Python and Numpy

Antonio J. Sánchez Salmerón

February 10, 2023

- Requirements:
  1. Install all the tools according to the guide “T0_Tools installation.pdf”.
  2. The estimated time for this activity is approximately one hour.
- Objectives of this notebook:
  1. Python presentation.
  2. Presentation of Numpy.
- Summary of activities:
  1. Small pieces of code will be analyzed.
  2. You will start programming in Python using Numpy.

Contents

1 Introduction to Python

Python is an interpreted programming language whose philosophy emphasizes the readability of its code. It is a cross-platform and multi-paradigm programming language, as it supports object-oriented, imperative and functional programming.

This tutorial is intended to be useful as a first contact with Python. More Python programming information can be found in this other Python tutorial or in this book.

[1]:
```
# **** It is important to consult the help of the version you are using ****
!python --version   # Shows the installed Python version
!jupyter --version
```

```
Python 3.8.13
Selected Jupyter core packages…
IPython          : 8.1.1
ipykernel        : 6.9.1
ipywidgets       : 7.6.5
jupyter_client   : 7.1.2
jupyter_core     : 4.9.2
jupyter_server   : 1.13.5
jupyterlab       : 3.3.2
nbclient         : 0.5.11
nbconvert        : 6.1.0
nbformat         : 5.1.3
notebook         : 6.4.8
qtconsole        : not installed
traitlets        : 5.1.1
```

1.1 Basic data types

1.1.1 Numbers

[ ]:
```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Power; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

1.1.2 Booleans

```
[ ]: t = True
     f = False
     print(type(t)) # Prints "<class 'bool'>"
     print(t and f) # AND; prints "False"
     print(t or f)  # OR; prints "True"
     print(not t)   # NOT; prints "False"
     print(t != f)  # XOR; prints "True"
```

1.1.3 Strings

```
[ ]: hello = 'hello'    # String literals can use single quotes
     world = "world"    # or double quotes; it does not matter.
     print(hello)       # Prints "hello"
     print(len(hello))  # length; prints "5"
     hw = hello + ' ' + world  # concatenation
     print(hw)          # Prints "hello world"
     hw12 = '%s %s %d' % (hello, world, 12)  # format
     print(hw12)        # Prints "hello world 12"
```

1.1.4 Lists

```
[ ]: # Assignments
     xs = [3, 1, 2]    # Creates a list
     print(xs, xs[2])  # Prints "[3, 1, 2] 2"
     print(xs[-1])     # Negative indices count from the end of the list; prints "2"
     xs[2] = 'foo'     # Lists can contain elements of different types
     print(xs)         # Prints "[3, 1, 'foo']"
     xs.append('bar')  # Appends an element to the end of the list
     print(xs)         # Prints "[3, 1, 'foo', 'bar']"
     x = xs.pop()      # Removes an element at the end of the list
     print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

```
[ ]: # Indexing
     nums = list(range(5)) # Creates a list with a range of integer values
     print(nums)           # Prints "[0, 1, 2, 3, 4]"
     print(nums[2:4])      # Gets the elements from index 2 to 4 (excluding the 4);
                           # prints "[2, 3]"
     print(nums[2:])       # Gets the elements from index 2 to the end;
                           # prints "[2, 3, 4]"
     print(nums[:2])       # Gets the elements from the beginning to the index 1;
                           # prints "[0, 1]"
     print(nums[:])        # Gets the elements of the whole list;
                           # prints "[0, 1, 2, 3, 4]"
     print(nums[:-1])      # The indices can be negatives; prints "[0, 1, 2, 3]"
     nums[2:4] = [8, 9]    # Assigns a sublist
     print(nums)           # Prints "[0, 1, 8, 9, 4]"
```

[ ]:
```
# Loops
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
    print('hello')
print('fin')
```

[ ]:
```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
```

[ ]:
```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)
```

[ ]:
```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)
```

[ ]:
```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)
```

1.1.5 Tuples

[ ]:
```
r= 4,3,1
```

[ ]:
```
# Assignments
a = (4,)                    # To create a tuple with a single element,
                            # you need to include a comma at the end
print('Type:',type(a))
print('Length:',len(a))
b = 'hola','coracola'       # The parentheses are not necessary
print(b)
a = a + b                   # concatenation operation
print(a)
print(a[-1])                # access to the last item
```

[ ]:
```
# Loops
for element in a:
    print(element)
```

[ ]:
```
# Problem: with tuples the following assignment of a value is not allowed by
# indexing the position
a[3]=0
```

However, it is shown below, that lists allows this kind of assignment.

