|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

**ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИКИ НИРС**

по теме: «Парсиг. Парсинг на языке программирования Python»

Группа РК6-46Б

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Бобров Д. Д.**

*подпись, дата                   фио.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .**

*подпись, дата                   фио.*

*Москва, 2023 г.*

**Техническое задание**

Разработать программу парсер на Python, которая отправляет запрос, получает данные из внешнего источника по заданному запросу, обрабатывает эти данные и выводит их в любом отличном от изначального формате. При написании программы использовать инструменты(библиотеки) парсинга Python.

**Введение**

Целью программы является сбор данных из внешнего ресурса. Программа предполагает использование библиотек запросов и перевода формата данных.

**Описание алгоритма**

Первой частью кода является сбор информации и первичная обработка. В качестве источника данных выступает API сайта котировок. При создании собственного URL-адреса воспользуемся методами конвертаций времени библиотек time и datetime. C помощью библиотеки requests происходит обращение к серверу, который возвращает данные в формате json. Используя встроенный модуль Python json, происходит декодирование полученных данных, после чего их значения присваиваются переменным по соответствующим по ключам.

Второй частью кода является вторичная обработка данных и их вывод в формате графика. Пользуясь ранее определёнными переменными, производим обработку для дальнейшего построения графиков. Для создания графического окна применяется библиотеки pyqtgraph, sys и виджеты PyQt5.

**Описание реализации**

Программа состоит из двух файлов: GetData и MakeGraph.

Первый файл после запуска требует ввести 5 значений: код акции, режим торгов, разрешение для графика, дату начала отсчёта и дату конца отсчёта.

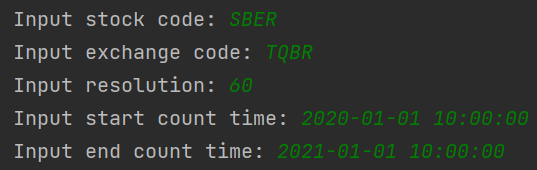
При помощи функции to\_unix() текстовый формат дат переводится в миллисекунды. Программой НЕ предусмотрен ошибочный ввод.

После ввода всех требуемых значений составляется URL-адрес, по которому скрипт обращается к API сайта. Если запрос был выполнен, то данные API заносятся в переменную. Далее эти данные конвертируются в обрабатываемый формат, и их записывают в списки по соответствующим ключам. Затем составляется список средней цены.

Второй файл отображает интерактивное окно графика. Дочерний класс окна MainWindow от QMainWindow определяет атрибуты и методы построения графика используя виджеты библиотеки PyQt5 и pyqtgraph.

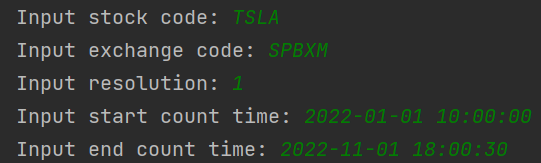
Метод plot() создаёт линию графика по заданным атрибутам.

**Тесты**

Ввод:

Вывод:



Ввод:

Вывод:

**Приложение 1**

# **Листинг программы**

**GetData.py**

import requests  
import time  
import json  
from datetime import datetime

def to\_unix(dttime):  
 year = int(dttime[0:4])  
 month = int(dttime[5:7])  
 day = int(dttime[8:10])  
 hour = int(dttime[11:13])  
 minutes = int(dttime[14:16])  
 seconds = int(dttime[17:19])  
 try:  
 dttime = datetime(year, month, day, hour, minutes, seconds) # 1970-01-01 05:30:00 2020-01-01 10:00:00  
 except ValueError:  
 exit(-1)  
 unix = (time.mktime(dttime.timetuple()))  
 #print(unix)  
 return(int(unix))  
  
  
  
stock\_code = input('Input stock code: ') #TSLA SBER GAZP  
exchange\_code = input('Input exchange code: ') #SPBXM TQBR TQBR  
resolution = input('Input resolution: ') #60 supported\_resolutions: "1", "5", "15", "30", "60", "1D"= D, "1W"= D, "1M"= D  
start\_count\_time = str(to\_unix(input('Input start count time: '))) # input('Input start count time: ')  
end\_count\_time = str(to\_unix(input('Input end count time: ')))   
custom\_url = 'https://api.bcs.ru/udfdatafeed/v1/history?symbol=' + stock\_code + '&classcode=' + exchange\_code + '&resolution=' + resolution + '&from=' + start\_count\_time + '&to=' + end\_count\_time  
  
if (requests.get(custom\_url)):  
 response = requests.get(custom\_url)  
else:  
 exit(-2)  
print(response)  
  
json\_text = json.loads(response.text)  
  
scale = json\_text['scale']  
unix\_time = json\_text['t']  
open\_price = json\_text['o']  
close\_price = json\_text['c']  
highest\_price = json\_text['h']  
lowest\_price = json\_text['l']  
  
mid\_price = []  
for i in range(len(open\_price)):  
 mid\_price.append((open\_price[i]+close\_price[i])/2)

**MakeGraph.py**

import GetData  
from PyQt5.QtWidgets import QApplication, QMainWindow  
import pyqtgraph as pg  
import sys  
  
  
class MainWindow(QMainWindow):  
  
 def \_\_init\_\_(self):  
 super(MainWindow, self).\_\_init\_\_()  
  
 self.graphWidget = pg.PlotWidget()  
 self.setCentralWidget(self.graphWidget)  
  
 time = GetData.unix\_time  
 #data\_1 = GetData.close\_price  
 data\_2 = GetData.mid\_price  
  
 #Add Title  
 self.graphWidget.setTitle(GetData.stock\_code+', '+GetData.exchange\_code, color="w", size="20pt")  
 #Add Axis Labels  
 styles = {"color": "#f00", "font-size": "20px"}  
 self.graphWidget.setLabel("left", "Price", \*\*styles)  
 self.graphWidget.setLabel("bottom", "Time (msec)", \*\*styles)  
 #Add grid  
 self.graphWidget.showGrid(x=True, y=True)  
 #Set Range  
 self.graphWidget.setXRange(min(GetData.unix\_time), max(GetData.unix\_time), padding=0)  
 self.graphWidget.setYRange(min(GetData.open\_price), max(GetData.close\_price), padding=0)  
 #Add plot  
 #self.plot(time, data\_1, "Close", 'r')  
 self.plot(time, data\_2, "Mid\_price", 'b')  
  
 def plot(self, x, y, plotname, color):  
 pen = pg.mkPen(color=color)  
 self.graphWidget.plot(x, y, name=plotname, pen=pen, symbol='o', symbolSize=5, symbolBrush=(color))  
  
def main():  
 app = QApplication(sys.argv)  
 main = MainWindow()  
 main.show()  
 app.exec()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()