Introduction on

Advanced Python Programming II

Kamal

AptComputingAcademy

Contents

Database communication
XML Parser
Web Scrapping
PyUnit
PySpark

Database communication (MySQL)

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Using “setUpClass” and “tearDownClass” for entire test suite:

```
class TestSuite01(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print("\n\n=======TestSuite01: Class method:
                    Setup========\n\n")
    @classmethod
    def tearDownClass(cls):
        print("\n\n=======TestSuite01: Class method:
                    TearDown========\n\n")
    def setUp(self):
        print("*** TestSuite01: Going to setup for
                        testcase ***")
    def tearDown(self):
        print("*** TestSuite01: Going to tear down after
                        testcase ***")
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

MySQL DataBase Management:

Most popular, Open source,

1) Install mysql server on ubuntu

2) Working with mysql-connector in Python env:

(https://www.w3schools.com/python/python_mysql_delete.asp)

- Connect to mysql
- Create data base
- Create table
- Insert row, Select all from table
- Where
- Sort the rows
- Delete/Update row and drop table
- Join tables

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

MySQL DataBase Management:

- Most popular, Open source
- Multi-User and Multi-Threaded database management
- Popular on web
- Runs on BSD Unix, Linux, Windows and MAC
- Wikipedia, Facebook and Youtube uses MySQL

345 systems in ranking, September 2018

Rank			DBMS	Database Model	Score		
Sep 2018	Aug 2018	Sep 2017			Sep 2018	Aug 2018	Sep 2017
1.	1.	1.	Oracle	Relational DBMS	1309.12	-2.91	-49.97
2.	2.	2.	MySQL	Relational DBMS	1180.48	-26.33	-132.13
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1051.28	-21.37	-161.26
4.	4.	4.	PostgreSQL	Relational DBMS	406.43	-11.07	+34.07
5.	5.	5.	MongoDB	Document store	358.79	+7.81	+26.06

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Installing 'mysql' server in UBUNTU:

(https://support.rackspace.com/how-to/installing-mysql-server-on-ubuntu/)

Install MySQL:

```
sudo apt-get update
sudo apt-get install mysql-server
```

Allow remote access:

```
sudo ufw allow mysql
```

Start the MySQL service

```
systemctl start mysql
```

Launch at reboot

```
systemctl enable mysql
```

Install security previleges:

```
mysql_secure_installation
```

Start the mysql shell

```
/usr/bin/mysql -u root -p
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Creating new user to access from remote:

Configure my.cnf:

/etc/my.cnf: Comment the line “bind-address=YOUR-SERVER-IP”

Create user with secure privileges:

Open mysql and create the user as shown below:

```
CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';
GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost' WITH GRANT OPTION;
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Creating new user to access from remote:

Configure my.cnf:

/etc/my.cnf: Comment the line “bind-address=YOUR-SERVER-IP”

Create user with secure privileges:

Open mysql and create the user as shown below:

CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';

GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost' WITH GRANT OPTION;

```
mysql> CREATE USER 'ittstarz'@'%' IDENTIFIED BY 'Bond@321';

mysql> grant all privileges on *.* to 'ittstarz'@'%';

mysql> grant all privileges on `database_name`.`table_name` to
'ittstarz'@'%';
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Connect to mysql

```
#1) Importing mysql connector
import mysql.connector

#2) Connect to mysql server using user credentials
db = mysql.connector.connect(host="192.168.43.26",
                             user='ittstar',
                             password='Bond@321')
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Create database and delete

```
#2) Connect to mysql server using user credentials
db = mysql.connector.connect(host="192.168.43.26",
                             user='ittstarz',
                             password='Bond@321')
#3)Creating database
mycursor = db.cursor();
mycursor.execute("CREATE DATABASE `ittstars_01`")
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Create table and delete table

```
#2) Connect to mysql server using user credentials
db = mysql.connector.connect(host="192.168.43.26",
                             user='ittstarz',
                             password='Bond@321',
                             database = 'ittstars_01')
#3)Creating database
mycursor = db.cursor();
#4)Create table
mycursor.execute("CREATE TABLE customers
(name VARCHAR(255), address VARCHAR(255),
designation VARCHAR(255))")
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Insert row, get and delete

