Introduction to Python

Sana Rasheed

AL NAFI,
A company with a focus on education,
wellbeing and renewable energy.

Section Four

Overview – Part 4

- Installing Modules
- Parsing XML with LXML
- Config parser
- Threading
- Numpy

Installing Modules

- Apart from built-in modules, Python also allows import external modules.
- Python is open-source, so there are abundance of modules out there for just about anything.
- For instance:
  - pygame - If you want to develop games.
  - pandas - If you want to work with data manipulation.
  - numpy - If you want to do intensive numerical calculations.
  - selenium - If you are interested in web automation.
- These modules do not come pre-installed with Python but a simple import statement will do the job.

Installing Modules

- Before you import you need to conduct installation. This can be done either via:
  - Command Prompt
  - Anaconda Prompt
  - Anaconda Interactive
- For Command prompt you need to use:
  - pip install [library name]
- For Anaconda prompt, you need to use:
  - conda install [library name]
- For Anaconda Interactive, you can simply select the available library and click install.

Installing Modules

Installing the library "pygame" through cmd prompt (Windows).

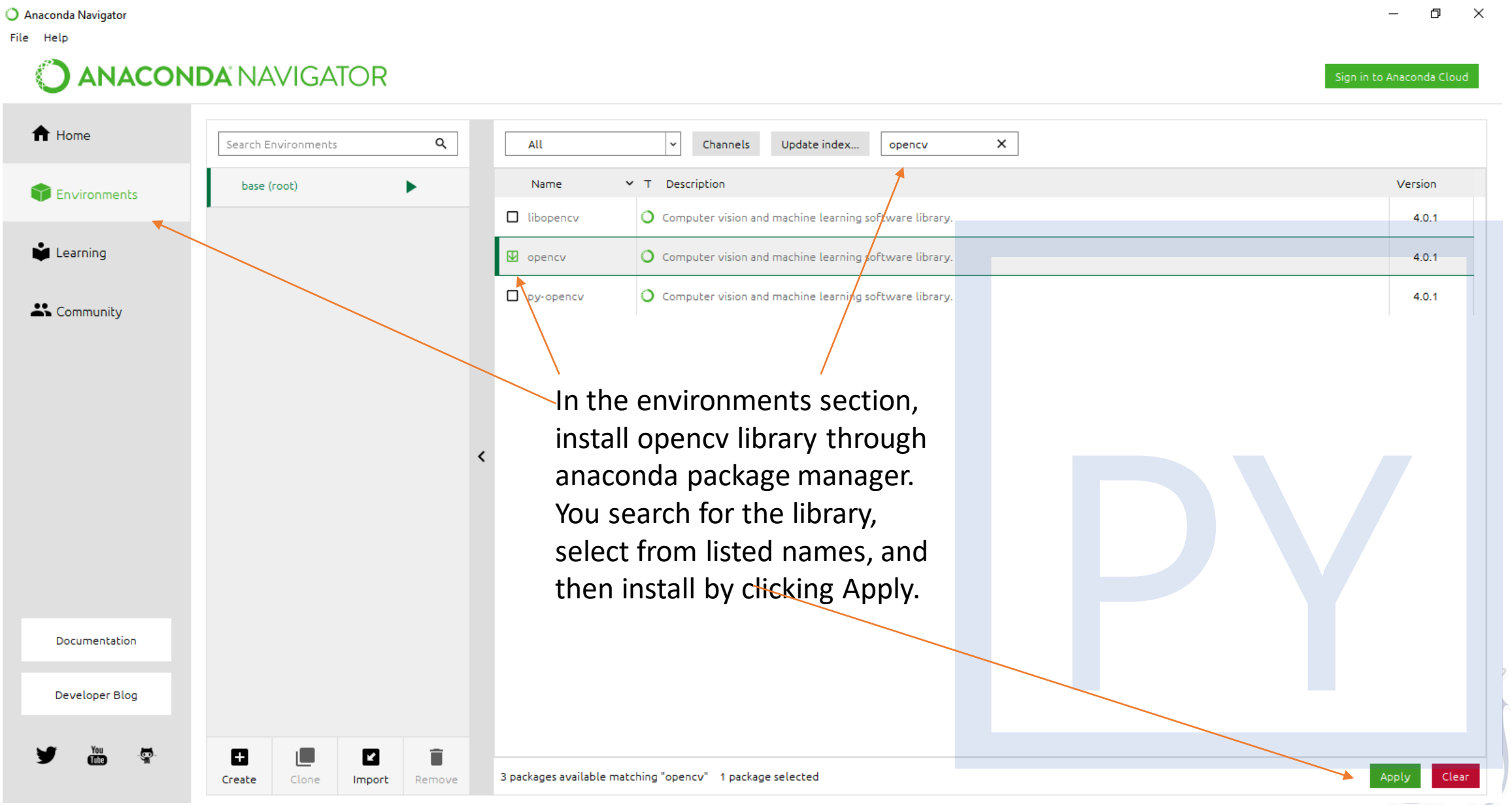
Command Prompt

```
Microsoft Windows [Version 10.0.17763.914]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\USER>pip install pygame
Collecting pygame
  Downloading https://files.pythonhosted.org/packages/ed/56/b63ab3724acff69f4080e54c4bc5f55d1fbdeeb19b92b70acf45e88a5908
/pygame-1.9.6-cp37-cp37m-win_amd64.whl (4.3MB)
     |                                | 4.3MB 467kB/s
Installing collected packages: pygame
Successfully installed pygame-1.9.6
```

