**I. Python introduction.**

Python is an open-source scripting language, cross-platform (Win, Mac, Linux). Latest versions are 2.7.x and 3.7.x which are developed in parallel, 2.7 is a part of Linux. Can be used in ObjectOriented or Procedural way. Has large collection of modules, frameworks for all tasks starting from simple scripts to FullStack Web Development, DataSince, MachineLearning, Audio processing, Astronomical calculations, plot graphics etc. MSSQL 2017 use Python Anaconda framework for introduced MachineLearning Service.

**II. Installation.**

Installation packages available on

<https://www.python.org/>

After installation path to executable binaries (like python.exe, pip.exe on Win) should be added to PATH variables.

After installation and set PATH please check via command:

python --version

pip --version

**III. IDE.**

Python is scripting language which doesn’t require compilation so it’s possible to use just notepad, but in complex projects impossible to work without debugger and IDE, There are many different IDE, for example MS Visual Studio 2017+(in earlier python was an extension) or PyCharm.

**1. Base language constructions and types.**

For block of code definition Python doesn’t have brackets like C# or Java and doesn’t use BEGIN END like T-SQL, instead of them idents are using, for example

if a > b:

c = a - b

print(c)

else:

c = b - a

print(c)

In Python exists a few immutable data types like bool,int, string, all others are mutable and python use for them reference counter to freed memory like in java or .net.

Q: what is difference between list and tuple?

*A: List (список)*

*Базовая структура данных в python. Элементы в списке хранятся последовательно, каждому из них присвоены индексы, начиная с нуля. В отличие от массива, список может хранить объекты любого типа.*

***В каких случаях использовать?***

*Для хранения элементов, объединенных каким либо признаком. В случае, если изменение элементов и/или расширение списка не предполагается, следует использовать неизменяемый аналог - кортеж.*

*Tuple (кортёж)*

*Кортеж - это* ***неизменяемый*** *и более быстрый аналог списка. Он защищает хранимые данные от непреднамеренных изменений и может использоваться в качестве ключа в словарях (словарь - ассоциативный массив в python).*

***В каких случаях использовать?***

*Для хранения данных вместо списка (если они не предполагают изменений).*

*Зачем нужны кортежи, если есть списки?*

*Защита от дурака. То есть кортеж защищен от изменений, как намеренных (что плохо), так и случайных (что хорошо).*

*Меньший размер.*

To run python code there are 2 option - by using interactive mode when you can start interpreter and type script in console or by running py script.

Task: write a script to calculate factorial of number. Input value can be hardcoded at the beginning of script.

Result of this and next tasks is .py file which can be executed from command line, for example “python.exe task1.py”

**2. Positional command line arguments, type cast**

To add reference to additional module Python use keyword import <module name>

Task: change script from previous task to use script positional parameter instead of hardcode. In this case command to run script will be like this “python.exe task2.py param1 param2”.

**3. Error handling part 1.**

Task: change script from previous task to handle type cast errors, for example when user passed string “123r” need to print message instead of error trace.

**4.Date functions, int limitation.**

Task: change script from previous task to calculate duration of calculation, add print to console message with datetime when execution started and completed. Test script on big values.

Q: what is result integer value limitation?

**5. Named arguments, argparse**

Task: modify previous script, replace positional argument by named, use **argparse** module for this task

**6. File reading**

Task: change script from previous task, now input parameter will be path to text file, file contains header and numbers like:

number\_to\_calc

4

5

1

7

54gt

4

Modified script should read this file, and calculate factorial of each number, print result and skip data conversion errors.

Note: don’t use csv module for this task.

**7. File writing**

Need to change script from previous task, now script should pass 2 parameters, first input file path, the second - output file path. Output file should contains header column and lines contained original value and result of function

Example input file:

number\_to\_calc

4

5

1

7

54gt

4

Example of output file:

number\_to\_calc,result

4,24

5,120

1,1

7,5040

54gt,

4,24

Task 7.1: output file encoding is UTF with BOM

Task 7.2: output file encoding is UTF without BOM

Task 7.3: output file new line char is \n

Task 7.4: output file new line char is \r\n

Note: don’t use csv module for this task.

