DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Introduction to Python

Machine Learning: Lab 0

Matteo Cederle
(formerly by Alessio Maritan, Jacopo Giordano,
Francesco Zanini and Luca Zancato)

Department of Information Engineering
University of Padova

✉ matteo.cederle@phd.unipd.it

October 16, 2025

How to access to Python in Te/Ue

- Turn on your computer and in the boot menu choose `Boot DEI` (this should happen automatically) and then `Linux`.
- Log in with your `@dei` credentials.
- Go to the web page of the Machine Learning course (on `stem.elearning`).
- Look for the folder `LAB0 - Introduction to Python` and download its content (there are also these slides).
- Open a terminal and type : `jupyter notebook`
- In the `jupyter` window browse to the notebook (the `.ipynb` file) you just downloaded from e-learning.

Labs schedule

- October 16th, 2025 : LAB0 - Intro to Python
- October 23rd, 2025 : LAB1 - Basic Probability with Python
- October 30th, 2025 : LAB2 - Least Squares
- November 6th, 2025 : LAB3/HW1 - Assistance to HW1
- November 13th, 2025 : LAB4 - GD and SGD
- November 27th, 2025 : LAB5/HW2 - Assistance to HW2
- December 4th, 2025 : LAB6 - Gaussian Processes for regression
- December 18th, 2025 : LAB7/HW3 - Assistance to HW3
- January 8th, 2026 : LAB8/HW4 - Assistance to HW4

Homeworks schedule

1. First HW (Linear Regression and Classification)
   - October 31st, 2025 : HW released
   - November 12th, 2025 : HW due
2. Second HW (Regularization)
   - November 20th, 2025 : HW released
   - December 3rd, 2025 : HW due
3. Third HW (SVMs and NNs)
   - December 11th, 2025 : HW released
   - December 28th, 2025 : HW due
4. Fourth HW (Unsupervised Learning)
   - December 29th, 2025 : HW released
   - January 13th, 2026 : HW due

Overview

1 Why Python ?

2 Python generalities

3 Objects and Types

4 Containers

5 Control Flow

6 Functions

7 Importing Modules

8 NumPy

Why Python for Machine Learning ?

Python is the number 1 programming language for ML, and here's why :

- simple and intuitive syntax
- concise and readable code
- does not require a license and is platform independent
- object oriented language
- a lot of resources available online

Why Python for Machine Learning ?

Python is the number 1 programming language for ML, and here's why :

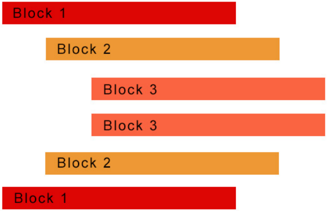
- simple and intuitive syntax
- concise and readable code
- does not require a license and is platform independent
- object oriented language
- a lot of resources available online

Machine Learning oriented modules in Python (a.k.a. libraries) :

- NumPy (high-performance scientific computing and data analysis)
- SciPy (for advanced computing)
- Scikit-learn (ML in general)
- Pandas (general-purpose data analysis)
- Seaborn (visualization)
- OpenCV (computer vision)
- PyTorch (Deep Learning)

Indentation

Python programs get structured through indentation ! No bracket {}, semicolon (;) or end string is necessary.



Block 1

Block 2

Block 3

Block 3

Block 2

Block 1

- **Increase indent** after a new statement (e.g. a conditional one)
- **Maintain indent** to declare the scope of the block
- **Reduce indent** to end the block
- **Blank and comment** lines are ignored

```python
for item in range(10):
    print('Surprisingly ' + str(item) + ' is an ', end=' ')
    if item % 2 == 0:
        print('even',  end=' ')  # this is a comment in Python
    else:
        print('odd',  end=' ')   # this is another comment in Python
    print(' number.')
print('Now you know even and odd numbers up to 9.')
```

Remark

Do not mix tabs and spaces ! You may get **indentation errors**.