Installing “selenium” through anaconda prompt.

Anaconda Prompt (Anaconda3)

```
(base) C:\Users\USER>conda install selenium
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\USER\Anaconda3

  added / updated specs:
    - selenium


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.8.2                |           py37_0         2.8 MB
    selenium-3.141.0           |   py37he774522_0         805 KB
    ------------------------------------------------------------
                                           Total:         3.6 MB

The following NEW packages will be INSTALLED:

  selenium           pkgs/main/win-64::selenium-3.141.0-py37he774522_0

The following packages will be UPDATED:

  conda                                     4.8.1-py37_0 --> 4.8.2-py37_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
conda-4.8.2          | 2.8 MB    | ################################################################################ | 100%
selenium-3.141.0     | 805 KB    | ################################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\USER>_
```



In the environments section, install opencv library through anaconda package manager. You search for the library, select from listed names, and then install by clicking Apply.

Installing Modules

EXERCISE – 1

- Try installing the following packages using either of the available choices (Anaconda, CMD Prompt, Anaconda Prompt):
  - pandas
  - scipy
  - numpy
  - scikit-learn

LXML Module

- Extensible Markup Language (**XML**) is a markup language that defines a set of rules for encoding documents in a format that is human-readable and machine-readable.
- Think structured webpages, catalogues, and more. Everything is parsed together with XML. It is used for describing how the content of a webpage will look in a webpage.
- In Python you can extract information from an XML page using a built-in library lxml.

LXML Module

- Let's consider an xml file that contains a catalogue for books. The entire catalogue is parsed within <catalogue></catalogue> tag.
- Within the catalogue, we have three books. Each book has the following descriptors (encapsulated in their own tags):
  - ID
  - Author
  - Title
  - Genre
  - Price
  - Publishing year
  - Description
- Source: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms762271%28v=vs.85%29

