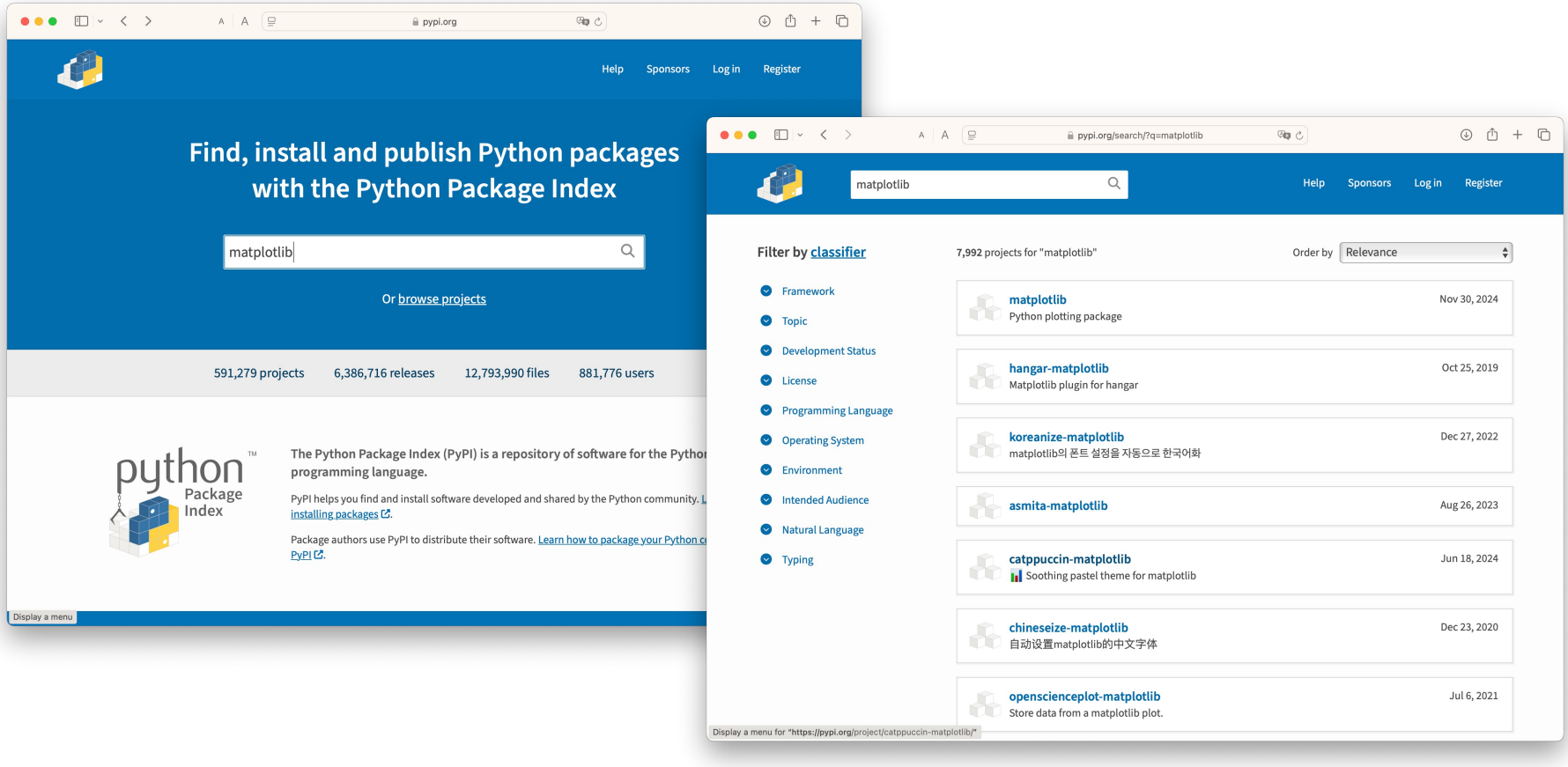
*There are *many" Python built-in modules*
Other are available after installing them on your machine

```
import platform as p          # platform is builtin

print(p.system())
Darwin                        # on Mac
print(p.machine())
arm64


dir(p)                        # list all function in a module
```

```
['AndroidVer', 'IOSVersionInfo', '_Processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES', '__builtins__', '__cached__',
'__copyright__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '__version__', '_comparable_version',
'_default_architecture', '_follow_symlinks', '_get_machine_win32', '_java_getprop', '_mac_ver_xml', '_node', '_norm_version',
'_os_release_cache', '_os_release_candidates', '_parse_os_release', '_platform', '_platform_cache', '_sys_version',
'_sys_version_cache', '_syscmd_file', '_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_stages', '_win32_ver', '_wmi',
'_wmi_query', 'android_ver', 'architecture', 'collections', 'freedesktop_os_release', 'functools', 'ios_ver', 'itertools',
'java_ver', 'libc_ver', 'mac_ver', 'machine', 'node', 'os', 'platform', 'processor', 'python_branch', 'python_build',
'python_compiler', 'python_implementation', 'python_revision', 'python_version', 'python_version_tuple', 're', 'release', 'sys',
'system', 'system_alias', 'uname', 'uname_result', 'version', 'win32_edition', 'win32_is_iot', 'win32_ver']
```