Objects and Types

Python programs manipulate **data objects**, they can be :

- **Scalar** (atomic units that cannot be subdivided)
- **Non-scalar** (have internal structure that can be accessed)

Scalar object table (Immutable types)

int	Any integer
float	Floating point number (64 bit)
complex	Complex numbers (imaginary unit is j)
bool	True, False
str	A sequence of characters
bytes	A sequence of unsigned 8-bit
NoneType	Null equivalent, every instance of None is of NoneType

```
>>> type(10)                     --->         <type 'int'>
>>> type(7.3)                    --->         <type 'float'>
>>> type(1+2j)                   --->         <type 'complex'>
>>> a = True                     --->         <type 'bool'>
>>> type('Machine Learning')     --->         <type 'str'>
>>> type(None)                   --->         <type 'NoneType'>
```

Scalar Operations

Arithmetic Operators :

- '+' Addition
- '-' Subtraction
- '*' Multiplication
- '/' Division
- '//' Floor Division
- '%' Modulo
- '**' Power

Comparison Operators

- '==' Equal to
- '>' Greater Than
- '>=' Greater/Equal To
- '<' Less Than
- '<=' Less/Equal To
- '! =' Not Equal

Boolean and String Operators

- 'and'
- 'or'
- 'not'
- ★ '+' Concatenate
- ★ '*' Repeat
- ★ '==' Comparison

```
>>> print(17/3)     # in python >= 3 the result is a float
5.666666666666667
>>> print(17. / 3)  # (or 17/3.) classic division, returns a float
5.666666666666667
>>> print(17 // 3)  # floor division discards the fractional part
5
>>> print(17 % 3)   # the % operator returns the remainder of the division
2
```

String Indexing

```
>>> word = 'Python'  # you can define a string using either ' ' or " "
>>> word[0]    # character in position 0
'P'
>>> word[-1]   # last character
'n'
>>> word[-2]   # second-last character
'o'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
>>> word[:2]   # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:]   # characters from position 4 (included) to the end
'on'
>>> word[-2:]  # characters from the second-last (included) to the end
'on'
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
  0   1   2   3   4   5
 -6  -5  -4  -3  -2  -1
```

Remark :

The start index is always included while the end is always excluded. Therefore we can write any string s as $s[: i] + s[i :]$.

TODO 1 :

```python
## TODO 1

our_course = "Machine Learning"

# Print 'ML' using characters from the variable our_course (exploiting concatenation: the plus operator)

# Print only the first word in the variable our_course (i.e. 'Machine')

# concatenate (with a space in between) the 4-th character in our_course ('h') and the first 4 characters from the second word ('Lear')

# Try the same 3 points using the reverse indexing

```

Remark :

Today we introduce only the **basic concepts** and **features** of the Python language and system, along with a description of **standard objects and their methods**, and modules. If you want to dive deeper into Python programming, check out the Python documentation (click on the link and go to Library reference).

Lists

Definition

A **list** represents an ordered, mutable collection of objects (any type of Python object).

Remark

You can mix and match any type of object in a list, add to it and remove from it at will.

How to define a list ?

```
>>> empty_list = []     # it evaluates False as boolean
>>> empty_list = list() # alternative syntax

>>> char_list = list(('a', 'b', 'c', 'd'))  # conversion to list
>>> num_list = [1, 2, 3, 4]
>>> string_list = ['I','will','pass','ML']

>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> print(x) ---> [['a', 'b', 'c'], [1, 2, 3]]
                   # how to index in this case?
```

TODO 2 :

```
## TODO 2

our_course_list = list(our_course)
# Please note the difference between `our_course` (that is a `string`) and `our_course_list` (that is a `list`)

# Perform the same indexing as before

# Modify the first character in our_course_list to its lowercase version

# Do the same with the string our_course (what do you observe?)

# Try to access the item 'b' and the item 3 in variable x
```

Lists

Indexing :

- Same as Strings indexing

Membership :