LXML Module

author, title, genre, etc. are contained within their own designated <></> tag.

```
<catalog>
    <book id="bk101">
        <author>Carol, Lewis</author>
        <title>Alice in Wonderland</title>
        <genre>Fiction</genre>
        <price>34.95</price>
        <publish_year>1875</publish_year>
        <description>A juvenile fiction story.</description>
    </book>
    <book id="bk102">
        <author>Doyle, Arthur Conan</author>
        <title>Sherlock Holmes</title>
        <genre>Mystery</genre>
        <price>2.95</price>
        <publish_year>1887</publish_year>
        <description>Solve mysteries with the detective Sherlock Holmes and Dr. Watson.</description>
    </book>
    <book id="bk103">
        <author>Austen, Jane</author>
        <title>Pride and Prejudice</title>
        <genre>Romantic Comedy</genre>
        <price>1.95</price>
        <publish_year>1813</publish_year>
        <description>A timeless story following the character of Elizabeth Bennet</description>
    </book>
</catalog>
```

Each book is contained within its own <book></book> tag. Hence, the details are easily extractable.

LXML Module

- Our objective is to read the content of the XML file and get the respective information, while maintaining a dictionary to access information later.
- We first read the object file into Python, and use a library method lxml.etree.fromstring() to convert the read file into an LXML object.
- Since the entire lxml structure is referred to as a “tree” and the outer most tag is called “root”, we can access its branches or “children”, using the getchildren() method.
- How that works is we obtain the children nodes, and iterate over each of those elements. Each element then contains “tags” that we can use to access information for each of the book’s descriptors.

LXML Module

The etree object parses the entire lxml file into a tree structure, with each tag as a branch, and nested tags become its children or branches.

The file is read using python internal IO methods and then converted to string using etree.

```
1   from lxml import etree
2
3   def parseBookXML(xmlFile):
4
5       with open(xmlFile) as fobj:
6           xml = fobj.read()
7       root = etree.fromstring(xml)
8       book_dict = {}
9       books = []
10      for book in root.getchildren():
11          for elem in book.getchildren():
12              if elem.text:
13                  text = elem.text
14              else:
15                  text = ''
16              if elem.tag == 'author':
17                  last_name, first_name = text.split(',')
18                  print(elem.tag + ':', first_name, last_name)
19              else:
20                  print(elem.tag + ": " + text)
21              book_dict[elem.tag] = text
22          if book.tag == "book":
23              books.append(book_dict)
24              book_dict = {}
25      return books
26
27  my_books = parseBookXML("test.xml")
```

The .getchildren() accesses the sub-tags within a tree. The method called on root gives access to the book tag. The method called on book gives access to the contents of the book (author, genre, etc).

elem.text accesses the text inside each of the tags whereas, tag gives the tag names such as author, price, etc.

The entire information is collected and stored in a variable my_books.

LXML Module

```
E:\Projects\Sana\Course - Python>python section_four.py
author:  Lewis Carol
title: Alice in Wonderland
genre: Fiction
price: 34.95
publish_year: 1875
description: A juvenile fiction story.
author:  Arthur Conan Doyle
title: Sherlock Holmes
genre: Mystery
price: 2.95
publish_year: 1887
description: Solve mysteries with the detective Sherlock Holmes and Dr. Watson.
author:  Jane Austen
title: Pride and Prejudice
genre: Romantic Comedy
price: 1.95
publish_year: 1813
description: A timeless story following the character of Elizabeth Bennet
```

The output generated from the earlier script

LXML Module

EXERCISE - 2

- https://www.w3schools.com/xml/simple.xml
- An xml tree is contained in the file simple.xml. It contains breakfast menu information: name, price, calories and food description.
- Extract all information and store price, calories and description against food name.

ConfigParser Module

- ConfigParser is a handy module, useful for structuring configuration files.
- These configuration files can be used by your applications to read or write settings as required.
- It follows a simple structure making it easy to align settings in a readable manner.
- The structure is similar to that of INI files by Windows.
- It can create configuration files directly through dictionaries by automatically parsing files read as strings.

ConfigParser Common Usage

- The most common use for a configuration file is to have a user or system administrator edit the file with a regular text editor to set application behavior defaults, and then have the application read the file, parse it, and act based on its contents.