```
#5) Insert row
sqlcmd = "INSERT INTO customers
(name, address,designation) VALUES (%s, %s, %s)"
vals = ("Nithin", "Bangalore", "Sr.Engineer")
mycursor.execute(sqlcmd, vals)
db.commit()
mycursor.execute("drop TABLE customers")
db.commit()


#6) Get all the rows
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)

#Delete the row
sql = "DELETE FROM customers WHERE address = 'Bangalore'"
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Search for row

```
#7) Get a particular entry
sqlcmd = "SELECT * FROM customers where
          address='Bangalore'"
mycursor.execute(sqlcmd)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Order

```
Order by field ascend order:
sqlcmd = "SELECT * FROM customers ORDER BY name"

Order by field descend order:
sql = "SELECT * FROM customers ORDER BY name DESC"
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Update row:

```
#9) Update
sqlcmd = "update customers set
          address='Chennai' where name='Kamal'"
mycursor.execute(sqlcmd)
db.commit()
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Working with mysql-connector in Python env:

- Set limit

```
Sqlcmd = "SELECT * FROM customers LIMIT 5"

Sqlcmd = "SELECT * FROM customers LIMIT 5 OFFSET 2"
```

Web-Scraping

Using Beautiful Soup

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Web scraping:

Extract the data from web and process it to make formatted data.

Required packages:

- **beautifulsoup4**
- **urllib**

Bring up setup:

- pip install bs4

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Programming:

1) Import packages
2) Get url content
3) Read the page
4) Read a class (depending on interested area)

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Importing modules:

```
#1) Import required modules
from bs4 import BeautifulSoup as soup
from urllib.request import urlopen
```

Reading URL page:

```
url = "https://www.cricbuzz.com/live-cricket-scores/
       20751/ind-vs-ban-final-asia-cup-2018"
html = urlopen(url)
page = html.read()
soup_obj = soup(page, 'lxml')
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Read all the data:

```
filter_data = soup_obj.findAll('span',
{"class":"cb-font-20 text-bold"})
```

Parse for the required data:

```
import re
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
matcher = re.compile(r'BAN (\d*)/(\d*) \((.*) Ovs\)')
```

XML-Parsing

https://docs.python.org/2/library/xml.etree.elementtree.html

https://www.tutorialspoint.com/online_xml_editor.htm
(To understand XML file)

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Xml Parsing:

Extract the data from xml files.

-> XML is structured data as JSON

-> Not a programming language

-> Use tags, no standard tags

```
<root>
    <child>
        <subchild>
            <Attr1>”Value1”</
Attr1>
        </subchild>
    </child>
</root>
```

```
<?xml version="1.0"?>
<head>
    <employee>
        <name>" Kamal Kumar "</name>
        <address> "yelahanka, bangalore "</address>
    </employee>
</head>
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Xml Rules:

-> Must have opening and closing tags

-> Tags are case sensitive

-> Should be nested properly

-> Must have root tag

-> Attributes must have quotes

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Required packages:

- resquest (To download web page)
- xml.etree.ElementTree (in built)

Bring up setup:

- pip install requests

Practice:

1) Read xml file, read each element and attributes
2) Write new xml file
3) Add new elements to xml file
4) Update elements
5) Read the data from web with the help of BeautifulSoup

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Importing module and reading xml file:

```
#1) import the class ElementTree
import xml.etree.ElementTree as et
file = "SampleXml.xml"

#2) Open xml file using 'et' and get element tree
tree = et.parse(file)
print(tree)
```

Reading each element:

```
#3) Get root element
root = tree.getroot()
print(root)
print(root.text)

#4) Get each element from the tree
for child in root:
    print(child)
```

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Getting attributes of elements:

```
#4) Get each element from the tree
for child in root:
    print(child.attrib)
    for subchild in child:
        print(subchild.tag)
        print(subchild.attrib)
```

Finding interesting values:

```
#5) Getting interesting elements
for item in root.iter("personal"):
    print(item.attrib)
```

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Update attribute, add new attribute:

```
# print("6) Updating elements")
for child in root:
    for subchild in child:
        print(subchild.tag)
        if subchild.tag == "personal":
            print("Attributes:",subchild.attrib)
            subchild.attrib["address"]=subchild.attrib["address"]
+
                                                    "- 560046"
            subchild.set('Added pin','yes')
        else:
            print("Text:",subchild.text)
```

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Update attribute/text, add new attribute:

```
# print("6) Updating elements")
for child in root:
    for subchild in child:
        print(subchild.tag)
        if subchild.tag == "personal":
            print("Attributes:",subchild.attrib)
            subchild.set("address","Bangalore: ")
            subchild.set('Added pin','yes')
        else:
            print("Text:",subchild.text)
            subchild.text = "Degree: Need to update"

tree.write(file)
```

- Database communication
- XML Parser
- **Web Scrapping**
- PyUnit
- PySpark

Remove elements:

```
# print("7) Removing elements")
for child in root:
    for subchild in child.findall("personal"):
        child.remove(subchild)
tree.write(file)
```

Add new elements:

```
#Add new element with details
david = et.Element("Ponting")
david.text="Details about ponting"
david.attrib={}
david.attrib["Name"] = "Ricky ponting"
contractor = root.find("Contractors")
contractor.insert(1,david)
et.dump(contractor)
```

PyUnit

https://docs.python.org/2/library/unittest.html

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Unit testing:

1) Introduction
2) Modules (Setup)
3) Using ‘unittest’ module

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Practices:

1) A simple test
2) Multiple test cases using “asserts”
3) Test case to test “Exceptions”
4) Setup and TearDown

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

A simple test file:

```
#1) Importing SUT : calc
import calc
#2) Importing unittest (in-build)
import unittest

class TestCases(unittest.TestCase):
    def test_add_01(self):
        calc.add(2,3)
```

How to run test cases??

1) At command line: *python3 -m unittest test.py*

2) At the end of the test case: Add the below line

"unittest.main()"

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Rules to write test cases:

1) Test case function should start with “test”

2) Test case functions should be there in a class with “unittest.TestCase” as base class.

3) Line “unittest.main()” should be there at the end of test cases.

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Using “assert” to verify the test result:

```
#1) Importing SUT : calc
import calc
#2) Importing unittest (in-build)
import unittest

class TestCases(unittest.TestCase):
    def test_add_01(self):
        self.assertEqual(calc.add(2,3) , 5)
```

assertFalse()
assertTrue()
assertEqual()
assertNotEqual()

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Using "setUp" and "tearDown":

```
#3) Start writing test cases
class TestCases2(unittest.TestCase):
    def setUp(self):
        print("Going to setup for the test case")
    def tearDown(self):
        print("Teardown the setup after test case")
    def tetadd_01(self):
        self.assertEqual(calc.add(2,3) , 5)
```

- setUp and tearDown are member functions in each testcase class.
- Case sensitive
- Runs for each test case
- Can have different definitions for each testcase class.

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Testing Exceptions raised by actual code:

```
def test_dev_01(self):
    with self.assertRaises(ValueError):
        result = calc.dev(3,0)
```

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Important command line arguments:

-m To run module

-v Verbose mode (Gives more prints about each test case pass or fail

-b Buffering STDOUT and STDERROR (Prints only in case of FAILURE)

-c Ctrl+C can stop the tests but continue the current one

-f Stop the test on first error or Failure

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Using PyTest:

PySpark

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Topics

1) Introduction on Hadoop, Spark
2) Introduction on PySpark?
3) Installation procedure?
4) How to start working with PySpark?

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Big data news:

-> 2.7 Zetabytes of data “breath” in the digital universe

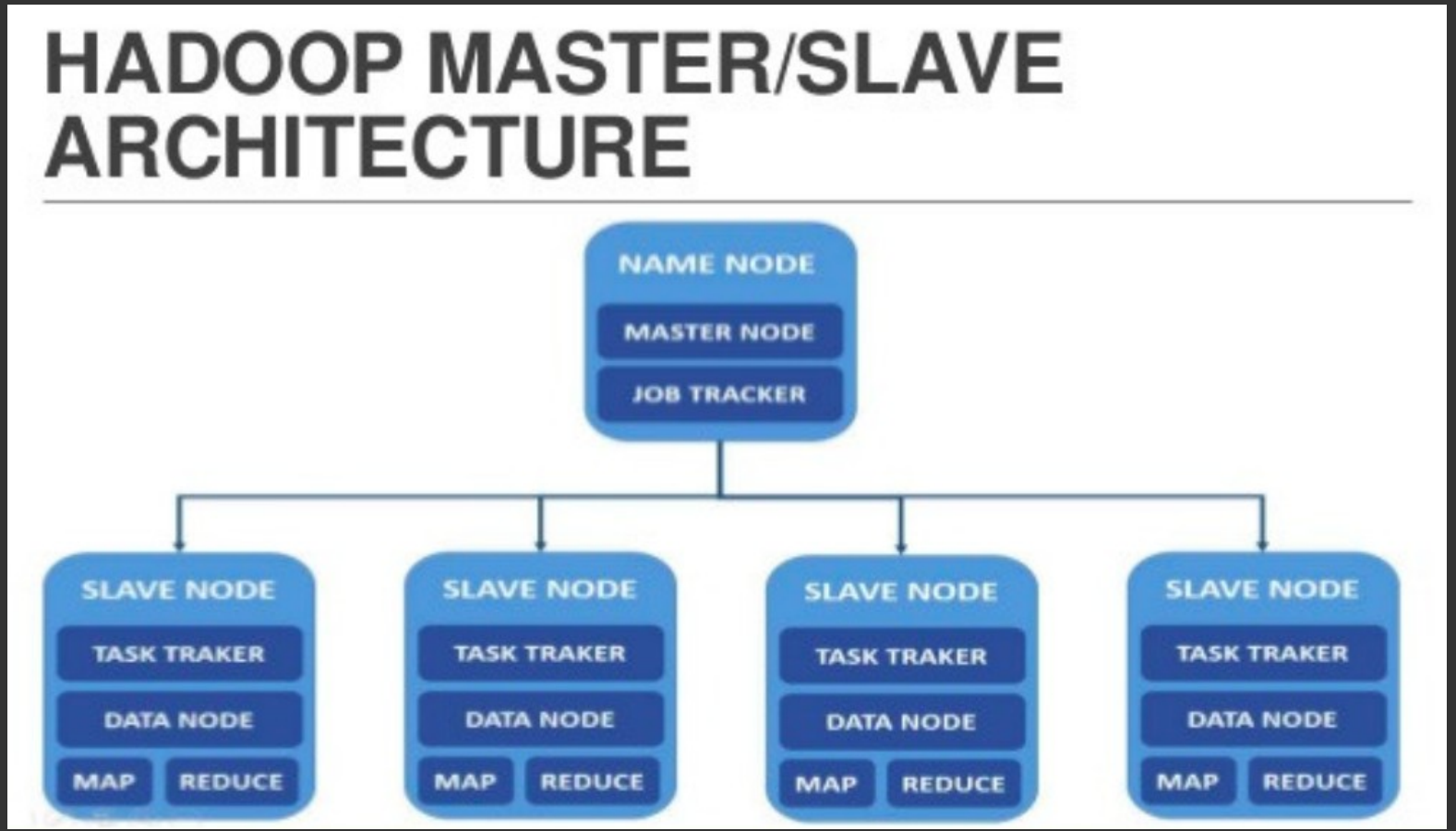
-> Facebook, alone, stores, accesses and analyzes 30+ Petabytes of user generated data

->Walmart’s customer database contains more than 2.5 Petabytes of data

-> One of AT&T’s database shelters 312 terabytes of data

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

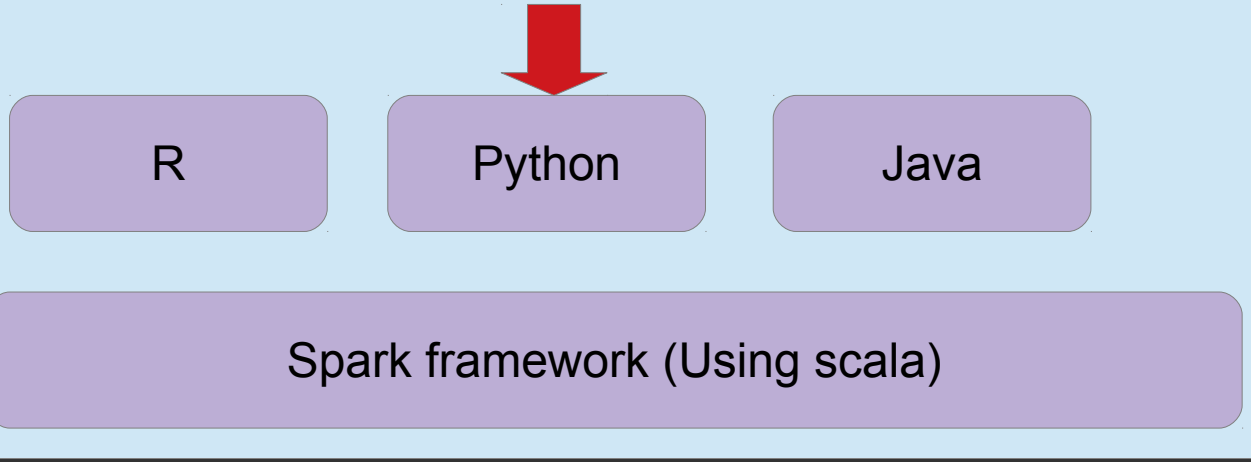
BigData and Hadoop a basic infrastructure:



- Database communicati on
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Spark:

Spark is built on to of **Scala** programming language which runs on **JAVA** virtual machine.



- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Brief history of Spark?

-> Open source processing engine originally developed by "Matei Zaharia" as his PhD thesis.

-> First version of Spark was release in 2012

-> Donated to Apache Software Foundation

Advantages:

-> An open-source

-> Better performance (as in-memory data processing engine)