**8. Error handling part 2**

Task: modify latest script to guarantee close files even in case of error. (try … finally, with)

**9. Csv module.**

Task: change script from previous task, use **csv** module to read and write csv files, use DictReader/DictWriter

**10. Threading**

Task: change latest script, import **threading** module, create queue(**queue** module) to store input numbers, then create 4 threads which will dequeue elements and calculate factorials.

**11. Configuration file**

Task: modify previous script, add yaml config file which will store input and output csv file path instead of named args. Use **pyyaml** module for this. This is not base module of python, so you need to use package manager to install it. Command looks like “pip install <module name>”

Instead of hardcoded 4 threads use number of cores from system information.

[optional] To store config, create separate class and implement singleton pattern.

()

**12. Monkey patching**

When we have big amount of data it’s difficult to test application, because it will require much time. To solve this issue we can create new function and at the beginning replace function reference, for example we have function “def ReadInput()” which reads billion of rows, we can create function “def ReadInputTest()” which will get small test dataset and at the beginning change referenc - “ReadInput = ReadInputTest”, so when the application call function ReadInput() in real will be called ReadInputTest. This technique known as “monkey patching”

Task: in this task when script call factorial function need to call cpu\_usage\_test function instead and replace data in your input file by numbers from script below. Calculate execution time for all input data and check CPU utilization during script execution (just in Windows task manager, not in python).

import random

import math

import datetime

def normal\_cdf(x, mu=0,sigma=1):

return (1 + math.erf((x - mu) / math.sqrt(2) / sigma)) / 2

def bernoulli\_trial(p):

return 1 if random.random() < p else 0

def binomial(n, p):

return sum(bernoulli\_trial(p) for \_ in range(n))

def make\_hist(p, n, num\_points):

data = [binomial(n, p) for \_ in range(num\_points)]

return data

# range 1000 - 10000

def cpu\_usage\_test(number):

return sum(make\_hist(0.75, 100, number))

print(str(datetime.datetime.utcnow()))

print(cpu\_usage\_test(67587))

print(cpu\_usage\_test(100000))

print(cpu\_usage\_test(10000))

print(cpu\_usage\_test(50000))

print(cpu\_usage\_test(68935))

print(cpu\_usage\_test(34577))

print(cpu\_usage\_test(98769))

print(cpu\_usage\_test(55555))

print(cpu\_usage\_test(98644))

print(cpu\_usage\_test(12349))

print(cpu\_usage\_test(92349))

print(cpu\_usage\_test(76349))

print(cpu\_usage\_test(86449))

print(cpu\_usage\_test(43549))

print(str(datetime.datetime.utcnow()))

**13. Multiprocessing.**

Task: change latest script to use **multiprocessing** module instead of threading. Calculate execution time for all input data and check CPU utilization during script execution. Compare performance with previous version.

Q: what is the difference between these 2 modules?

**\* 14. Excel files**

Task: convert csv to xlsx file, install **openpyxls** (openpyxl?) module and modify script to use excel as input and output files

**15. Databases**

Task: remove multiprocessing and factorial replacement from previous task (only one thread and simple factorial function), now we will use database as source and destination. you can use any DB which has connector, for example mssql (**pymssql** module), PostgreSQL (**psycopg2** module) or SQLite (**sqlite3**). Create in config files parameters for in and out tables, modify script to read input and save result to DB.

**16. Generators**

Task: if data source is too big from one hand we can’t read all rows at a time (fetchall), because we don’t have enough memory, from the other hand read by one row resultset isn’t efficient too. The best way is to read by batches. Need to change script, add function which reads records batch by batch while we have data in a source dataset.

What is expected:

def somereadfunction():

#here is generator

# and here is reading

For row in somereadfunction():

# processing

**17. Logging**

Task: add error logging, use **logging** module for this task

**\* 18. petl**

Task: modify script from previous task, install **petl** module, create simple etl which will use db as source, then add calculated column and save result to db and csv file.