ConfigParser INI file

- A sample configuration file with section “bug_tracker” and three options would look like:

- [bug_tracker]
- url = http://localhost:8080/bugs/
- username = dhellmann
- password = SECRET

ConfigParser INI file

```
#######################
### Read INI file ###
#######################
import configparser

config = configparser.ConfigParser()
config.read("C:\\Users\sana.rasheed\.spyder-py3\spyder.ini")
config.sections()
```

ConfigParser Module – Read Dictionary

```
1   import configparser
2
3   parser = configparser.ConfigParser()
4   parser.read_dict(
5       {'section1':
6           {'tag1': '1','tag2': '2','tag3': '3'},
7       'section2':
8           {'tagA': 'A','tagB': 'B','tagC': 'C'},
9       'section3':
10          {'foo': 'x','bar': 'y','baz': 'z'} })
11  parser.sections() # ['section1', 'section2', 'section3']
12  [option for option in parser['section3']] # ['foo', 'bar', 'baz']
13
```

create an instance of configparser and read a dictionary using read_dict() for the configuration you wish you create.

Here, our file contains three sections section1, section2 and section3. Within each section, we have three specific settings tag1, tag2, tag3.

The parser object then automatically creates sections which are then accessible through .sections() method.

ConfigParser Module – String Type File

```
13
14  sample_config = """
15  [My Settings]
16  user = username
17  profile = /my/directory/to/profile.png
18  gender = male
19  """
20  # creates an instance of ConfigParser called config
21  config = configparser.ConfigParser()
22
23  # read the instance of string using read_string() method
24  config.read_string(sample_config)
25
26  config.sections() # ['My Settings']
27  config["My Settings"]["user"] # 'username'
```

Here, we try to read another setting in a different format (this time, as a string). The parser is able to identify sections (specified by [] and settings separated by = and \n

The settings within the parser object can be accessed like a nested dictionary.

ConfigParser Module

EXERCISE - 3

- Your team has just finished up creating a software product for a marketing company. The software reports summary statistics for data plugged in.
- You are required to create a configuration file that takes the following infor mation for the software to pick up and use
  - root directory
  - Username
  - number of databases connected.
  - summary unit [percent or absolute values]
- Create your own default values. Save your file as configuration.txt. Use the appropriate library.

Threading

- Python interpreter executes code in a line by line fashion. That means, it cannot move ahead unless it is done executing a line.
- What if you had to simultaneously run more than one code executions through a single script?
- A thread is a flow of execution. It is possible for your program to run more than one independent tasks at once, through “threading”.
- However, depending on the Python implementation, it may or may not support threading but rather appear to run simultaneously.

Threading

- threading is a built-in library in Python that allows the functionality of working with threads.
- We will create two methods, get_cubic() to cube the input and get_inverse() to invert the input.
- Using the Thread method, we pass the method and its respective arguments.
- This runs both the arguments simultaneously. Simple right?

Threading

```python
import logging
import threading
import time
import threading

def get_cubic(num):
    print("Cube: {}".format(num * num * num))

def get_inverse(num):
    print("Inverted: {}".format(1 / num))

if __name__ == "__main__":
    # creating thread
    t1 = threading.Thread(target=get_cubic, args=(10,))
    t2 = threading.Thread(target=get_inverse, args=(10,))
    # Starting the threads
    t1.start()
    t2.start()
    # waiting for each to finish
    t1.join()
    t2.join()
    # both threads completely executed
    print("Done!")
```

Target takes the function to run in the respective thread. The arguments to the function are passed separately in args as a tuple.

