Лабораторная работа №1

Тема: Основы языка Python.
Цель: Научиться создавать скрипты на языке Python в процедурном и функциональном стилях.

Темы для предварительной проработки [УСТНО]:
1. Синтаксис языка Python.
2. Типизация и управляющие конструкции Python.
3. Работа со списками, строками, кортежами, словарями.
4. Генераторы и итераторы.
5. Декораторы.
6. Анонимные функции. Функции map, filter, reduce, zip.
7. Модули functools и itertools.

Индивидуальные задания [КОД]:
1. Напишите скрипт, который преобразует введенное с клавиатуры вещественное число в денежный формат. Например, число 12,5 должно быть преобразовано к виду «12 руб. 50 коп.». В случае ввода отрицательного числа выдайте сообщение «Некорректный формат!» путем обработки исключения в коде.
2. Написать скрипт, который выводит на экран «True», если элементы программно задаваемого списка представляют собой возрастающую последовательность, иначе – «False».
3. Напишите скрипт, который позволяет ввести с клавиатуры номер дебетовой карты (16 цифр) и выводит номер в скрытом виде: первые и последние 4 цифры отображены нормально, а между ними – символы «*» (например, 5123 **** **** 1212).
4. Напишите скрипт, который разделяет введенный с клавиатуры текст на слова и выводит сначала те слова, длина которых превосходит 7 символов, затем слова размером от 4 до 7 символов, затем – все остальные.
5. Напишите скрипт, который позволяет ввести с клавиатуры текст предложения и сформировать новую строку на основе исходной, в которой все слова, начинающиеся с большой буквы, приведены к верхнему регистру. Слова могут разделяться запятыми или пробелами. Например, если пользователь введет строку «*город Донецк, река Кальмиус*», результирующая строка должна выглядеть так: «*город ДОНЕЦК, река КАЛЬМИУС*».
6. Напишите программу, позволяющую ввести с клавиатуры текст предложения и вывести на консоль все символы, которые входят в этот текст ровно по одному разу.

7. Напишите скрипт, который обрабатывает список строк-адресов следующим образом: сначала определяет, начинается ли каждая строка в списке с префикса «www». Если условие выполняется, то скрипт должен вставить в начало этой строки префикс «http://», а затем проверить, что строка заканчивается на «.com». Если у строки другое окончание, то скрипт должен вставить в конец подстроку «.com». В итоге скрипт должен вывести на консоль новый список с измененными адресами. Используйте генераторы списков.
8. Напишите скрипт, генерирующий случайным образом число n в диапазоне от 1 до 10000. Скрипт должен создать массив из n целых чисел, также сгенерированных случайным образом, и дополнить массив нулями до размера, равного ближайшей сверху степени двойки. Например, если в массиве было n=100 элементов, то массив нужно дополнить 28 нулями, чтобы в итоге был массив из 2^8=128 элементов (ближайшая степень двойки к 100 – это число 128, к 35 – это 64 и т.д.).
9. Напишите программу, имитирующую работу банкомата. Выберите структуру данных для хранения купюр разного достоинства в заданном количестве. При вводе пользователем запрашиваемой суммы денег, скрипт должен вывести на консоль количество купюр подходящего достоинства. Если имеющихся денег не хватает, то необходимо напечатать сообщение «Операция не может быть выполнена!». Например, при сумме 5370 рублей на консоль должно быть выведено «`5*1000 + 3*100 + 1*50 + 2*10`».
10. Напишите скрипт, позволяющий определить надежность вводимого пользователем пароля. Это задание является творческим: алгоритм определения надежности разработайте самостоятельно.
11. Напишите генератор frange как аналог range() с дробным шагом. Пример вызова:

```
for x in frange(1, 5, 0.1):
    print(x)
    # выводит     1    1.1    1.2    1.3    1.4   …   4.9
```

12. Напишите генератор get_frames(), который производит «оконную декомпозицию» сигнала: на основе входного списка генерирует набор списков – перекрывающихся отдельных фрагментов сигнала размера size со степенью перекрытия overlap. Пример вызова:

```
for frame in get_frames(signal, size=1024, overlap=0.5):
    print(frame)
```

13. Напишите собственную версию генератора enumerate под названием extra_enumerate. Пример вызова:

```
for i, elem, cum, frac in extra_enumerate(x):
    print(elem, cum, frac)
```

В переменной cum хранится накопленная сумма на момент текущей итерации, в переменной frac – доля накопленной суммы от общей суммы на момент текущей итерации. Например, для списка x=[1,3,4,2] вывод будет таким:

```
(1, 1, 0.1)    (3, 4, 0.4)    (4, 8, 0.8)    (2, 10, 1)
```

14. Напишите декоратор non_empty, который дополнительно проверяет списковый результат любой функции: если в нем содержатся пустые строки или значение None, то они удаляются. Пример кода:

```
@non_empty
def get_pages():
    return ['chapter1', '', 'contents', '', 'line1']
```

15. Напишите параметризированный декоратор pre_process, который осуществляет предварительную обработку (цифровую фильтрацию) списка по алгоритму: s[i] = s[i]–a·s[i–1]. Параметр a можно задать в коде (по умолчанию равен 0.97). Пример кода:

```
@pre_process(a=0.93)
def plot_signal(s):
    for sample in s:
        print(sample)
```

16. Напишите скрипт, который на основе списка из 16 названий футбольных команд случайным образом формирует 4 группы по 4 команды, а также выводит на консоль календарь всех игр (игры должны проходить по средам, раз в 2 недели, начиная с 14 сентября текущего года). Даты игр необходимо выводить в формате «14/09/2016, 22:45». Используйте модули random и itertools.

