Lecture 6 – Python Files + NumPy

Mr. Yousif G. Arshak

University of Zakho

Computer Science Department

yousif.arshak@uoz.edu.krd

OUTLINES

- Python File Handling
- Python NumPy

File Handling

- The key function for working with files in Python is the open() function.
- The open() function takes two parameters; filename, and mode.
- There are four different methods (modes) for opening a file:
  - "r" - Read - Default value. Opens a file for reading, error if the file does not exist
  - "a" - Append - Opens a file for appending, creates the file if it does not exist
  - "w" - Write - Opens a file for writing, creates the file if it does not exist
  - "x" - Create - Creates the specified file, returns an error if the file exists
    - "t" - Text - Default value. Text mode
    - "b" - Binary - Binary mode (e.g. images)

Open a File on the default location

- ```
  f = open(“mytext.txt", "r")
  print(f.read())
  ```
  ```

Open file from Different location

- ```
  f = open("D:\\mytext.txt", "r")
  print(f.read())
  ```

Read Only Parts of the File

- ```
  f = open(“mytext.txt", "r")
  print(f.read(5))
  ```

Read Lines

- You can return one line by using the readline() method:
  - ```
    f = open(“mytext.txt", "r")
    print(f.readline())
    ```

Read two lines of the file

  - ```
    f = open(“mytext.txt", "r")
    print(f.readline())
    print(f.readline())
    ```

Loop through the file line by line:

- ```
  f = open("demofile.txt", "r")
  for x in f:
    print(x)
  ```

Close Files

- It is a good practice to always close the file when you are done with it.
  - ```
    f = open(“mytext.txt", "r")
    print(f.readline())
    f.close()
    ```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Write to an Existing File

- To write to an existing file, you must add a parameter to the open() function:
  - "a" - Append - will append to the end of the file
  - "w" - Write - will overwrite any existing content
- Open the file "mytext.txt" and append content to the file:
  - ```
    f = open("demofile2.txt", "a")
    f.write("Now the file has more content!")
    f.close()
    ```
    ```

Open the file “mytext.txt" and overwrite the content:

- ```
  f = open(“mytext.txt", "w")
  f.write("Woops! I have deleted the content!")
  f.close()
  ```

Note: the "w" method will overwrite the entire file.
  ```

Create a file called "myfile.txt":

- `f = open("myfile.txt", "x")` #Result: a new empty file is created!

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

Delete a File

- To delete a file, you must import the OS module, and run its `os.remove()` function:

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:

- Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

What is NumPy?

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.

Why Use NumPy?

- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

Installation of NumPy

- If you have Python and PIP already installed on a system, then installation of NumPy is very easy.
  - Install it using this command:
  - C:\Users\yousif>pip install numpy

Note: If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy by using this command

- `import numpy`

- Example

```
arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

NumPy is usually imported under the np alias.

- `import numpy as np`

NumPy Creating Arrays

We can create a NumPy `ndarray` object by using the `array()` function.

- ```
  import numpy as np

  arr = np.array([1, 2, 3, 4, 5])

  print(arr)

  print(type(arr))
  ```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```
```

- To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:
  - ```
    import numpy as np
    arr = np.array((1, 2, 3, 4, 5))
    print(arr)
    ```
    ```

Create a 2-D array containing two arrays

- ```
  import numpy as np
  arr = np.array([[1, 2, 3], [4, 5, 6]])
  print(arr)
  ```

3-D arrays

- ```
  import numpy as np
  arr = np.array([[[1, 2, 3], [4, 5, 6]],
  [[1, 2, 3], [4, 5, 6]]])
  print(arr)
  ```

Check Number of Dimensions?

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3],
[4, 5, 6]]])

print(a.ndim) #0
print(b.ndim) #1
print(c.ndim) #2
print(d.ndim) #3
```

Access Array Elements

- Array indexing is the same as accessing an array element.
- You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
  - ```
    import numpy as np
    arr = np.array([1, 2, 3, 4])
    print(arr[0])
    ```
    ```

Access the 2nd element on 1st dim:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st dim: ', arr[0, 1])
```

NumPy Array Slicing

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [*start:end*].
- We can also define the step, like this: [*start:end:step*].
  - ```
    import numpy as np

    arr = np.array([1, 2, 3, 4, 5, 6, 7])

    print(arr[1:5])
    ```

Note: The result *includes* the start index, but *excludes* the end index.
    ```

STEP

- Use the `step` value to determine the step of the slicing:
- Return every other element from index 1 to index 5:
  - ```
    import numpy as np

    arr = np.array([1, 2, 3, 4, 5, 6, 7])

    print(arr[1:5:2]) #[2 4]
    ```
    ```

Converting Data Type on Existing Arrays

The `astype()` function creates a copy of the array, and allows you to specify the data type as a parameter.

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

NumPy Array Shape

- The shape of an array is the number of elements in each dimension.
- Example
  - Print the shape of a 2-D array:
  - ```
    import numpy as np
    arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
    print(arr.shape) #(2, 4)
    ```
    ```

NumPy Array Reshaping

- By reshaping we can add or remove dimensions or change number of elements in each dimension.
- Example
  - Convert the following 1-D array with 12 elements into a 2-D array.
  - ```
    import numpy as np
    arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
    newarr = arr.reshape(4, 3)
    print(newarr)
    ```
-

```
[[ 1 2 3]
 [ 4 5 6]
 [ 7 8 9]
 [10 11 12]]
```
```

NumPy Joining Array

- Joining means putting contents of two or more arrays in a single array.
- Example: Join two arrays
  - ```
    import numpy as np

    arr1 = np.array([1, 2, 3])

    arr2 = np.array([4, 5, 6])

    arr = np.concatenate((arr1, arr2))

    print(arr) #[1 2 3 4 5 6]
    ```
    ```

NumPy Splitting Array

- Example
- Split the array in 3 parts:
-
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr)
```

Note: The return value is an array containing three arrays.

NumPy Searching Arrays

- You can search an array for a certain value, and return the indexes that get a match.
- To search an array, use the where() method.
- Example: Find the indexes where the value is 4:
  - ```
    import numpy as np

    arr = np.array([1, 2, 3, 4, 5, 4, 4])

    x = np.where(arr == 4)

    print(x) #(array([3, 5, 6]),)
    ```
    ```

Random Numbers in NumPy

- NumPy offers the random module to work with random numbers.
- Example: Generate a random integer from 0 to 100:
  - ```
    from numpy import random

    x = random.randint(100)

    print(x)
    ```
    ```

Generate Random Float

- The random module's rand() method returns a random float between 0 and 1.
- Example: Generate a random float from 0 to 1:
  - ```
    from numpy import random

    x = random.rand()

    print(x)
    ```
    ```

Generate Random Array

- Example: Generate a 1-D array containing 5 random integers from 0 to 100:
  - ```
    from numpy import random

    x=random.randint(100, size=(5))

    print(x) #[3 71 43 78 26]
    ```
    ```

- Example
- Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:
  -
```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

- Example: Generate a 1-D array containing 5 random floats:
  - ```
    from numpy import random
    x = random.rand(5)
    print(x)
    ```

- Example: Return one of the values in an array:
  - ```
    from numpy import random
    x = random.choice([3, 5, 7, 9])
    print(x)
    ```