Python – Package

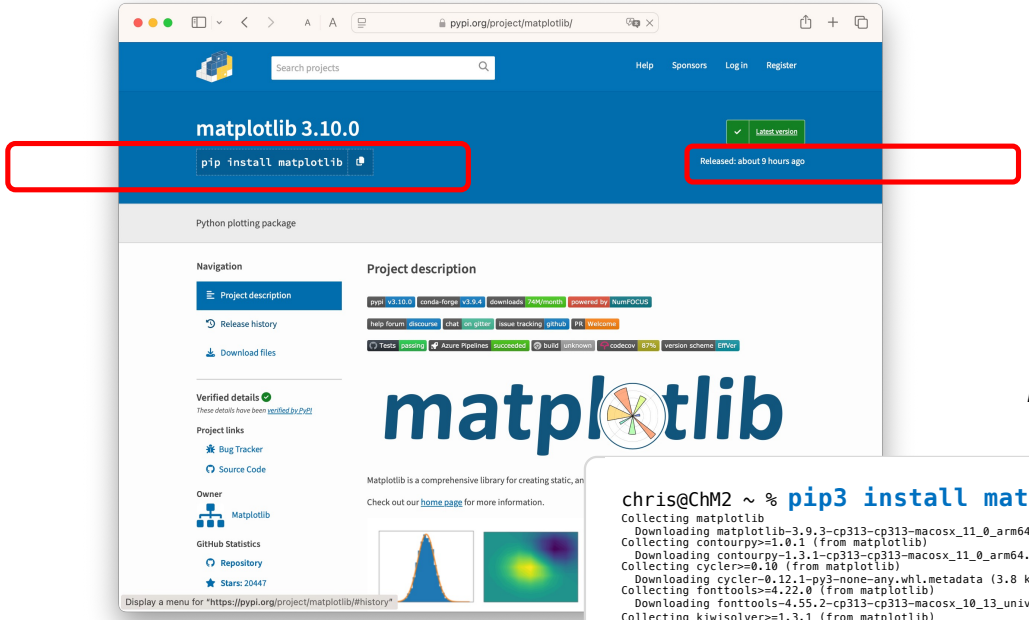
There are "zillions" of Python packages you can install on your machine
A package is a collection of modules that are organized in a directory hierarchy.
Pypi.org is a repository of packages, it has currently 592966 projects available



Python – Package

Browse pypi to find the package you want

The terminal command to install this package



I installed an "older" version just 2 days ago

***matplotlib** requires **11** other packages !!*

```
chris@ChM2 ~ % pip3 install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.9.3-cp313-cp313-macosx_11_0_arm64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.1-cp313-cp313-macosx_11_0_arm64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.55.2-cp313-cp313-macosx_10_13_universal2.whl.metadata (164 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.7-cp313-cp313-macosx_11_0_arm64.whl.metadata (6.3 kB)
…
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.5/13.5 MB 5.5 MB/s eta 0:00:00
Downloading packaging-24.2-py3-none-any.whl (65 kB)
Downloading pillow-11.0.0-cp313-cp313-macosx_11_0_arm64.whl (3.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 5.2 MB/s eta 0:00:00
Downloading pyparsing-3.2.0-py3-none-any.whl (106 kB)
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, pyparsing, pillow, packaging,
numpy, kiwisolver, fonttools, cycler, python-dateutil, contourpy,
matplotlib
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.55.2 kiwisolver-1.4.7 matplotlib-3.9.3 numpy-2.1.3
packaging-24.2 pillow-11.0.0 pyparsing-3.2.0 python-dateutil-2.9.0.post0 six-1.17.0
```