-> Supports by default: GraphX, Streaming and sql work loads efficiently

-> Built in Machine learning library (in Hadoop, need third party lib)

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

What is PySpark?

PySpark helps data scientists interface with **Resilient Distributed Datasets** (RDDs) in apache spark and **python**.

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Installing Spark?

-> Install java

-> Setup “Apache Spark”:

-> Download from “http://spark.apache.org”

-> Install and add in PATH variable

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

RDD (Resilient Distributed Dataset):

- Entire Spark depends on RDD
- Can be list of strings/integers/rows in relational database

Operations on RDD:

1) Transformation
2) Actions

- Database communication
- XML Parser
- Web Scrapping
- PyUnit
- PySpark

Writing Program:

```
from pyspark import SparkContext, SparkConf
file = "../data/airport.csv"
output_file = "../out/sample10"

def fun1(str1):
    print(str1)
    return str1.split(",")

if "__main__"==__name__:
    conf = SparkConf()
    conf.setAppName("ParsingAirports")
    conf.setMaster("local[2]") #4 threads on 4 cores

    context = SparkContext(conf = conf)

    RDD1 = context.textFile(file)

    RDD2 = RDD1.map(fun1)

    RDD3 = RDD2.filter(lambda x: x[2]=="\"heliport\"")

    RDD4 = RDD3.repartition(1)

    RDD4.saveAsTextFile(output_file)
```

References:

Maximising Python speed
(http://docs.micropython.org/en/v1.8.6/pyboard/reference/speed_python.html)

Multiprocessing module:
https://docs.python.org/2/library/multiprocessing.html

Write own C code and invoke in Python using Cython:
https://medium.com/@shamir.stav_83310/making-your-c-library-callable-from-python-by-wrapping-it-with-cython-b09db35012a3