Контрольные вопросы [ОТЧЕТ]:

1. Чем отличаются компилируемые и интерпретируемые языки программирования?
2. Каковы особенности типизации в языке Python? Как интерпретатор Python работает с памятью?
3. Каковы особенности преобразования типов в языке Python?
4. Что общего и отличного в языке Python имеют строки, списки, словари, кортежи, множества?
5. Как в Python обявляются и вызываются пользовательские функции?
6. Что такое область видимости функции и правило LEGB?
7. Что такое анонимные функции? Когда их удобно использовать?
8. Что делают функции map, filter, reduce, zip?
9. Каковы особенности обработки исключений в языке Python?

Краткая теоретическая справка.

Python – кросс-платформенный интерпретируемый язык. Установку Python на Windows/Linux/MacOS можно выполнить несколькими способами. Оптимальным вариантом является установка бесплатного дистрибутива Anaconda, компонентами которого являются: интерпретатор Python версии 3.5 (доступна также версия 2.7), интерактивная среда IPython и тетради jupyter, а также инструментальная среда разработчика Spyder. На рис. 1 приведен фрагмент тетради jupyter.

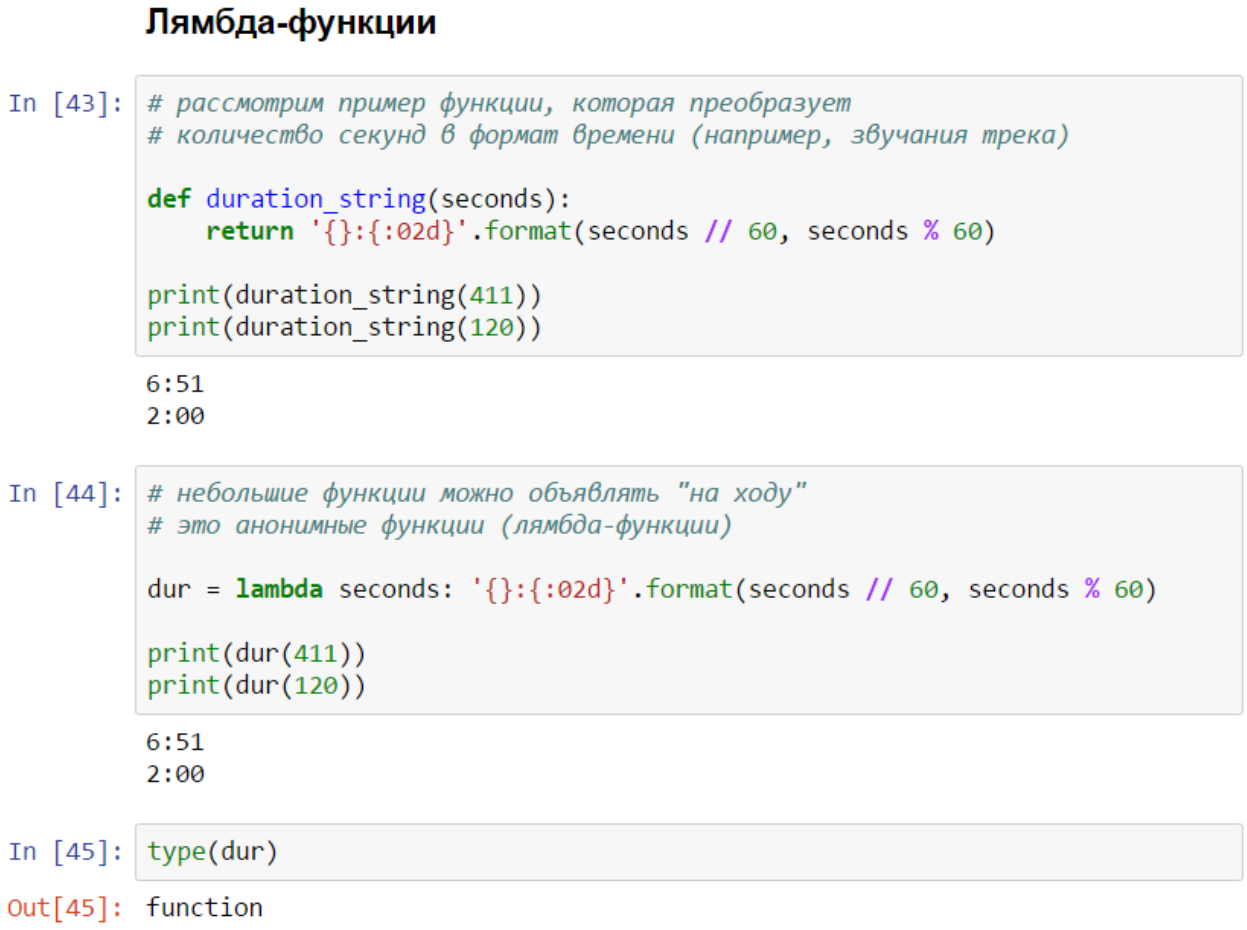

Рисунок 1 – Фрагмент тетради jupyter notebook

Важной особенностью синтаксиса Python является кодирование блоков с помощью отступов (по соглашению – 2 или 4 пробела). Ниже приведен пример кода с ветвлением:

```
if age < 25:
    print("You're young!")
else:
    print("You're grown up!")
```

Для организации циклического процесса используются операторы while и for. Оператор while стандартен; оператор for является переборным (аналог foreach в языке C#). Пример кода, перебирающего числа от 0 до 9:

```
for i in range(10):
    print(i)
```

В Python есть специальный вариант цикла for..else. Например:

```
extensions = ['h', 'cpp', 'hpp', 'c']

# цикл for-else:
for ext in extensions:
    if ext == 'py':
        break
else:
    print('Среди расширений нет питоновского')
```

Python – идиоматичный язык. Это означает, что во многих случаях нужно использовать характерные только для него синтаксические и методологические приемы. Подробнее об этом можно прочитать в стандарте оформления кода PEP8 и в сообщении модуля this (import this).

Пользовательские функции в Python объявляются с помощью ключевого слова def:

```
def hello(name):
    return 'Hello, {}!'.format(name)
```

В языке Python функции являются объектами. Таким образом, функции можно присваивать переменным:

```
def foo(x):
    return x * 10

func = foo
func(73)
```

Также функции можно передавать как объект в функцию:

```
def process_elementwise(seq, func):
    return [func(s) for s in seq]

def first_letter(s):
    return s[0]

words = ['Fundamentals', 'of', 'brainwashing...']
process_elementwise(words, first_letter)
```

