TYPES OF ARRAYS

April 21, 2021

1 INTRODUCTION TO NUMPY

In this practice, we are going to work with arrays. Arrays are fundamental for algebra, since they allow us to carry out a multitude of operations on our computer, spending less resources than if we used other storage structures. In addition, they are generally used to work with one type of data at a time, which greatly simplifies operations.

The basic package for creating vectors and matrices in Python is **Numpy** (**NUM**eric **PY**thon).

Main benefits:

- Spend **less memory**.
- **Faster** to read.
- You don't have to check what **datatype** it is.

[1]:
```
# Importar paquetes
#==================================
import numpy as np
```

2 Basic functions

2.1 Know the properties of the vector (ndim, shape, dtype)

- **ndim**: Know the dimension of the array
- **shape**: Know the number of columns and rows
- **dtype**: What is the datatype (number of bits)

[3]:
```
# Know the dimension of the array
#==================================
matrix = np.array([[9,8,7],[6,5,4]])

print("Dimension: {}".format(matrix.ndim))
print("Datatype: {}".format(matrix.dtype))


# Turn int to float
#==================================
matrix = np.array([[9,8,7],[6,5,4]], dtype="float32")
```

```
print("Datatype: {}".format(matrix.dtype))


# Know the shape of the array
#================================
print("Shape: {}".format(matrix.shape))
```

```
Dimension: 2
Datatype: int32
Datatype: float32
Shape: (2, 3)
```

2.2 Storage

We can always change the format of the data, although a smaller format means less precision if your project does not require great precision, it will not be a problem to change them.

I repeat, depending on what your project is, you can change or not the format of your data.

For floats

Storage for decimal numbers will be **bigger** than integer numbers but that's not a problem.

(Don't worry too much about the data type)

3 Accessing/ Changing specific elements, rows, columns...

[4]:
```
matrix = np.array([[1,2,3,4,5,6,7],[1,4,9,16,25,36,49]])

matrix
```

[4]:
```
array([[ 1,  2,  3,  4,  5,  6,  7],
       [ 1,  4,  9, 16, 25, 36, 49]])
```

[8]:
```
# Get a specific element
#================================

print("Element: ", matrix[1,5]) #row 1, col5


# Get a specific column
#================================

print("Column: ", matrix[:,1]) # row (:=all), col 1


# Get a specific row
#================================
```

```
print("Row: ", matrix[0,:]) # row 0, col (:=all)
```

```
Element:  36
Column:  [2 4]
Row:  [1 2 3 4 5 6 7]
```

[5]:
```
# If we need to change the stepsize...
#==================================

matrix[1, 0:6:2] # [startindex:stopindex:stepsize]
```

[5]:
```
array([ 1,  9, 25])
```

[6]:
```
# Change sth
#==================================

matrix[:,1] = 4
matrix[:,1] = [2,4]

matrix
```

[6]:
```
array([[ 1,  2,  3,  4,  5,  6,  7],
       [ 1,  4,  9, 16, 25, 36, 49]])
```

4 What happens with more than 2D...?

[7]:
```
# Create a 3d matrix
#==================================
matrix_3d = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])

print(matrix_3d)


# Get a specific element (6)
#==================================

print(matrix_3d[1,0,1]) #nmatrix, row, col


# Get a specific row or column
#==================================

matrix_3d[1,0,:] #nmatrix, row, col
```

```
[[[1 2]
  [3 4]]

 [[5 6]
```

```
  [7 8]]]
6
```

[7]:
```
array([5, 6])
```

5 Create different types of arrays

- Zero arrays
- One arrays
- Any Other Number
- Arrays (by copying shape)
- Identity matrix

[11]:
```
# Creating zero arrays
#================================
np.zeros(shape=(6,4))


# Creating one arrays
#================================
np.ones(shape=(2,3,2))


# Creating arrays with any other number
#================================
np.full(shape=(2,2), fill_value=25, dtype= np.int8)
```

[11]:
```
array([[25, 25],
       [25, 25]], dtype=int8)
```

[10]:
```
# Creating arrays with specific shape
#================================

print("Matrix_shape: {}\n".format(matrix.shape))


# Use function _like to copy the shape
#================================

print(np.full_like(a=matrix, fill_value=99))


# Same process for ones and zeros
#================================

print(np.zeros_like(a=matrix, dtype=np.float32))
print(np.ones_like(a=matrix))
```

```
Matrix_shape: (2, 7)

[[99 99 99 99 99 99 99]
 [99 99 99 99 99 99 99]]
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
[[1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1]]
```

[12]:
```
# Creating the identity matrix
#==================================

identity = np.identity(n=3) #ndim

identity
```

[12]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

6 Creating random arrays

- Random decimal numbers
- Random integer values
- Random arrays (by copying shape)

[14]:
```
# Create a random array with decimal numbers (size)
#==================================

rand = np.random.rand(2,2)
print("Decimal Numbers: \n", rand)


# Create a integer array (low_value, high_value, size)
#==================================

rand = np.random.randint(0, 10, size=(2,2))
print("Integer numbers: \n", rand)


# Create a random array by copying other array
#==================================

rand = np.random.random_sample(size=rand.shape)
print("Array like: \n", rand)
```

```
Decimal Numbers:
 [[0.71542859 0.37338753]
```

```
 [0.16186048 0.92030282]]
Integer numbers:
 [[0 5]
 [7 9]]
Array like:
 [[0.15785419 0.07315103]
 [0.86718952 0.00600566]]
```

7 It's all by now !

7.1 Session information

[17]:
```
from sinfo import sinfo

sinfo()
```

```
-----
numpy       1.19.2
pandas      1.1.5
sinfo       0.3.1
-----
IPython             7.19.0
jupyter_client      6.1.7
jupyter_core        4.7.0
jupyterlab          2.2.6
notebook            6.1.6
-----
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Windows-10-10.0.19041-SP0
8 logical CPU cores, Intel64 Family 6 Model 126 Stepping 5, GenuineIntel
-----
Session information updated at 2021-04-18 18:58
```