Basic Programming

Lesson 7. Creating functions. Recursive functions.
Vectors and matrices with numPy

Alexandra Ciobotaru

26 nov 2021

Syllabus

- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- **Lesson 4. Strings. Working with strings.**
- **Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While**
- Lesson 6. ~~Working with matrices in Python. Python numpy library~~. Lesson 5 continued Quiz 1 (25%).
- Lesson 7. Creating functions. Recursive functions. Matrices if we have time.
- Lesson 8. Collections: Dictionaries. JSON construction.
- Lesson 9. Working with files. Reading and Writing.
- Lesson 10. Analyzing dataframes. Pandas and Matplotlib Python libraries.
- Lesson 11. Creating functions. Recursive functions. Quiz 2 (25%).
- Lesson 12. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism.
- Lesson 13. Object-Oriented Programming (cont.). Error handling. Best practices when programming.
- Lab (20%) + final exam (30%).

What you'll learn today:

- 1. Functions
  - 1.1. What are functions?
  - 1.2. What is a recursive function?
  - 1.3. Installing and importing libraries
- 2. Vectors and Matrices in Python using NumPy library
  - 2.1. What are vectors?
  - 2.2. What are matrices?
  - 2.3. What are tensors?

1.1. What is a function?

- You already used some functions before -> rememeber `type, id, len`?
- A mathematical function has an input and an output:

  y = f(x) ---→ x is the input and y is the output (Ex: y = 2*x + 1)

- The same principle applies in programming – a function has one or more inputs and returns one or more outputs:

Function signature

```
def function_name(input):
    # something is done to the input, for instance:
    output = 2 x input
    return output
```

Notice the indentation and words written in red!

- A function has as input one or more **arguments** (if any)
- And returns one or more **values** (if any)
- A good practice is to comment your function just so you remember what it does, what it takes as input and what returns as output. This comment is called **docstring**.

```
def function_name(input):
    '''
    This function triples the input string.
    :param input: string
    :return: string
    '''
    output = input+input+input
    return output
```

- After you define your function, from somewhere in your code you will **call** your function:

```
print(function_name('Hola'))
```

- Defining a function allows you to encapsulate a segment of code, specifying the information that enters and leaves the code.
- You can make use of this “code-capsule” repeatedly and in many different contexts.
- For example, suppose you want to count how many vowels are in a string. The following defines a function that accomplishes this:

```
def count_vowels(in_string):
    '''
    Computes the number of vowels contained in `in_string`
    :param in_string:
    :return:
    '''
    num_vowels = 0
    vowels = "aeiouAEIOU"
    for char in in_string:
        if char in vowels:
            num_vowels += 1   # equivalent to num_vowels = num_vowels + 1
    return num_vowels
```

Exercise – write a basic function

- Write a function named `count_even`. It should accept one input argument, named `numbers`, which will be a list containing integers.
- Have the function return **the number of even-valued integers** contained in the list. Include a reasonable docstring.

1.2. Recursive functions

- A function can call other functions.
- A recursive function is a function that calls **itself**.
- Recursion is the process of defining something in terms of itself.
- Remember factorial from Loops Homework?
- 5! = 1*2*3*4*5

```
# naïve way to find the factorial (23!)
n = 23
fact = 1
for i in range(1,n+1):
    fact = fact * i

print ("The factorial of 23 is : ",end="")
print (fact)
```

- Here is the recursive way of computing factorial:

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = 23
print("The factorial of", num, "is", factorial(num))
```

When we call this function with a positive integer, it will recursively call itself by **decreasing** the number:

- factorial(3) # 1st call with 3
- 3 * factorial(2) # 2nd call with 2
- 3 * (2 * factorial(1)) # 3rd call with 1
- 3 * 2 * 1 # return from 3rd call as number=1
- 3 * 2 # return from 2nd call
- 6 # return from 1st call

1.3. Installing and importing libraries

- Python libraries are functions made by other people
- Example: let's use some code that translates text:
  - In PyCharm terminal, type: `pip install langdetect`
  - To use the installed library, you must import the function you wish to use from that library:

```
>>> from langdetect import detect
```

  - To detect the language of a text:

```
>>> detect("War doesn't show who's right, just who's left.")
'en'
>>> detect("Ein, zwei, drei, vier")
'de'
```

2. Vectors and Matrices in Python using NumPy library

- NumPy is a Linear Algebra Library in Python
- NumPy stands for “Numerical Python”
- In Google colab numpy is pre-installed
- If you use PyCharm, to install NumPy go in terminal and type:

```
pip install numpy
```

- Then, at the beginning of your program type:

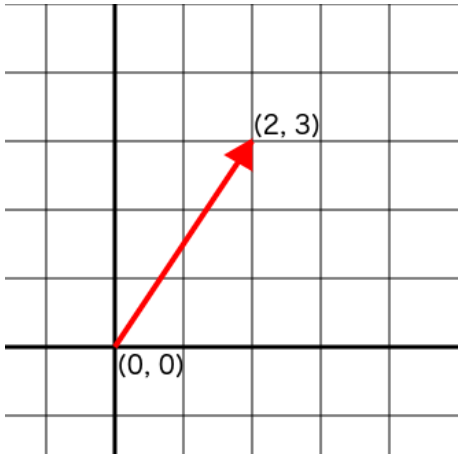
```
import numpy as np
```

alias: In Python, alias is an alternate name for referring to the same thing.

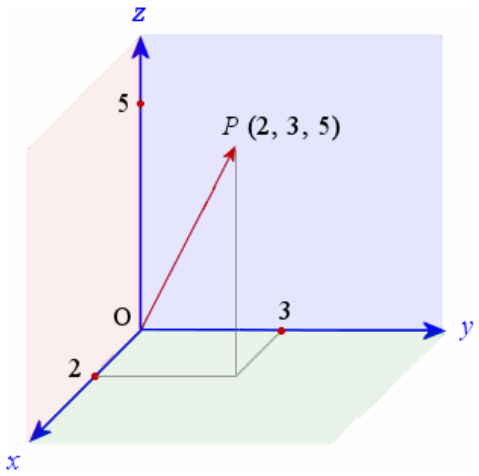
- We usually use numPy to work with **arrays** (1D, 2D) and **tensors** (multi dimensional arrays)
- (optional) If you want to master numPy: https://numpy.org/doc/stable/numpy-user.pdf

What are vectors?

- Vectors are pairs of numbers that represent points in space



2D
v = [2, 3]



3D
v = [2, 3, 5]

4D -> vector with 4 elements
v = [2, 3, 5, 7]

5D -> vector with 5 elements
v = [2, 3, 5, 3, 8]

6D -> vector with 6 elements
v = [2, 3, 5, 3, 6, 4]

…and so on

Image source: https://medium.com/hackernoon/more-functional-refactoring-a-vector-library-4bf3d6b88612

Image source: https://www.intmath.com/vectors/7-vectors-in-3d-space.php

- Creating a vector out of a list:

```
my_list = [1, 2, 3]
arr = np.array(my_list)
```

- Find its shape: `arr.shape`

- Creating a vector of zeros: `z = np.zeros(3)` → `array([0., 0., 0.])`
- Creating a vector of ones: `a = np.ones(3)` → `array([1., 1., 1.])`
- Creating a vector of evenly spaced points:

```
b = np.linspace(2,10,5) -> array([ 2.,  4.,  6.,  8., 10.])
```

What are matrices?

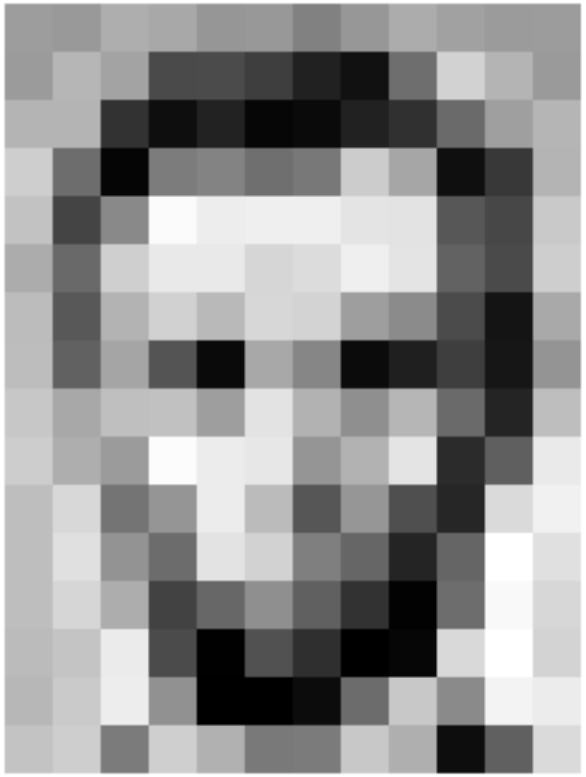
- A matrix is a rectangular array of numbers, with *n* columns and *m* lines:

$$A_{m,n} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

- Creating a 3x3 numpy matrix out of a list of lists

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

How do computers see grayscale images?



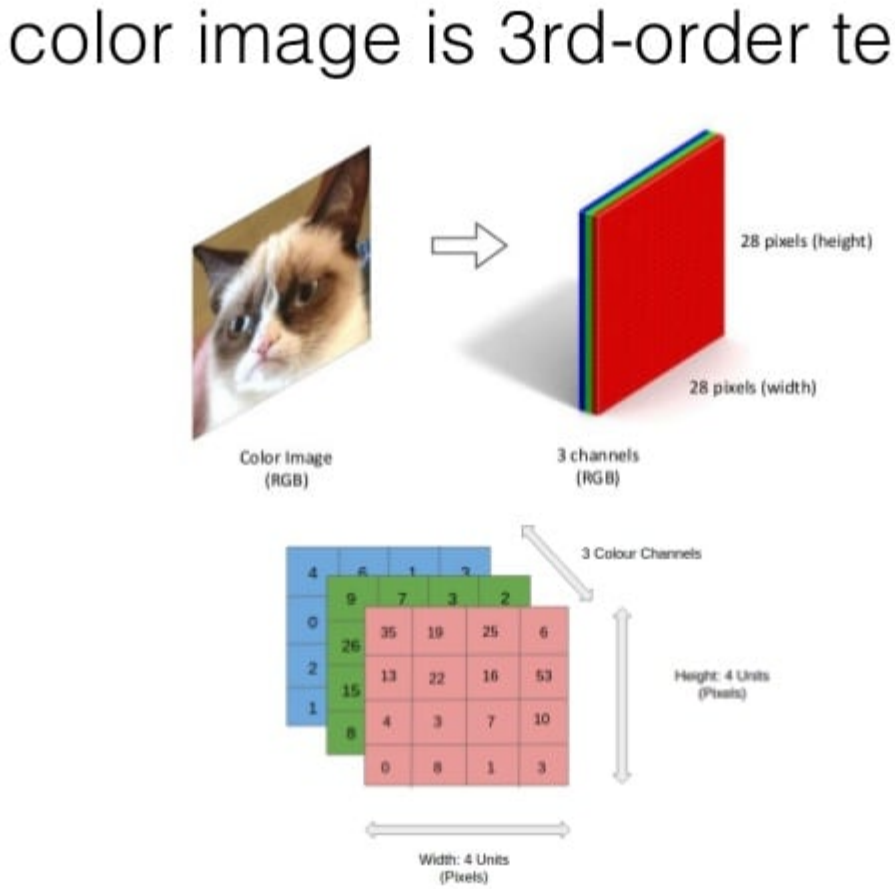
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

0 = black
255 = white

Image source: https://towardsdatascience.com/computer-vision-101-working-with-color-images-in-python-7b57381a8a54

How do computers see color images?



More on this subject:
https://youtu.be/mAMTXJJQBDI

Image source: https://lisaong.github.io/mldds-courseware/01_GettingStarted/numpy-tensor-slicing.slides.html