*Now you can do **`import matplotlib`***

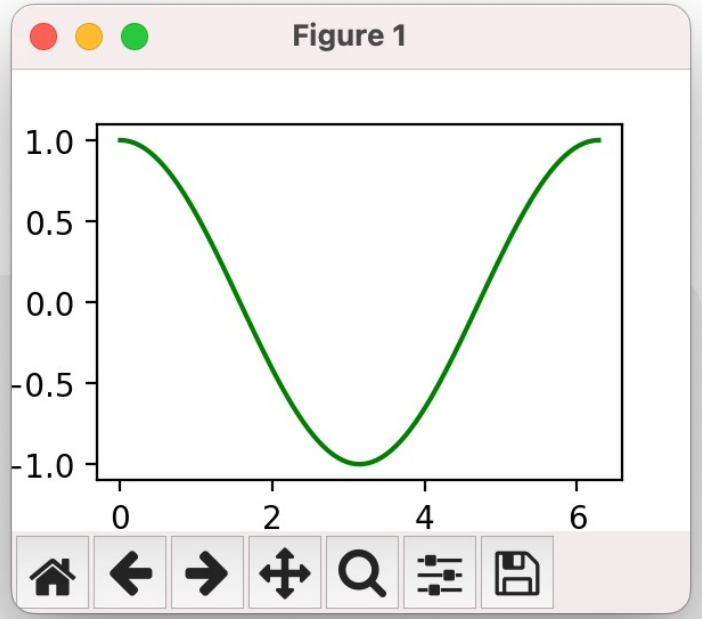
***pip3** is a package installer*

*It will **donwnload** and **install** the give module/package*

*Once completed you can **import** the module/package*

Python – matplotlib

Many functions in matplotlib are the same as Matlab functions



Multi-lines comment

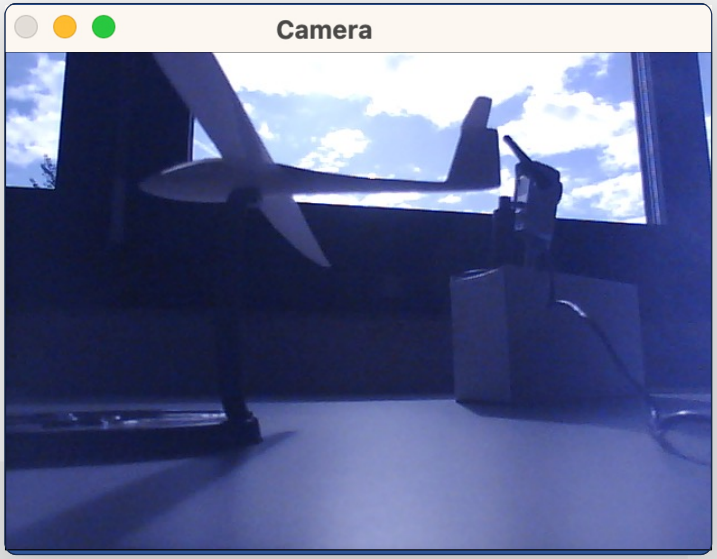
```
"""
Demo matplotlib – very simple plot
ChS 2024
"""

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(0, 2 * np.pi, 100)
Y = np.cos (X)
plt.plot(X, Y, color='green' )
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Another definition for π

See https://matplotlib.org/stable/gallery/
and https://matplotlib.org/cheatsheets/_images/cheatsheets-1.png

Python – OpenCV

OpenCV *(Open Source Computer Vision Library) is a library of functions mainly for real-time computer vision. It was originally developed by Intel, opencv-python provides a python interface to openCV*




```
import cv2

cam = cv2.VideoCapture(0)               # open default camera

while True:
    ret, frame = cam.read()             # grab current frame
    cv2.imshow('Camera', frame)         # display frame
    if cv2.waitKey(1) == ord('q'):      # 'q' pressed ?
        break

cam.release()                           # release camera
cv2.destroyAllWindows()                 # close all windows
```

See https://docs.opencv.org/4.x/index.html

Python – try catch

When an error/exeption occurs, Python will normally stop and generate an error message.
These exceptions can be delt with before the normal python handling

*The **try** block lets you test a block of code for errors.*
*The **except** block lets you handle the error.*
*The **else** block lets you execute code when there is no error.*
*The **finally** block lets you execute code, regardless of the result of the try/except blocks.*

```
try:
  print(x)                                # x is undefined -> error

except:                                   # catch the error before
  print("An exception occurred")          #    normal handling
```

One can raise exception with `raise Exception("my personal exception is raised!")`

Python – try catch

*There can be more than one **except,** one for each error type*

```
try:
  import Tralala
  print(x)

except NameError:
  print("x not defined")

except ModuleNotFoundError:
  print("Tralala not found")

else:
  print("no error")

finally:
  print("always printed")
```

Python – numpy

NumPy *is an open source the package for scientific computing with Python, written manly in C/C++*
It provides among many other things fast ***array*** *objects than works up to 50x faster than* ***list***
Many many Matlab functions have a ***numpy*** *equivalent*

```
import numpy as np

arr1D = np.array([1, 2, 3, 4])   # note the ',' between values
print(arr1D[0])                  # accessing
1
print(arr1D[-1])                 # negative indexing
4
print(arr1D[1:3])
[2 3]
```

Python – numpy

As for other python objects, be carful with reference vs copy