[ ]:
```
a=[1,'po', 'lists', 4]
print(a[-1])
a[-1]= 'success'
print(a[-1])
```

[ ]:
```
# In addition, lists also allow concatenation with the + operand
b=[[],[]]
b[0]=b[0]+[1]
b[0]=b[0]+[2]
print(b)
```

1.1.6 Dictionaries

[ ]:
```
# Assignments
d = {'cat': 'cute', 'dog': 'furry'}  # Creates a dictionary
print(d['cat'])       # Gets the value from the key entry 'cat'; prints "cute"
print('cat' in d)     # Checks whether the dictionary has the entry 'cat';
                      # prints "True"
d['fish'] = 'wet'     # Adds a new entry with the key 'fish' to the dictionary
print(d['fish'])      # Prints "wet"

# print(d['monkey'])  # KeyError: 'monkey' is not an entry of d
print(d.get('monkey', 'N/A')) # Gets the key 'monkey' with a value by default;
                              # prints "N/A"
print(d.get('fish', 'N/A'))   # Prints "wet"
del d['fish']         # Removes the 'fish' item from the dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer an input key; prints "N/A"
```

[ ]:
```
# Loops
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

[ ]:
```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

1.1.7 Sets

[ ]:
```
# Assignments
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element belongs to a set; print "True"
print('fish' in animals)  # Print "False"
```

```
animals.add('fish')       # Add an element to the set
print('fish' in animals)  # Print "True"
print(len(animals))       # Cardinal number of the set; prints "3"
animals.add('cat')        # Does not add anything if the element already␣
 ↪belongs to the set
print(len(animals))       # Prints "3"
animals.remove('cat')     # Removes an element from the set
print(len(animals))       # Prints "2"
```

[ ]:
```
# Loops
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: fish", "#2: dog", "#3: cat"
```

[ ]:
```
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Prints "{0, 1, 2, 3, 4, 5}"
```

1.2 Functions

[ ]:
```
def sign(x):
    if x > 0:
        return 'positivo'
    elif x < 0:
        return 'negativo'
    else:
        return 'cero'
```

[ ]:
```
for x in [-1, 0, 1]:
    print(sign(x))
```

1.3 Classes

[ ]:
```
class Object:
    def __init__(self, name, pos=(0,0)):
        self.name = name
        self.pos = pos
        self.inc_pos = (10,5)

    def next_pos(self):
        self.pos = ( self.pos[0]+self.inc_pos[0], self.pos[1]+self.inc_pos[1] )
```

[ ]:
```
obj = Object('car')
```

[ ]:
```
for t in range(10):
    print('Object pos of', obj.name, 'at', t, obj.pos)
    obj.next_pos()
```

2 Introduction to Numpy

Numpy is Python's scientific computing library. It provides the definition of the multidimensional array object class and all the methods for working with these arrays.

There are many tutorials to learn Numpy, such as this tutorial de Numpy. To use Numpy's functionality, you must first import the library.

[ ]:
```
import numpy as np
```

[ ]:
```
# **** It is important to consult the help of the version you are using ****
print('Numpy:',np.__version__) # Displays the version of Numpy
```

2.1 Data type ndarray

[ ]:
```
a = np.array([1, 2, 3])   # Creates a vector from a list
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Changes the value of the first element
print(a)                  # Prints "[5, 2, 3]"
b = np.array([[1,2,3],[4,5,6]])    # Creates a two-dimensional array from a
                                   # list of lists
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

[ ]:
```
a = np.zeros((2,2))    # Creates an array of zeros
print(a)               # Prints "[[ 0.  0.]
                       #          [ 0.  0.]]"
b = np.ones((1,2))     # Creates a vector of ones
print(b)               # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Creates a constant array of 7
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"
d = np.eye(2)          # Creates an identity matrix of size 2x2
print(d)               # Prints "[[ 1.  0.]
                       #          [ 0.  1.]]"
```

2.1.1 Indexed access

[ ]:
```
# Creates a matrix of size (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Indexes a submatrix
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# Indexes an element
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

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

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

[ ]:
```
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)  # prints "array([[ 1,  2,  3],
          #                [ 4,  5,  6],
          #                [ 7,  8,  9],
          #                [10, 11, 12]])"

# Creates a vector of indexes
b = np.array([0, 2, 0, 1])

# Selects one element from each row with the indices of b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"

# Increases by 10 units one element of each row with the indices of b
a[np.arange(4), b] += 10
print(a)  # prints "array([[11,  2,  3],
          #                [ 4,  5, 16],
          #                [17,  8,  9],
          #                [10, 21, 12]])
```

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

bool_idx = (a > 2)   # Finds the elements that are greater than 2;
```

```
                      # returns an array of Booleans of the same size as a
print(bool_idx)       # Prints "[[False False]
                      #          [ True  True]
                      #          [ True  True]]"