What are tensors?

- A tensor is a multidimensional matrix

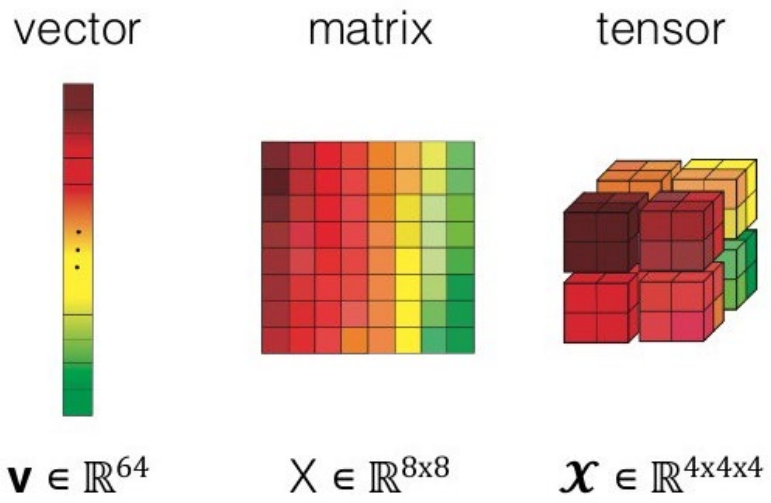


Image source: https://medium.com/analytics-vidhya/numpy-on-gpu-tpu-efb8d367020a

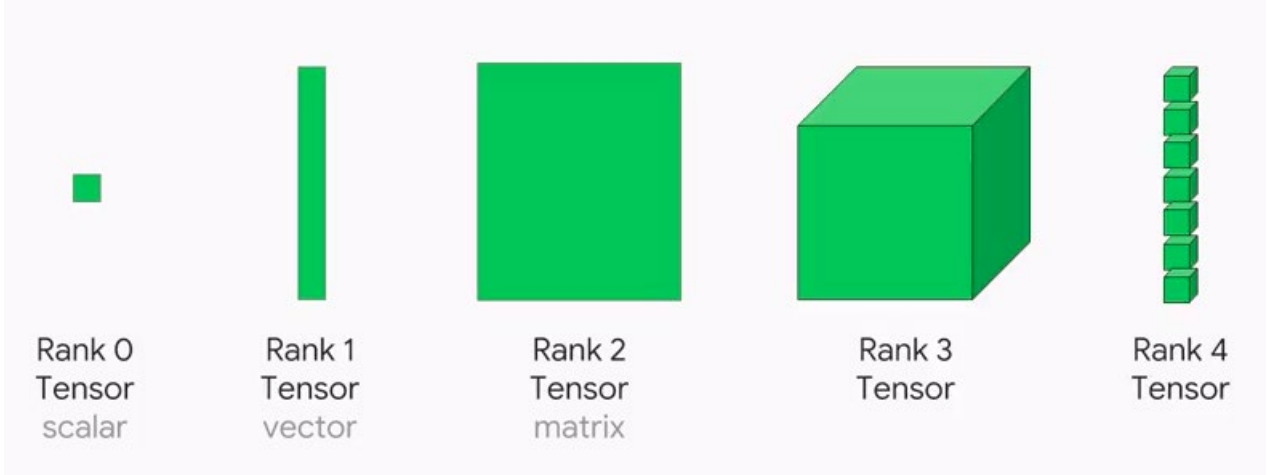


Image source: https://medium.com/mlait/tensors-representation-of-data-in-neural-networks-bbe8a711b93b

1 dimensional array -> vector
2 dimensional array -> matrix
3 dimensional array -> cubical matrix
4 dimensional array -> stack of cubical matrices

```
from skimage import io, color
import matplotlib.pyplot as plt

photo = io.imread('photo.jpeg')
print(type(photo))
print(photo.shape)

plt.imshow(photo)
plt.show()
plt.imshow(photo[::-1])
plt.show()

grayscale = color.rgb2gray(photo)
plt.imshow(grayscale, cmap=plt.cm.gray)
plt.show()
```

Optional – more on this subject: https://youtu.be/xECXZ3tyONo minute 6:20

Thank you!