В языке Python одну из центральных ролей играют генераторы. Понятия «итератор» и «генератор» часто смешивают. Итератор – это концепция (любой объект, имеющий методы next() и __iter__()). Генератор – это языковое средство (объект вокруг функции с инструкцией yield). **Любой генератор является итератором, но не наоборот.**

Пример генератора нечетных чисел:

```
def odd_iterator(x):
    for i in range(1, x, 2):
        yield i

for x in odd_iterator(10):
    print(x)
```

Небольшие функции можно объявлять "на ходу". Это т.н. анонимные функции (лямбда-функции). Например:

```
dur = lambda seconds:
        '{}:{:02d}'.format(seconds // 60, seconds % 60)

print(dur(411))
# выведет '6:51'
```

Часто анонимные функции применяются в качестве блоков в функциях map, filter, reduce. Функция filter() позволяет из исходного списка получить новый (а точнее, итерируемый объект) на основе некоторой *фильтрующей* функции, которая передается в качестве параметра. Например:

```
files = ['1.wav', '2.mp3', '3.jpg', '4.png', '5.wav']

sound_files = filter(
    lambda f: f.endswith('wav') or f.endswith('mp3'), files)

print(list(sound_files))
```

Функция map() позволяет из исходного списка получить новый (а точнее, итерируемый объект) на основе некоторой *отображающей* функции, которая передается в качестве параметра. Например:

```
extensions = map(lambda f: f[f.index('.')+1:], files)
print(list(extensions))
```

Ниже приведен эквивалент map() в виде т.н. генератора списка:

```
extensions = [f[f.index('.')+1:] for f in files]
print(extensions)
```

Декораторы – это языковое средство Python, позволяющее динамически модифицировать поведение уже существующей функции. Например:

```
def copyright(func):
    def _wrapper():
        func()
        print('(c) Wise man')

    # декоратор возвращает функцию (callable)
    return _wrapper

# декорируем функцию
@copyright
def print_slogan():
    print('The truth will set you free')

print_slogan()
# здесь происходит такой вызов:
# copyright(print_slogan())()
```

В приведенном коде создается декоратор copyright, который после вызова любой функции выводит на консоль строку с копирайтом «(c) Wise man». Декоратор применяется к функции print_slogan(), выводящей, в свою очередь строку «The thuth will set you free».

Правило **LEGB** (правило перебора областей видимости):

1) область «L, Local» – все локальные имена: имена, любым способом присвоенные внутри функции (объявленных как def или lambda), и не помеченные ключевым словом global в этой функции;

2) область «E, Enclosing function locals» – все имена в локальной области видимости всех замыкающих функций (объявленных как def или lambda), от самой вложенной до внешней;

3) область «G, Global» (на уровне модуля) – имена, глобальные для модуля или помеченные ключевым словом global внутри конструкции def где-либо в файле;

4) область «B, Built-in» (Python) – предопределенные глобальные имена Python, которые списком содержатся в переменной __builtins__ (которые, в свою очередь, берутся из модуля builtins): print, open, reversed, ValueError и т.д.