The .start() initiates the thread execution. Whereas, .join() holds the interpreter for each of the thread to finish.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
E:\Projects\Sana\Course - Python>python section_four.py
Cube: 1000
Inverted: 0.1
Done!
```

Both the methods are run simultaneously, however, it is hard to decipher that since the print statements are still sequential. To understand this better, we try to trip up one thread with an error and observe the results.

Threading

- The .start() method initiate the thread and run the respective methods. Whereas, the .join() method wait for the thread to finish before interrupting the flow of the code.
- That does not say much about threading. However, if in case of 10, a numeric value, you pass ’10’, a string value to the get_inverse() method, it should throw and error and break the moment t2.start() is executed.
- Since we have used threading, therefore, the code will not blow up rather, await ending the thread, throw the error in between but continue to execution until the done statement.

Threading

```
29  import logging
30  import threading
31  import time
32  import threading
33
34  def get_cubic(num):
35      print("Cube: {}".format(num * num * num))
36
37  def get_inverse(num):
38      print("Inverted: {}".format(1 / num))
39
40  if __name__ == "__main__":
41      # creating thread
42      t1 = threading.Thread(target=get_cubic, args=(10,))
43      t2 = threading.Thread(target=get_inverse, args=('10',))
44      # Starting the threads
45      t1.start()
46      t2.start()
47      # waiting for each to finish
48      t1.join()
49      t2.join()
50      # both threads completely executed
51      print("Done!")
```

Here, a string argument is passed to get inverse, which is bound to throw type error. However, since we are using threading, the code will not break entirely but continue to execute for the other thread, that is the get_cubic function.

Threading

```
E:\Projects\Sana\Course - Python>python section_four.py
Cube: 1000
Exception in thread Thread-2:
Traceback (most recent call last):
  File "C:\Users\USER\AppData\Local\Programs\Python\Python37\lib\threading.py", line 926, in _bootstrap_inner
    self.run()
  File "C:\Users\USER\AppData\Local\Programs\Python\Python37\lib\threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "section_four.py", line 38, in get_inverse
    print("Inverted: {}".format(1 / num))
TypeError: unsupported operand type(s) for /: 'int' and 'str'

Done!