- *use 'obj_name in list_name'* to check membership

Other operations :

- '+' : concatenation
- 'len(list)' : get list length
- '.append(obj)' : append item `obj`
- '.pop()' : return and remove last item
- '.remove(obj)' : remove the *first* occurrence of `obj`

```
>>> print(x[0])                          ---> ['a', 'b', 'c']
>>> print(x[1][1])                              ---> 2

>>> int_list = [10,9,8,7,6,5,4,3,2,1]
>>> int_list.append(9)                   ---> [10,9,8,7,6,5,4,3,2,1,9]
>>> int_list.remove(9)                   ---> [10,8,7,6,5,4,3,2,1,9]
>>> int_list.pop()                       ---> [10,8,7,6,5,4,3,2,1]
>>> print(int_list[::-1])                ---> [1,2,3,4,5,6,7,8,9,10]
                                         # all elements in reverse order
>>> print(int_list[0:9:2])               ---> [10,7,5,3,1]
                                         # elements in even positions


>>> print(5 in int_list)                 ---> True
>>> print('hello' in 'hello world')      ---> True
```

Mutability

int, float, strings are immutable while lists are mutable. What's the difference ?
Immutable :

```
>>> x = 5
>>> y = x      # This means that y = x = 5
>>> x = 9      # Now we have the x = 9 and y = 5 (y does not change!)

>>> word = 'Python'
>>> word[0] = 'T'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Mutable :

```
>>> x = [1,2,3]
>>> y = x          # This means that y = x = [1,2,3]
>>> x.append(4)    # x = [1,2,3,4] and y = [1,2,3,4] (y does change!)
>>> y.pop()        # x = [1,2,3] and y = [1,2,3] (x does change!)
>>> x[1] = 123     # both x and y equal to [1, 123, 3]
```

Remark

Mutable means that x now has a reference to the value of y (both of them 'point' to the same memory). **Immutable** means that x now has a copy of the value of y.

TODO 3 :

```python
## TODO 3

# Print every intermediate step of the previous cell
# (just copy-paste it and add a print statement after each line)

int_list = [1,2,3,4]
# Use only the methods .append() and .remove() to reverse int_list
```

Tuples

Definition

Tuples are ordered immutable collections.

```
>>> num_tuple = (1, 6)    # tuple definition, using ( ) instead of [ ]
```

Indexing :

- Same as Strings and lists indexing

Membership :

- Same as lists membership

Other operations :

- '+' Concatenation
- 'len(tuple)' Get tuple length

Remark

Tuples are just like lists but they are immutable : their values cannot be changed !

Question : why do we need tuples if lists do more ?

- Tuples are lighter and more memory efficient (if used properly)
- Tuples can be used as keys in a dictionary

Dictionaries

Definition

A dictionary (a.k.a. associative array) consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

Creating a dictionary :

```
>>> name2grade = dict()             # Creating an empty dictionary
>>> name2grade['Fabio'] = 18        ---> {'Fabio': 18}
>>> name2grade['Leonardo'] = 31     ---> {'Fabio': 18, 'Leonardo': 31}
>>> name2grade['Alessandro'] = 30   ---> {'Fabio': 18, 'Leonardo': 31,
                                          'Alessandro': 30}
```

The first part of a couple is the **key**, the second is the **value**.

```
>>> print(name2grade['Alessandro']) ---> 30
>>> print(name2grade.keys())   ---> ['Fabio', 'Leonardo', 'Alessandro']
>>> print(name2grade.values()) ---> [18, 31, 30]
>>> del name2grade['Fabio']    ---> {'Leonardo': 31, 'Alessandro': 30}
>>> print(name2grade.items())  ---> [('Leonardo', 31), ('Alessandro', 30)]
```

if Statement

Standard syntax using **if**, **elif** and **else**.

Remark

Remember to put colon after the conditions (and else).

