COMP2123 Programming Technologies and Tools

Lab 7.X. NumPy Library

Lv Ruyi 3035142560

Objective

At the end of this self-learning lab, you should be able to:

- Understand the simple usage of NumPy library in Python.
- Please note that this lab uses Python3 IDLE

Section 1. Background knowledge NumPy array

NumPy is a fundamental package in Python designed specifically for scientific computing. In this lab, we focus on its powerful function in dealing with N-dimensional arrays.

- In NumPy, the dimensions of an array are called axes. For example, the following array has only one axis. Note that the elements are separated by commas.

```
[1, 2, 3, 4, 5, 6, 7]
```

- The following array represents nine integers in a cube. How many axes does it have?

```
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
```

Answer: (change color to see)

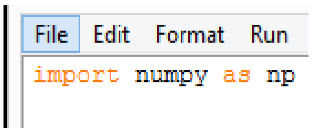
In NumPy, the array class is called **ndarray**. The elements in the **ndarray** are indexed by a tuple of positive integers. Here are some of the attributes of an **ndarrary** object.

ndarray.ndim	The number of axes of the array
ndarray.shape	A tuple of integers indicating the dimensions of the array
ndarray.size	The total number of elements in an array
ndarray.dtype	The type of elements in an array

Note: the **dtype** of the **ndarray** can be standard Python types, such as int, float or NumPy specific types, like numpy.int32, numpy.int16, and numpy.float64.

Try it yourself

- Open the Python file `~\numpy\Background.py`
- Remember the import function mentioned in the previous sessions? What does **np** mean in this context?
- We can see that a numpy array can simply be created using `np.array([content of the array])`.



```
import numpy as np
```

- Follow the instructions and run the module.
- Uncomment the last part and replace all `##replaceme##` untill all results are True.

Section 2. Creation and Basic Operations

In the previous section, we showed you that an array can be created using `np.array.` Here are some other ways to create a NumPy array:

- The `zeros` function creates an array full of zeros. For example, the following two commands have the same effect.

  ```
  np.zeros((2,3))
  ```
  ```
  np.array( [ [ 0., 0., 0. ], [ 0., 0., 0. ] ] )
  ```

  The dots at the end of the elements show that the **dtype** of the array is float. In this case, it is numpy.float64.
- The `ones` function works the same. It creates an array full of ones.
- Remember in previous lab we have learned the function `range`. NumPy provides a function `arange` that works the same with `range` except that it returns an array instead of a list. For example, the following two commands have the same effect.

  ```
  np.arange(0, 20, 5)
  ```
  ```
  np.array( [ 0, 5, 10, 15 ] )
  ```
- However, if the number of elements in an array is more important for us, we can use the function `linspace`. For example, the first command below specify that we want nine numbers from 0 to 2. It is equivelant to the second command below.

  Don't understand?
  Review the function
  `range`

  ```
  np.linspace(0, 2, 4)
  ```
  ```
  np.array( [ 0., 0.5, 1, 1.5, 2.] )
  ```
- When creating an array using the above methods, we can specify the datatype by simply adding the dtype argument. For example, if we want an array full of ones with shape (3,3) and the data type is integer, we can use the following command.

  ```
  np.ones( (3,3), dtype=int )
  ```

Arithmetic operations are applied on arrays elementwise. For example:

```
a = np.array( [ 0, 20, 5 ] )
```
```
b = np.array( [ 0., 10., 2.5 ] )
```
```
c = a-b
```

What is c? What is the type of the elements in c?

We can use print function to directly print out the array and the type of elements.

```
print ( c )
```

```
print ( a.dtype )
```

```
print ( b.dtype )
```

```
print ( c.dtype )
```

The result is:

```
[ 0. 10. 2.5]        int32                    float64                    float64
```

Q: We see that the plus function is applied to the two arrays elementwise, but why is the data type of c float64 instead of int32?

A: This is called upcasting. The type of the resulting array always corresponds to the more precise one.

How can we do matrix multiplication?

```
a = np.array([[5,6],[7,8]])
```

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

```
c = a*b
```

The result is:

```
[[ 5 12]
21 32]]
```

This is not what we want!
This is simply cross product element -wise

Remember in NumPy, if we want to do matrix multiplication, '*' is not the way to go! Instead we use one of the following:

```
c = a.dot(b)
```

```
c = np.dot(a,b)
```

The result is:

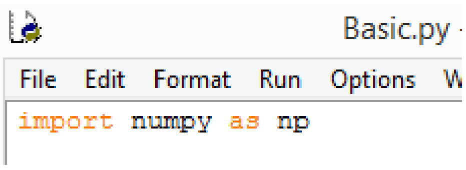
```
[[ 23 34]
31 46]]
```

This is the actual matrix multiplication
that we are looking for

Arithmetic operations also include **(power), <, > etc. We will now experiment with some of them.

Try it yourself

- Open the Python file ~\ `numpy\Basic.py`
- Follow the instructions to add the appropriate demand
- Does the final results match your guess?



```
import numpy as np
```

Section 3. Indexing, Slicing and Iterating

We have now learned how to create, print NumPy arrays and perform arithmetic operations on them. Now it is time for you to do some self-studying. The ability to look for information on your own is important and will be helpful in academic and industrial settings. Simply type "NumPy" in google and try to understand the following:

1. How the multi-dimensional arrays are indexed?
2. How can you slice an array given certain indexes?
3. How to iterate over part or whole of the array?

Checkpoint 7.X **(Please submit your answer to Moodle.)**

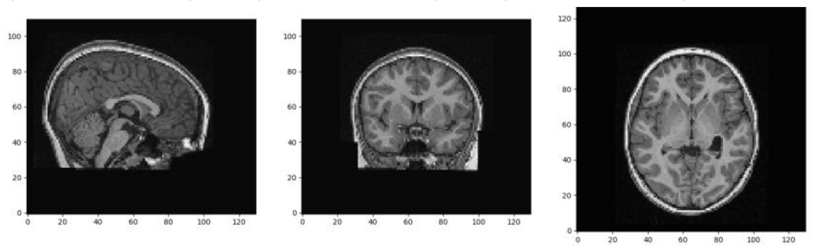
Open the Python file `~\numpy\MRI.py`
NumPy library is commonly used for data-preprocessing. The objective of this checkpoint is to edit the original brain image to a desired shape. The MRI image is represented by a NumPy array. The process includes two steps:

1. Crop the center for any dimension where the original is larger than the desired
2. Pad to desired shape for any dimension where the original is smaller than the desired

The main command has already been completed for you. Although you can use the completed show_slices function to do sanity check if necessary. What you need to do is to complete the crop_center and pad_todesire functions.

If you have done well, the module should output the shape change and the final comparison should be True. More explanations are in the source code.

For the small MRI image the shape change should be:
(65, 65, 55)--->(60, 60, 55)--->(60, 60, 60)
For the large MRI image the shape change should be:
(130, 130, 110)--->(120, 120, 110)--->(120, 120, 120)



Acknowledgement

- Special thanks to Kit for the template of this lab
- The MRI images are legally obtained through INDI sharing scheme.