На собеседованиях в IT-компаниях часто предлагается следующее задание: «Найдите количество нулей, которыми заканчивается факториал 100! (1 · 2 · 3 · 4 · 5 · ... · 100)». На языке Python эта задача решается просто: 1) в цикле вычисляем произведение чисел от 1 до 100 (это возможно, т.к. Python изначально поддерживает длинную арифметику и работу с произвольно большими числами); 2) преобразуем число в строку; 3) считаем количество символов ‘0’ в конце этой строки. Код решения выглядит так:

```
# вычисление факториала 100! в простейшем цикле:

factorial = 1
for i in range(1, 101):
    factorial *= i

print('100! = ')
print(factorial)

# преобразование числа 100! в строку:

s = str(factorial)

# нахождение количества нулей в конце числа 100!:

zerocount = 0

# перебор элементов коллекции в обратном направлении
# (идиоматичный вариант)
for i in reversed(s):
    if i != '0':
        break
    zerocount += 1

# можно реверсировать строку с помощью среза с шагом -1:
# for i in s[::-1]:
#    if i != '0':
#        break
#    zerocount += 1

print('Количество нулей в конце 100!: {}'.format(zerocount))
```

При запуске скрипта вывод на консоль будет следующим:

```
100! =
933262154439441526816992388562667004907159682643816214685929
638952175999932299156089414639761565182862536979208272237582
511852109168640000000000000000000000
```

```
Количество нулей в конце 100! : 24
```

Лабораторная работа №2

Тема: Работа с файловой системой и процессами в Python.

Цель: Научиться программировать на языке Python скрипты, взаимодействующие с файлами и процессами операционной системы.

Темы для предварительной проработки [УСТНО]:

1. Чтение/запись текстовых файлов.
2. Модули os, sys.
3. Модули shutil, glob, pathlib.
4. Модуль subprocess.
5. Регулярные выражения.

Индивидуальные задания [КОД]:

1. Напишите скрипт, который читает текстовый файл и выводит символы в порядке убывания частоты встречаемости в тексте. Регистр символа не имеет значения. Программа должна учитывать только буквенные символы (символы пунктуации, цифры и служебные символы слудет игнорировать). Проверьте работу скрипта на нескольких файлах с текстом на английском и русском языках, сравните результаты с таблицами, приведенными в wikipedia.org/wiki/Letter_frequencies.
2. Напишите скрипт, позволяющий искать в заданной директории и в ее подпапках файлы-дубликаты на основе сравнения контрольных сумм (MD5). Файлы могут иметь одинаковое содержимое, но отличаться именами. Скрипт должен вывести группы имен обнаруженных файлов-дубликатов.
3. Задан путь к директории с музыкальными файлами (в названии которых нет номеров, а только названия песен) и текстовый файл, хранящий полный список песен с номерами и названиями в виде строк формата «01. Freefall [6:12]». Напишите скрипт, который корректирует имена файлов в директории на основе текста списка песен.
4. Напишите скрипт, который позволяет ввести с клавиатуры имя текстового файла, найти в нем с помощью регулярных выражений все подстроки определенного вида, в соответствии с вариантом. Например, для варианта № 1 скрипт должен вывести на экран следующее:

```
Строка 3, позиция 10 : найдено '11-05-2014'
Строка 12, позиция 2 : найдено '23-11-2014'
Строка 12, позиция 17 : найдено '23-11-2014'
```

   Вариант 1: найдите все даты – подстроки вида «11-05-2014».
   Вариант 2: найдите все значения времени – подстроки вида «23:15:59».

Вариант 3: найдите все IPv4-адреса – подстроки вида «192.168.5.48».
Вариант 4: найдите все строки вида «*type x = value*», где *type* – это тип (может принимать значение int, short или byte), *x* – любое слово, *value* – любое число.
Вариант 5: найдите все номера телефонов – подстроки вида «(000)1112233» или «(000)111-22-33».
Вариант 6: найдите все строки вида «*x: type [N]*», где *type* – это тип (может принимать значение int, short или byte), *x* – любое слово, *N* – любое положительное целое число.
Вариант 7: найдите все «смайлы» – подстроки вида «:)», «:-)», «)))» (количество скобок может быть любым, начиная с 1).
Вариант 8: найдите все логические выражения – подстроки вида «x&&y», «x&y», где *x* и *y* – любые слова. Количество пробелов может быть также любым.
Вариант 9: найдите все донецкие почтовые индексы – подстроки вида «83000, Донецк» (первые 2 символа строго закреплены: «83»).
Вариант 10: Найдите все полные имена директорий Windows – подстроки вида «C:\Dir\SubDir3».