```
arr1D = np.array([1, 2, 3, 4])

arr1D_b = arr1D                 # Δ not a copy, a reference
arr1D[0]= 6

print(arr1D_b)
[6, 2, 3, 4]

arr1D_c = arr1D.copy()          # arr1D_c is a copy
arr1D[0]= 7

print(arr1D_c)
[6, 2, 3, 4]

arr1D[0:2] = [22, 33]           # similar to ML, note ending index -1
print(arr1D)
[22 33  3  4]
```

Python – numpy

```
arr2D = np.array([[1, 2, 3], [4, 5, 6]])

print(arr2D[0,0])
1

print(arr2D[1],[2])
6

print(arr2D.shape)
(2, 3)

print(arr2D[:,1])          # all lines, col 1
[2 5]

print(arr2D[1,:])          # line 1, all cols
[4 5 6]

print(np.diag(arr2D))
[1, 5]
```

Python – numpy

Some NumPy functions

```
arr1D = np.array([1, 2, 3, 4])
print(arr1D.tranpose())                 # same as ML arr1D'
[[1]
 [2]
 [3]
 [4]]

print(np.sort([8,2,4,1,9]))
[1 2 4 8 9]

print(np.ones((1,3)))                   # same as ML ones(1,3)
[[1. 1. 1.]]

print(np.arange(2,8,3))                 # same as python range
[2, 5]
```

Many many more functions are available
see Numpy reference https://numpy.org/doc/2.1/reference/index.html

Python – venv

Soon, many packages will be installed on your machine. Some packages, depending on the version are not compatible with other packages. Also, you have to ensure that the deployment machine has the right packages.
A virtual environment will ensure that these dependencies are under control.
Create virtual environment based on a dependency file (requirements.txt), then run your code from the create virtual environment

```
>> pip3 freeze

contourpy==1.3.1
cycler==0.12.1
fonttools==4.55.2
kiwisolver==1.4.7
matplotlib==3.9.3
numpy==2.1.3
packaging==24.2
pillow==11.0.0
pyparsing==3.2.0
python-dateutil==2.9.0.post0
six==1.17.0
```

*Use **pip3 freeze** to get installed module version*

requirements.txt *contains a subset of this list*

Python – venv

Create virtual environment, activate it, install needed package according to **requirements.txt,** *(can take some times)*
venv should be recreated "à la demande"
Run your script from within your venv

requirements.txt

```
contourpy==1.3.1
matplotlib==3.9.3
numpy==2.1.3
```

```
>> python3 -m venv /Documents/my_venv
```
*Create a venv named **my_venv***

```
>> source /Documents/LV_venv/bin/activate
```
*Activate **my_venv***

```
>> pip3 install -r /Documents/LV_venv/requirements.txt
```
*Populate **my_venv** with needed packages*
Can take some time (has to download files)
Skip if requirement are already satisfied

```
>> /Documents/LV_venv/bin/python3 yourScritp.py
```
Launch python from within your venv

Anaconda is another IDE/venv manger

Python – input and F String

Use **input()** *to read from the console*

```
name = input("Enter your name:")

print("Hello  " + name)
```

Python – f-string

print() *support formatting,* ***f-String*** *is now the preferred way to specify format*
{ } *defines the placeholder (similar to %.. in C), it contains expression and optionally format specifications similar to C*

```
txt1 = f"the answer is 42"
print(txt1)
the answer is 42

ans = 43
print(f"the answer is {ans}") # {} is a placeholder
the answer is 43

print(f"the answer is {ans:.1f}") # :format similar to C
the answer is 43.0

print(f"the old answer was {ans-1}") # can use expression
the old answer was 42.0
```

Python – File

Very similar to C and Matlab
Use **try-exept***: to handle file error*

```
f = open("demo.txt", "r")
print(f.read(5))           # read 5 char
print(f.readline())        # read a line, until eol char
print(f.read())            # read the remaining char
f.close()
```

```
try:
  f = open("noFile.txt","r")
  print(f.read())
except:
  print("Problems opening noFile.txt")
finally:
  f.close()
```

Conclusions

- Scratched the surface, similar to Matlab (and C) with differences
- Indentation defines **bloc**
- **:** at the end of loop/condition expression
- New compound types
- **.**notation to access object methods, this is OOP, not *(yet)* covered
- **Δ** reference copy and not content (same in C) use **.copy** to copy content
- No memory management ☺
- More than one ways to do the same operations
- Many modules provides similar functions
- Numpy is used everywhere, it speedups array operations among other thing
- Use **venv** to run script within specific environments
- Keep exploring Python to write more pythonic code, i.e. follows the conventions of the Python community and uses the language in the way it is intended to be used.