```python
x = 1
if x > 2:
    print('x > 2')
elif x == 2:
    print('x = 2')
else:
    print('x < 2')
print("The value of x is: " + str(x))
```

```python
grade = 27.2
course = "Calculus"
if (grade > 26) and (not (course != "ML")):
    print("Everything is good")
else:
    print("Need to study for ML!")
```

for Statement

Also the **for** statement is pretty straightforward !

```python
num_string = [1, 10, 42]

# These five loops print the same!

for x in num_string:
    print(x)

for x in range(3):
    print(num_string[x])

for index in range(len(num_string)):
    print(num_string[index])

for index in range(0, len(num_string), 1):   # range(start, stop, step)
    print(num_string[index])

for index, element in enumerate(num_string):  # tuple unpacking
    print('Index is: ' + str(index))
    print('Value is: ' + str(element))
    print('That is equivalent to ' + str(num_string[index]))
```

The same can be done with any list (regardless of its content), tuple or string.

TODO 4 :

```python
## TODO 4

# Modify the previous code using the following `string_list` in place of `num_string`.
string_list = list(['I','will','pass','ML'])

# Print the elements of `string_list` separated by spaces.
```

Iterating over dictionaries

The **for** statement can be exploited to iterate through dictionaries too !

```python
grades = {30: ['Leonardo', 'Alessandro']}       # grades is a dictionary
grades[26] = ['Fabio', 'Luca']                  # adding a new key
for key in grades.keys():
    print("Students with grade = " + str(key))
    for elem in grades[key]:
        print(elem)
```

```python
grades = {30: ['Leonardo', 'Alessandro'], 26: ['Fabio', 'Luca']}

for key in grades.keys():         # iterate over the keys of a dict
    print("Key = " + str(key))

for val in grades.values():       # iterate over the values of a dict
    print("Value = " + str(val))

for key, val in grades.items():   # iterate over both keys and values
    print("Key: " + str(key) + "\nValue: " + str(val))
```

TODO 5 :

```
## TODO 5

# Add to the dictionary grades your own (and possibly those of some of your
# friends) with an unbiased estimate of your grade

# Write a for loop to get the average of the grades in the ML course
```

while Statement

Also **while** loop has a standard syntax and you can use flow controllers :

Flow controllers

- **break** : exit the loop
- **continue** : go to the next iteration

```
>counter = 1
while (counter < 10):
    print(counter)
    counter = counter +1
    if counter == 5:
        break
```

```
>counter = 1
while (counter < 10):
    counter = counter +1
    if counter in [5, 6, 9]:
        continue
    print(counter)
```

functions

In Python you can :

- construct new functions during the execution of a program (using `def`)
- **store functions in data structures**
- **pass functions** as arguments to other functions
- **return functions** as the values of other functions.

Important facts :

- all functions return a value (if no return statement is specified then None is returned)
- tuples are used to return multiple values
- functions can take **positional** and **keyword arguments**
- functions can take variable number of arguments (also no arguments)
- you can specify default arguments

```python
def my_add(a, b=2):  # b is a default argument
    return a + b

def my_div(a, b):
    if b == 0:
        return 'Division by zero'
    else:
        return a / b
```

functions

```python
def say_hello():
    print('Hello')
```

How to call a function :

```python
print(my_add(1,6))                   --->   7  # positional order
print(my_add(5))                     --->   7  # default values
print(my_add(a=3, b=4))              --->   7  # keyword arguments
print(my_add(b=4, a=3))              --->   7

functions_list = [my_add, my_div]    # storing functions
inputs = [0, 1, 2, 3, 4]

functions_list[0](inputs[0])         --->   2

for f in functions_list:
    for i in inputs:
        print(f(10, i))
```

TODO 6 :

```
## TODO 6

# Write the function: my_repeater(n) that calls say_hello() n times

# Create a dictionary with 2 keys: 'my_arithmetics' and 'my_print'.
# Insert in the first key a list containing both my_add and my_div.
# Insert in the second key the functions say_hello and my_repeater.