5. Введите с клавиатуры текст. Программно найдите в нем и выведите отдельно все слова, которые начинаются с большого латинского символа (от A до Z) и заканчиваются 2 или 4 цифрами, например *«Petr93», «Johnny70», «Service2002»*. Используйте регулярные выражения.
6. Напишите скрипт reorganize.py, который в директории --source создает две директории: Archive и Small. В первую директорию помещаются файлы с датой изменения, отличающейся от текущей даты на количество дней более параметра --days (т.е. относительно старые файлы). Во вторую – все файлы размером меньше параметра --size байт. Каждая директория должна создаваться только в случае, если найден хотя бы один файл, который должен быть в нее помещен. Пример вызова:

```
reorganize --source "C:\TestDir" --days 2 --size 4096
```

7. Написать скрипт trackmix.py, который формирует обзорный трек-микс альбома (попурри из коротких фрагментов mp3-файлов в пользовательской директории). Для манипуляций со звуковыми файлами можно использовать сторонние утилиты, например, FFmpeg.
Пример вызова и работы скрипта:

```
trackmix --source "C:\Muz\Album" --count 5 --frame 15 -l -e

  --- processing file 1: 01 - Intro.mp3
  --- processing file 2: 02 - Outro.mp3
  --- done!
```

Параметры скрипта:
--source (-s) – имя рабочей директории с треками, обязателен;
--destination (-d) – имя выходного файла, необязателен (если не указан, то имя выходного файла – mix.mp3 в директории source);
--count (-c) – количество файлов в "нарезке", необязателен (если он не указан, то перебираются все mp3-файлы в директории source);
--frame (-f) – количество секунд на каждый файл, необязателен (если не указан, скрипт вырезает по 10 секунд из произвольного участка каждого файла);
--log (-l) – необязательный ключ (если этот ключ указывается, то скрипт должен выводить на консоль лог процесса обработки файлов, как в примере);
--extended (-e) – необязательный ключ (если этот ключ указывается, то каждый фрагмент попурри начинается и завершается небольшим fade in/fade out).

Контрольные вопросы [ОТЧЕТ]:
1. Как в Python осуществляется чтение/запись текстовых файлов?
2. Как в Python можно получить информацию о содержимом директории?
3. Как в Python можно скопировать, удалить и переименовать файл?
4. Каковы базовые синтаксические элементы регулярных выражений?
5. Какие возможности предоставляет модуль subprocess?

Краткая теоретическая справка.
Чтение/запись файлов в Python осуществляется с помощью функции открытия файла open() и функций непосредственно чтения и записи данных. Пример кода:

```
s = ['Hello ', 'World\n', '(c) Admin\n']

# запись списка строк в текстовый файл demo.txt:
try:
    f = open('demo.txt', 'wt')
    f.writelines(s)
except IOError as err:
    print(err)
except:
    print('Something\'s gone wrong...')
else:
    print('File\'s been updated succesfully')
finally:
    if f:
        f.close()
```

Приведенный код можно сократить с помощью конструкции with, цель которой – предоставить более короткую запись кода вида:

```
# -- setup (начало работы с ресурсом) --
# try:
#     -- do_smth (действия с ресурсом) --
# finally:
#     -- tear down (корректное закрытие ресурса) --
```

Например:

```
# чтение строк из файла (с конструкцией with)
try:
    with open(r'data\demo.txt', 'rt') as f:
        lines = f.readlines()
except IOError as err:
    print(err)
    lines = []

# вывести строки из файла в "сыром" виде
print('Read from file:', lines)

# убрать \n на конце каждой строки с помощью rstrip и map
newlines = list(map(lambda x: x.rstrip(), lines))

# убрать \n на конце каждой строки
# с помощью rstrip и list comprehension
newlines = [x.rstrip() for x in lines]

print('After post-processing:', newlines)
```

Все необходимые функции для работы с файловой системой в Python содержатся в модулях os, shutil, pathlib, glob.
Пример кода для получения списка только папок в текущей директории:

```
folders = [entry for entry in os.listdir(os.getcwd())
                 if os.path.isdir(entry)]
print(folders)
```

Для вычисления контрольной суммы файла существует модуль hashlib. Пример кода с использованием этого модуля приведен ниже:

```
import hashlib

with open(r'data\somefile.txt', 'r') as f:
      content = f.read()

checksum = hashlib.md5(content).hexdigest()
```

Одним из распространенных способов сериализации/десериализации объектов в Python является использование функций модуля pickle. Например:

```
import pickle

data = (1, 3.15, 'Hello', [1, 4, 5])

# сериализация объекта в pkl-файл
pickle.dump(data, open(r'data\data.pkl', 'wb'))

# десериализация (загрузка объекта из pkl-файла)
obj = pickle.load(open(r'data\data.pkl', 'rb'))
print(obj)
```

Для работы с процессами операционной системы в настоящее время предпочтительнее использовать модуль subprocess. Ниже приведен код вызова внешнего процесса (ipconfig.exe) с синхронным получением выходного результата:

```
output = subprocess.check_output(['ipconfig', '/all'])
print(output)
```

Функция check_output() будет ожидать завершения вызванного процесса. Если необходимо отслеживать поток вывода порожденного процесса, можно в простейшем случае организовать цикл опроса потока. Например, утилита ping.exe выводит данные на консоль в определенные промежутки времени. Код скрипта, работающего с данной утилитой, выглядит так:

```
process = subprocess.Popen(['ping', '127.0.0.1'],
stdout=subprocess.PIPE)

# синхронный busy spin, работа с потоком вывода,
# пока процесс не отработает
while True:
    output = process.stdout.readline()

    # когда процесс завершится,
    # poll() вернет код завершения, а не None
    rc = process.poll()

    if rc is not None:
        break
    if output:
        print(output.strip())

print('The process has finished with return code {}\n'.
   format(rc))
```

Лабораторная работа №3

Тема: Объектно-ориентированное программирование на языке Python.
Цель: Научиться писать скрипты в объектно-ориентированном стиле с графическим интерфейсом пользователя на языке Python.