# Boolean array can be used to access the elements True
print(a[bool_idx])  # Prints "[3 4 5 6]"

# All of the above can be implemented in one line:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

[ ]:
```
# Data type of the matrix elements
x = np.array([1, 2])   # Numpy chooses the datatype
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Numpy chooses the datatype
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)   # We force a particular datatype
print(x.dtype)                         # Prints "int64"
```

2.1.2 Element-by-element operations

[ ]:
```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Two ways to add element to element
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Two ways to add element to element
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Two ways of multiplying element to element
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Two ways to divide element by element
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
```

```
print(x / y)
print(np.divide(x, y))

# Square root element by element
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

2.1.3 Matrix operations

[ ]:
```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Scalar product of two vectorss; prints 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix product by vector; prints vector [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Product of matrix by matrix;  prints the matrix:
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

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

print(np.sum(x))  # Sums all elements; prints "10"
print(np.sum(x, axis=0))  # Sums the elements of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Sums the elements of each row; prints "[3 7]"
```

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

print(x)    # Prints "[[1 2]
            #          [3 4]]"
print(x.T)  # Prints "[[1 3]
            #          [2 4]]"

# NNote: the transpose of a vector does nothing
v = np.array([1,2,3])
print(v)    # Imprime "[1 2 3]"
print(v.T)  # Imprime "[1 2 3]"
```

2.1.4 Propagation or ‘broadcasting’

[ ]:
```
# Add vector v to each row of matrix x,
# storing the result in matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Creates an empty array with the same size as x

# Adds vector v to each row of matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

print(y)    # Prints [[ 2  2  4]
            #         [ 5  5  7]
            #         [ 8  8 10]
            #         [11 11 13]]
```

[ ]:
```
# Adds vector v to each row of matrix x,
# storing the result in matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))   # 4 copies of v are stacked on top of each other
print(vv)                 # Prints "[[1 0 1]
                          #          [1 0 1]
                          #          [1 0 1]
                          #          [1 0 1]]"

y = x + vv  # Sums x and vv element by element
print(y)  # Prints "[[ 2  2  4
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

[ ]:
```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])

y = x + v  # Adds v to each row of x using propagation or 'broadcasting'
print(y)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

[ ]:
```
# Calculates the outer product of two vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
# First, v is transformed into column format
# and then the product with propagation 'broadcast' against w is performed:
```

```
# [[ 4  5]
#  [ 8 10]
#  [12 15]]
print(np.reshape(v, (3, 1)) * w)

# Adds one vector to each row of the matrix
x = np.array([[1,2,3], [4,5,6]])
# x has size (2, 3) and v (3,) then it propagates to (2, 3), resulting in:
# [[2 4 6]
#  [5 7 9]]
print(x + v)


# Adds a vector to each column of the matrix
# x has size (2, 3) and w (2,).
# If we transpose x then it has size (3, 2) and can be propagated
# against w to produce a result of size (3, 2); by transposing
# this result we will have a size of (2, 3)::
# [[ 5  6  7]
#  [ 9 10 11]]
print((x.T + w).T)

# Another solution is to format w to a column vector (2, 1);
# then it can be propagated directly against x:
print(x + np.reshape(w, (2, 1)))

# Multiply a matrix by a constant:
# x has size (2, 3). Numpy treats scalars as vectors of ();
# propagation against (2, 3), produces the following result:
# [[ 2  4  6]
#  [ 8 10 12]]
print(x * 2)
```

2.2 Type conversion between Numpy and Python

[ ]:
```
a = np.arange(10)
print(a)

b = list(a)         # Conversion from ndarray to list
b[5]='hola'
print(b)

if 0 in b:
    print(True)

print(a)            # variables a and b are independent
```

```
c = np.asarray(b)  # Conversion from list to ndarray
print(c)
```

[ ]:
```
a = np.arange(10)
print(a)

b = tuple(a)           # Conversion from ndarray to tuple
print(b)

if 0 in b:
    print(True)

c = np.asarray(b)      # Tuple to ndarray conversion
print(c)
```

2.3 Random number generation

[ ]:
```
e = np.random.random((2,2))  # Creates a 2x2 array of random numbers
print(e)                     # Prints aprox. "[[ 0.91940167  0.08143941]
                             #                 [ 0.68744134  0.87236687]]"
print(np.random.uniform(-1.0,1.0))
```

3 References

- Python documentation
- Numpy documentation
- NumPy Quick Tutorial
- NumPy Reference Manual
- Numerical Python : Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib. by Johansson, Robert. 2nd ed, Apress, 2019.