E:\Projects\Sana\Course - Python>
```

Threading

- Threading is a powerful way to streamline your tasks in Python that can let your program run without potential error breaks.
- Although, if used recklessly, it can impact the speed, efficiency and structure of your code, rendering the overall usage of threading pointless.
- Key takeaway is to use threading where necessary and with caution.

Threading

EXERCISE – 4

- You are required to conduct analysis on data obtained from a plant related to temperature readings.
- The file temperatures.csv contains the appropriate data required for this task.
- Write methods to generate:
  - Average time lapse in readings
  - Average per hour temperature recording
  - Largest spike in data (percentage)
  - Overall deviation in the data
- Conduct the above calculations through two threads (each thread conducts two calculations).
- Measure up time with and without threads. Log your results in section_four/temperature.txt using the appropriate library.

Numpy Module

- Computers make it very easy to perform complex computation in a matter of seconds.
- Python has a dedicated library for numeric computation known as numpy.
- The applications of numpy are diverse but here we will focus on matrices multiplication.
- At the base of it all, it uses the numpy.my_array data type that can function as a vector or matrix of any dimension.
- Once your data is in the my_array form, you can then perform a multitude of operations on them.

Numpy Module

- Before we move further, let's have a look at numpy my_array first.
- Numpy my_array is very similar to a list but has added operations applicable on it such as multi-indexing which lets you slice and subset wrt columns and rows.
- In the following table are some ways to slice and subset numpy my_arrays.

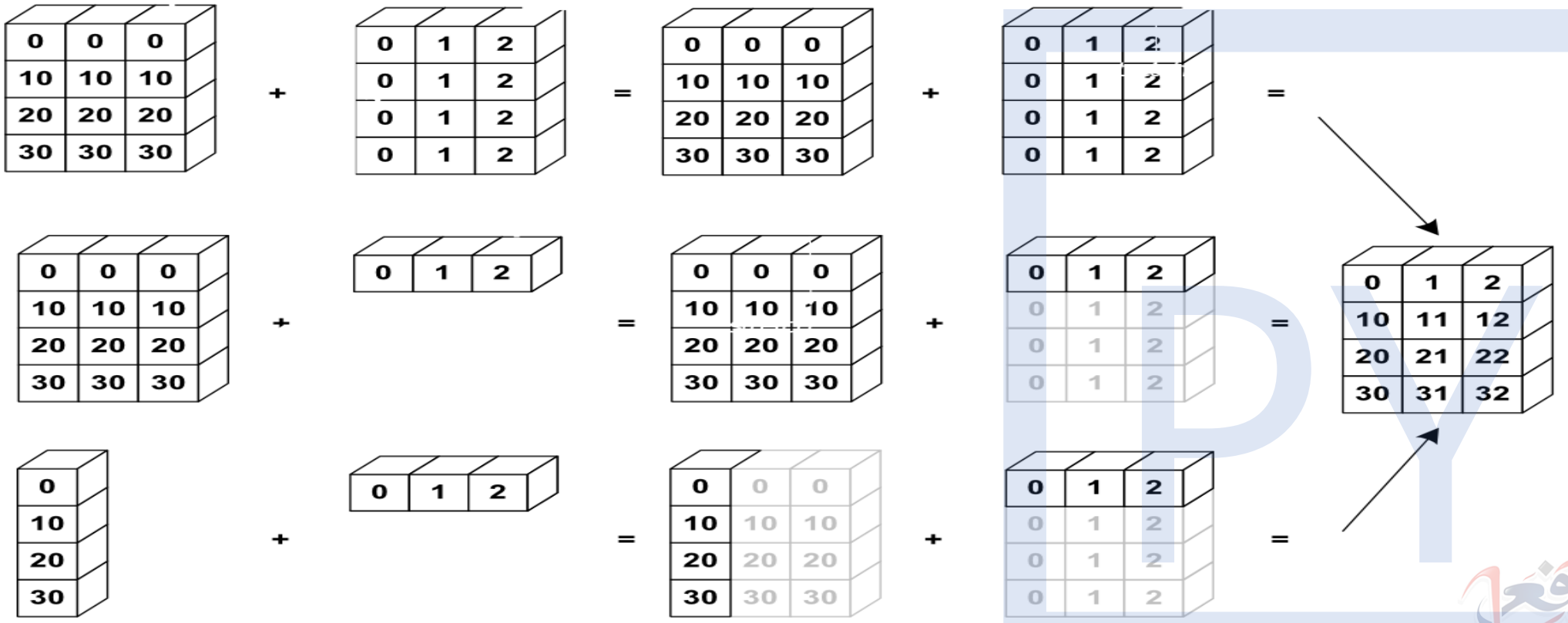
Numpy Module

Operator	Description
my_array[i]	1d my_array at index i
my_array[i,j]	2d my_array at index[i][j]
my_array[i<4]	Boolean Indexing, see Tricks
my_array[0:3]	Select items of index 0, 1 and 2
my_array[0:2,1]	Select items of rows 0 and 1 at column 1
my_array[:1]	Select items of row 0 (equals my_array[0:1, :])
my_array[1:2, :]	Select items of row 1
my_array[: :-1]	Reverses my_array

Numpy Module

- Now that we are familiar with how to interact with an array let's move forward.
- We are going to try to create matrices with numpy and then attempt some calculations on them.
- Numpy my_array have another interesting property known as broadcasting.
- If you have two my_arrays of the same size, basic arithmetic operations occur then on element-wise basis.

Numpy Module



Picture Credits: Scipy.docs

Numpy Module

- Let’s try to perform some basic operations on numpy array such as:
  - Creating random sequence
  - Creating patterned sequence
  - Creating 1D and 2D arrays
  - Finding minimum and maximum values
  - Finding indices of minimum and maximum values
  - Creating identity and diagonal matrices
  - Performing transpose and dot products

Numpy Module