PDF documentation: <https://media.readthedocs.org/pdf/petl/latest/petl.pdf>

**[advanced]19. Freezing**

Task: install **cx\_Freeze** module and make freezed build

**[advanced]20. Virtual environment**

Task: now we need some linux machine, you can use image from osboxes.org if you don’t have. As Python 2.7 is a part of OS we can have some issues with running scripts, installing modules, or when we have to run different project on the same server would be better to separate projects. For this purposes exists virtual environments. The task is install Python 3.x, create virtual environment, activate, install dependencies, and test the script. To install all dependencies within one command create file requirements.,txt, specify all modules and use it as parameter for pip.

**[advanced]21. Distributing modules.**

setuptools is module to make user’s python module install-able with "pip install ...", so common code across multiple project can be separated, created distributed module and all project which require this functionality can install this lib using pip install command.

Task: create distributed module with factorial function, install to your project with pip command

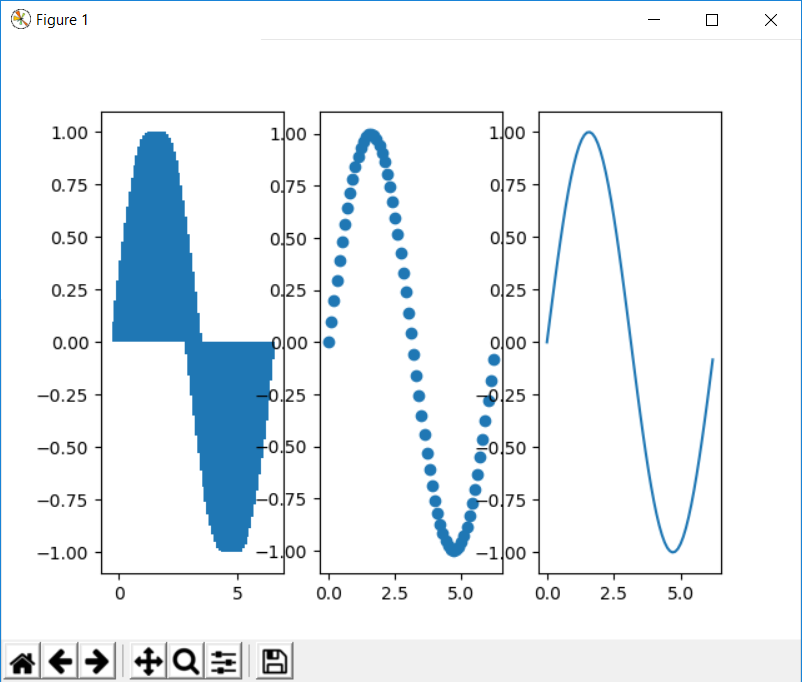
**\*[advanced] 22. Apache Airflow**

Task: install **airflow** module, create DAG to run script from previous task by schedule.

**[advanced] 23. matplotlib, numpy**

Description. matplot module is using to draw different diagrams etc, can save result to pdf or png file, exists one more module to plot in browser web page. matplot require tkinter module which can be installed with Python only, on Windows checkbox “tcl/tpl” should be selected.

Task 1: create script to write graphics below (sin function), use numpy arrays to store axis data



Task 2: modify script to save the result to pdf file

\*Task 3: install MSSQL 2017 with Python ML Servicie, enable external script, and execute plotting to file from TSQL, instead of generating data in Python, generate in T-SQL and pass as to external script.

**24. urllib, zip**

Task: create script to download and extract files from <http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz> .

**\*[advanced] 25. Anaconda, MSSQL Machine learning**

TBD

**\*\*[advanced] 26. Flask**

Task 1: create RESTFull service for TODO list - GET, PUT, UPDATE, DELETE

Task 2: use flask SocketIO to implement socket which will send changed todo items.

Task 3: deploy flask service to Apache, use mod\_WSGI,

**\*\*[advanced] 27. sqlalchemy**

Task: modify flask restfull to store TODOs in database, use **sqlalchemy** module as ORM.

**\*\*[advanced] 28. Django**

TBD

\* - for DB Developers

\*\* - for FrontEnd Developers