# Now call one time each function you stored in the dictionary of lists
```

Importing Modules

Writing code in python is fun but sometimes it is necessary (easier and faster) to exploit some kind of library ! That is why we need modules !

Definition

A *module* is a file containing Python definitions and statements.

This means that we can import in our script functions written by others in order to improve our code ! In this course we will need (the following items are links) :

- numpy
- scipy
- sklearn

```
>>> import numpy as np        # import all numpy with alias np: use
                               # np.function_you_want()

>>> arr = np.zeros((3,4))     # this creates an array of zeros of 3 times 4
```

```
>>> from numpy import ones    # this import only the function you want (no
                              # reference to numpy is necessary anymore).
>>> arr = ones((3,4))         # this creates an array of ones 3 times 4
```

NumPy

Definition

NumPy is a very optimized array-processing package. Moreover it presents a syntax very similar to MATLAB (it will be very intuitive for you !)

Let’s introduce the main object in NumPy : **ndarray object**

- Table of elements (usually numbers) of the same type, indexed similarly The dimensions are called *axes*. The number of axes is called *rank*.

```
>>> import numpy as np
>>> a = np.array([1,2,3])               # single row vector
>>> b = np.array([[1,2,3],[4,5,6]])     # 2 times 3 ndarray (i.e. matrix)
>>> c = np.eye(4)                       # 4 times 4 identity matrix
>>> d = np.random.random((2,4))         # 2 times 4 random matrix
```

NumPy

Indexing in NumPy (the same as before)

```
>>> print(d[:,:])       # whole ndarray
>>> print(d[1,1])       # element in the second row and second column
>>> print(d[:,0:2])     # the first 2 columns
>>> print(d[:,1:1])     # empty, since the slice is on the same index
>>> print(d[:,1:-1])    # second and third column
>>> print(d[0, 1:])     # from second to last elements of the first row
```

Access properties of ndarrays

```
>>> d = np.random.random((2,4))
>>> print(d.ndim)   --->  2     # array dimensions (axes)
>>> print(d.shape)  ---> (2,4)  # shape of the array (note it is a tuple)
>>> print(d.size)   --->  8     # total number of elements of the array
```

NumPy

Creating ndarrays and **reshaping**. (check the documentation)

```
# sequence of integers from 0 (included) to 30 (excluded) with step 5
>>> f = np.arange(0,30,5)

# sequence of 10 values evenly spaced in the interval [0, 5]
>>> g = np.linspace(0,5,10)

# 3 times 4 matrix whose elements are random floats in [0, 1)
>>> arr = np.random.random((3,4))

# converts the list into an array
>>> listarr = np.array([1, 2, 3])

# reshape array from 3x4 to 2x2x3
>>> newarr = arr.reshape(2,2,3)

# collapse to 1 dimension
>>> fltarr = arr.flatten()
```

NumPy

Stacking ndarrays and **Splitting**. Run the code and see the output !

```
>>> a, b = np.zeros((2,2)), np.ones((2,2))  # note the double assignment
>>> c = [5,6]

>>> print(np.vstack((a,b)))         # Vertical stacking
>>> print(np.hstack((a,b)))         # Horizontal stacking
>>> print(np.column_stack((a,c)))   # Stacking columns
>>> print(np.concatenate((a,b),1))  # Concatenate along axis 1
                                    # (same as column_stack)

>>> print(np.hsplit(a,2))           # Horizontal splitting
>>> print(np.vsplit(a,2))           # Vertical splitting
```

TODO 7 :

```
## TODO 7

# Print every line (that was not printed in the NumPy cells) and compare numpy to MATLAB syntax.

# Write a function my_random(a, b, c) with three arguments, which outputs a one-dimensional array of length c, where each entry is uniformly distributed between a and b.

# Assume you pass the exam (i.e. 18<=grade<=30). By making use of my_random, draw two samples of possible grades, convert them into an integer and print the higher one.
```