```
53  import numpy as np
54
55  mat = np.array([[1,2,3], [4,5,6], [7,8,9]])
56  print('The shape of mat is:', mat.shape)
57  # it can be reshaped to any shape that is consistent with the data size
58  new_mat = mat.reshape(1,9)
59  print('The shape of new_mat is:', new_mat.shape)
60  # you can try transposing it as well.
61  trans_new_mat = new_mat.T
62  print('The transposed shape of new_mat is:', trans_new_mat.shape)
63
64  # you can create diagonal matrices, identity matrices
65  diag = np.diag([1,2,3,4])
66  identity = np.identity(10) # creates a 10 x 10 matrix with ones at diagonals and rest zeros
67
68  # you can create arrays of random numbers or sequences
69  seq = np.arange(start=0, stop=100, step=5) # creates an array starting from 0, with a step size of 5 till 100. [0,5,10,....,90,95]
70  # If you only pass np.random.rand(10), it creates a 1D vector of length 10
71  rands_1d = np.random.rand(10)
72  rands_2d = np.random.rand(10, 10) # creates 10x10 matrix with random numbers.
73
74
75  # you can find min and max and min as well as the index of max number or min number
76  my_array = np.array([1,3,5,7,23,6,2,31,5])
77  print('The max is {} at index {}'.format(my_array.max(), my_array.argmax()))
78  print('The min is {} at index {}'.format(my_array.min(), my_array.argmin()))
79
80  import time
81  # you can perform dot products of matrices in a matter of seconds
82  my_mat = np.random.rand(1000, 1000)
83  another_mat = np.random.rand(1000, 1000)
84  start = time.time()
85  # make sure the dimensions agree!
86  dot_mul = my_mat.dot(another_mat)
87  end = time.time()
88  print('It took {} seconds to calculate dot product!'.format(round(end-start, 4)))
```

Numpy Module

- Within numpy there is a sub-module linalg, short for linear algebra, that contains most commonly used linear algebra functions such as:
  - Determinant of a matrix
  - Inverse of a matrix
  - Cholesky factorization of a matrix
  - Eigen values of a matrix
  - Rank of a matrix

- Let’s have a look at how that works.

Numpy Module

```
# linear algebra
import numpy.linalg as la

my_mat = np.array([1,2], [3,4])
# calculate determinant
det = la.det(my_mat)
# calculate inverse of a matrix. Inv = adj(matrix)/det(matrix). NOTE: Make sure its not SINGULAR!
inverse_mat = la.inv(my_mat)
# calculate eigen values. NOTE: make sure the matrix is SQUARE [eigen values do not exit for matrix whose dimensions are mxn where m!=n]!
eig_values, eig_vectors = la.eig(my_mat)
# calculate cholesky factorization. NOTE: make sure matrix is Positive Definite!
cholesky = la.cholesky(np.array([[7,2], [2,1]]))
# calculate rank of a matrix. Rank of a matrix is the number of linearly independent columns/rows in a matrix.
rank = la.matrix_rank(my_mat)
```

Numpy Module

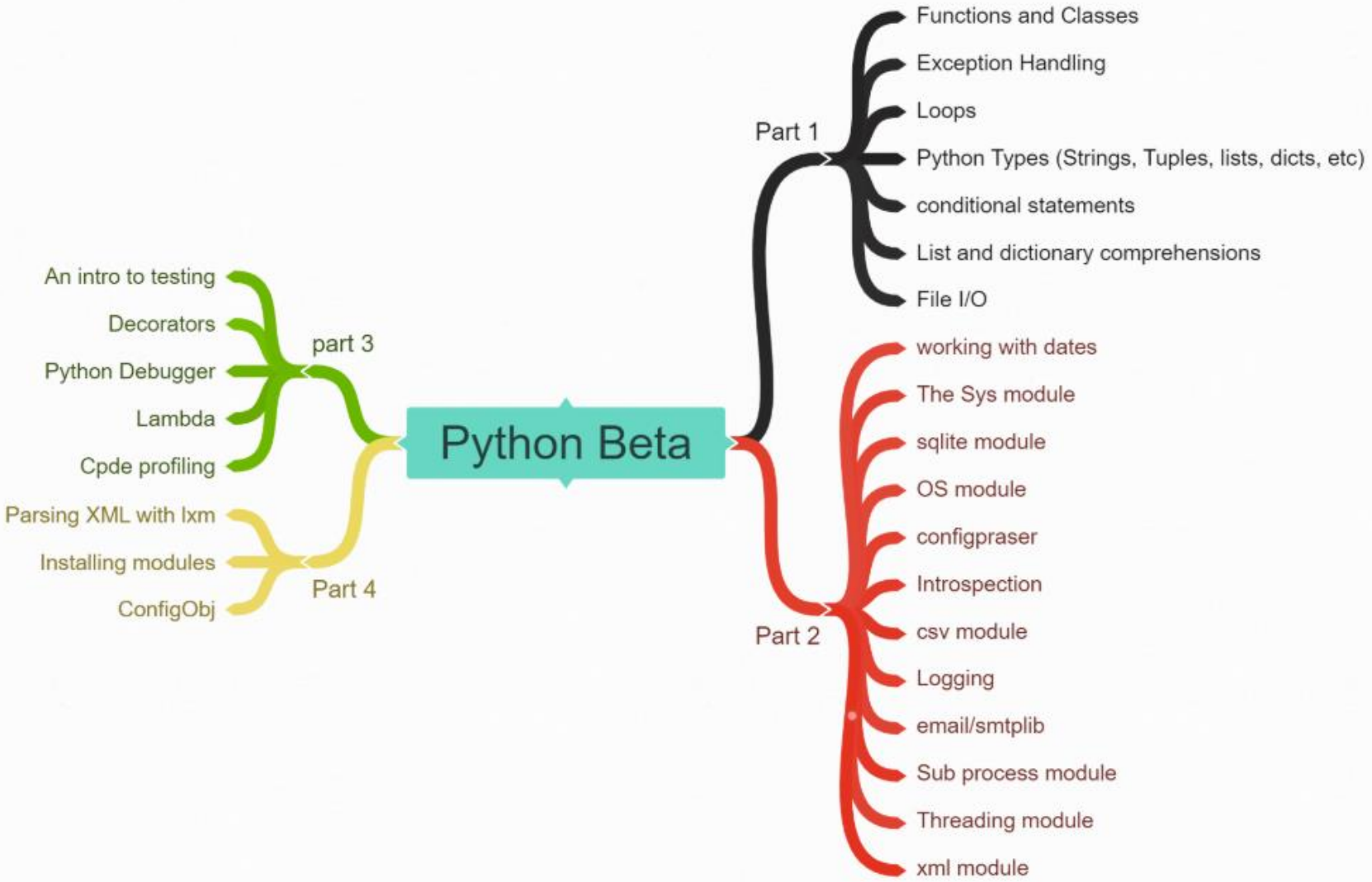
EXERCISE – 5

- Write a NumPy program to extract upper triangular part of a NumPy matrix
- Write a NumPy program to extract all the elements of the second and third columns from a given (4x4) my_array
- Write a NumPy program to count the occurrence of a specified item in a given NumPy my_array
- Write a NumPy program to sum and compute the product of a NumPy my_array elements

Recap

- Installing Modules
- Parsing XML with LXML
- Config parser
- Threading
- Numpy

Python Beta Track



Basic Introduction

Built-in Modules

Testing and Debugging

Module Installation and Configuration

جزاك اللهُ

To ask questions, Join the Al Nafi Official Group

https://www.facebook.com/groups/alnafi/

(This group is only for members to ask questions)