Темы для предварительной проработки [УСТНО]:
1. Классы и объекты в Python.
2. «Магические» методы классов.
3. Динамическое создание классов. Метаклассы.
4. Менеджеры контекста.
5. Библиотеки tkinter, wxPython, pyQt для программирования скриптов с графическим интерфейсом пользователя в Python.

Индивидуальные задания [КОД]:
1. Задан простой класс Fraction для представления дробей:

```
class Fraction(object):

     def __init__(self, num, den):
          self.__num = num
          self.__den = den
          self.reduce()

     def __str__(self):
          return "%d/%d" % (self.__num, self.__den)

     def reduce(self):
          g = Fraction.gcd(self.__num, self.__den)
          self.__num /= g
          self.__den /= g

     @staticmethod
     def gcd(n, m):
          if m == 0:
               return n
          else:
               return Fraction.gcd(m, n % m)
```

Дополнить класс таким образом, чтобы выполнялся следующий код:

```
frac = Fraction(7, 2)
print(-frac)                    # выводит -7/2
print(~frac)                    # выводит 2/7
print(frac**2)                  # выводит 49/4
print(float(frac))              # выводит 3.5
print(int(frac))                # выводит 3
```

2. Напишите классы «Книга» (с обязательными полями: название, автор, код), «Библиотека» (с обязательными полями: адрес, номер) и корректно свяжите их. Код книги должен назначаться автоматически при добавлении книги в библиотеку (используйте для этого статический член класса). Если в конструкторе книги указывается в параметре пустое название, необходимо сгенерировать исключение (например, ValueError). Книга должна реализовывать интерфейс Taggable с методом tag(), который создает на основе строки набор тегов (разбивает строку на слова и возвращает только те, которые начинаются с большой буквы). Например, tag() для книги с названием ‘War and Peace’ вернет список тегов [‘War’, ‘Peace’]. Реализуйте классы таким образом, чтобы корректно выполнялся следующий код:

```
lib = Library(1, ’51 Some str., NY’)
lib += Book(‘Leo Tolstoi’, ‘War and Peace’)
lib += Book(‘Charles Dickens’, ‘David Copperfield’)

for book in lib:

  # вывод в виде: [1] L.Tolstoi ‘War and Peace’
  print(book)

  # вывод в виде: [‘War’, ‘Peace’]
  print(book.tag())
```

3. Создайте графическую оболочку для скрипта, написанного в ходе выполнения задания № 4 лабораторной работы № 2, в виде диалогового окна (рис. 2). Рекомендуется использовать wxPython или PyQt.

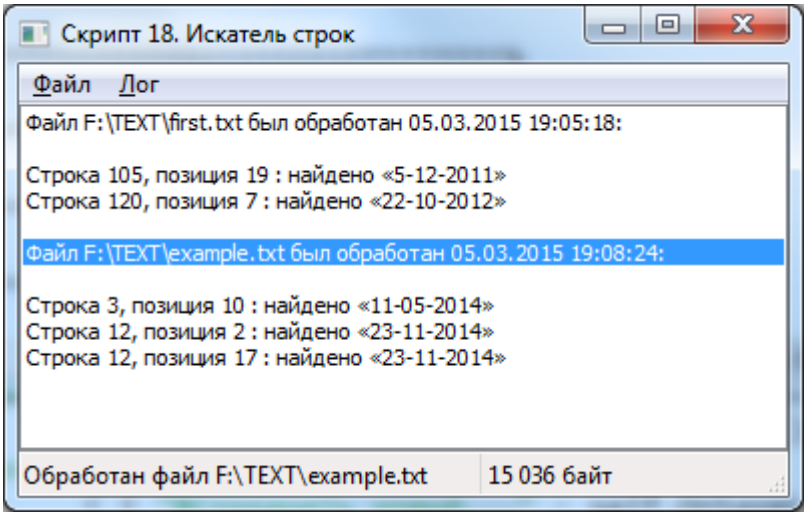

Рисунок 2 – Внешний вид окна результатов поиска строк

Требования к окну и скрипту:

- всю область окна должен занимать список с результатами поиска строк по шаблону в файле и указанием даты и времени поиска. Поиск производится автоматически при каждом открытии какого-либо файла, при этом список не очищается, а пополняется новыми результатами. При запуске скрипта список изначально должен быть пустым (из файла лога данные подгружать не нужно);
- строка меню содержит пункты «*Файл*» (с подпунктом «*Открыть...*» для открытия файла, в котором необходимо искать строки) и «*Лог*» (с подпунктами «*Экспорт...*», «*Добавить в лог*», «*Просмотр*»). Файл лога находится в рабочей папке скрипта и называется *script18.log*. Если файл отсутствует, скрипт при запуске должен выдать диалоговое окно с информацией «*Файл лога не найден. Файл будет создан автоматически*» и кнопкой «*ОК*». При выборе пункта меню «*Экспорт...*» содержимое списка должно сохраниться в файле, который укажет пользователь. При выборе пункта «*Добавить в лог*» содержимое списка приписывается в конец файла *script18.log*. При выборе пункта «*Просмотр*» текущее содержимое списка удаляется, и список заполняется данными из лога. Перед этим действием скрипт должен выдать диалоговое окно с вопросом «*Вы действительно хотите открыть лог? Данные последних поисков будут потеряны!*» и кнопками «*Да*» и «*Нет*»;
- статусная строка должна состоять из двух полей: в первом поле (60% ширины окна), в зависимости от последнего произведенного действия, выводится либо текст «*Открыт лог*», либо текст «*Обработан файл <полное_имя_файла>*»; второе поле (40% ширины окна) служит для отображения размера последнего обработанного файла в байтах. Эта строка форматируется: выводятся пробелы между степенями тысячи (например, «*2 036 231 байт*»);
- файлы нужно открывать и сохранять с помощью стандартного диалогового окна (рис. 3).

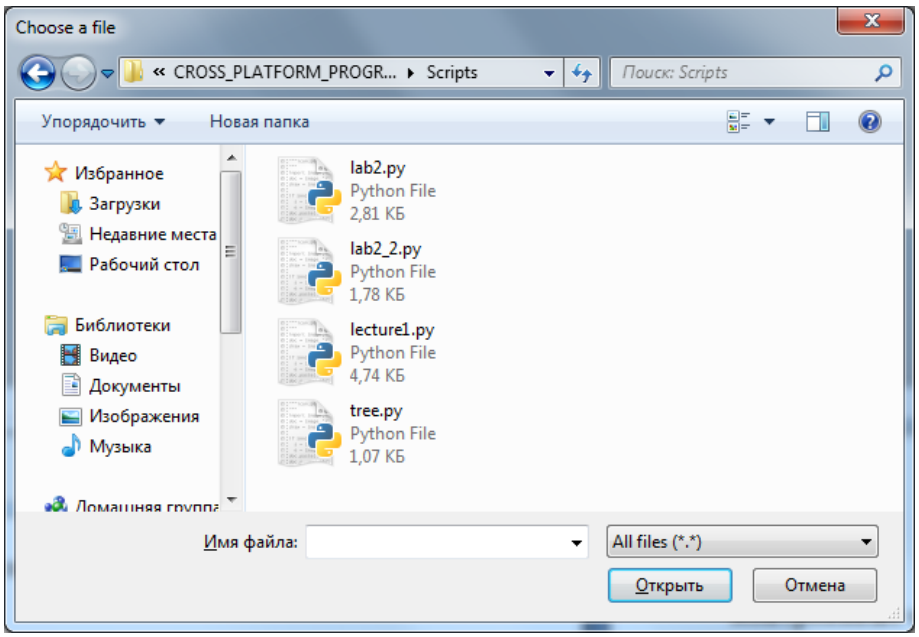

Рисунок 3 – Вид окна открытия файла

4. Напишите простой класс StringFormatter для форматирования строк со следующим функционалом:
   – удаление всех слов из строки, длина которых меньше n букв;
   – замена всех цифр в строке на знак «*»;
   – вставка по одному пробелу между всеми символами в строке;
   – сортировка слов по размеру;
   – сортировка слов в лексикографическом порядке.

   Примечание. *Разделители слов можно задавать отдельно. По умолчанию в качестве разделителя принимается только символ пробела.*

5. Напишите скрипт с графическим интерфейсом пользователя для демонстрации работы класса StringFormatter. Примеры окон приведены на рис. 4 (все элементы управления необходимо обязательно реализовать те же, что присутствуют на рисунке). Разные комбинации отмеченных чекбоксов приводят к разным цепочкам операций форматирования задаваемой в верхнем поле строки с разными результатами:

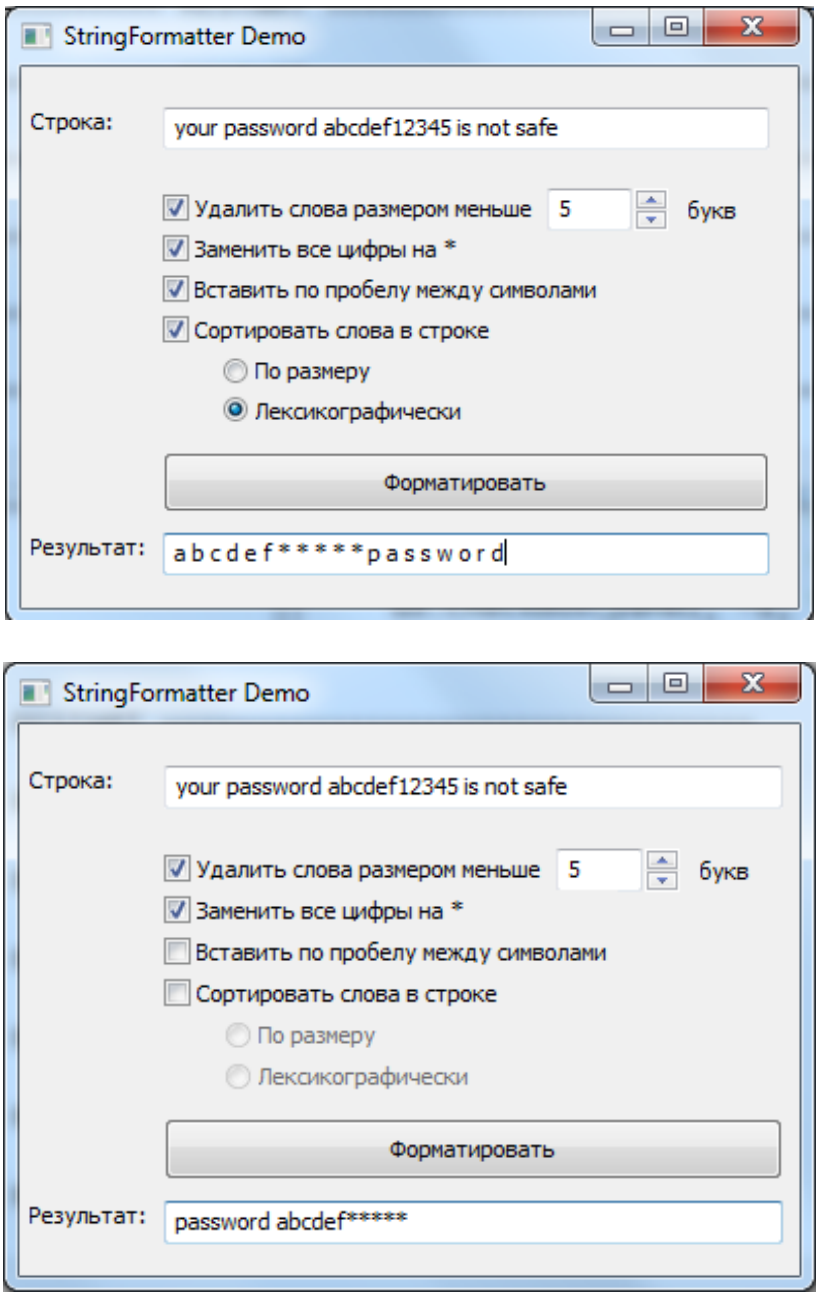

Рисунок 4 – Вид окна для демонстрации работы класса StringFormatter

Контрольные вопросы [ОТЧЕТ]:

1. Какими способами можно создать класс в Python?
2. Каковы особенности инкапсуляции в Python?
3. Чем отличаются статические методы, методы класса и методы-члены?
4. Каковы особенности перегрузки операторов в Python?
5. Что такое менеджер констекста и как его реализовать в Python?
6. Что такое метакласс?
7. Для чего используется атрибут __slots__?
8. Что означает глубокое копирование и как оно производится в Python?
9. Для чего предназначен модуль enum?
10. Для чего предназначен модуль attrs?

Краткая теоретическая справка.

В языке Python в достаточной степени реализована поддержка объектно-ориентированного программирования. Можно выделить такие особенности классов и объектов в Python:

- инкапсуляции как таковой нет, однако есть соглашение об именовании членов класса, подлежащих сокрытию. Имена таких членов должны начинаться с двух символов подчеркивания. Защищенные члены должны иметь имена, начинающиеся с одного символа подчеркивания. Также язык предоставляет возможность оформления геттеров и сеттеров в виде свойств (properties):

```
@property
def num(self):
    return self.__num

@num.setter
def num(self, n):
    self.__num = n
```

- поддерживается традиционное наследование классов. Родительский класс указывается в скобках при объявлении подкласса; вызов методов родителя производится с помощью функции super(). Например:

```
class Barcode(Label):
    ''' Класс для наклеек со штрих-кодом
          (наследуется от класса наклеек Label)
    '''
    def __init__(self):
        super(Barcode, self).__init__(100, 25)
```

- полиморфизм естественным образом присущ классам в Python, т.к. это язык с динамической типизацией. Ниже демонстрируется полиморфный метод parse():

```
from classes.parsers import *

parser1 = JSONParser()
parser1.load('a.json')

parser2 = XMLParser()
parser2.load('a.xml')

parsers = [parser1, parser2, StringParser('hahaha')]

for parser in parsers:
    parser.parse()      # полиморфизм («утиная» типизация)
```

- перегрузка операторов реализована в виде т.н. «магических» методов, полный список которых приведен в табл. 1.

Таблица 1 – «Магические» методы классов в Python

«Магический» метод	**Пример вызова**	**Пояснение**
__new__(cls [,...])	instance = MyClass(arg1, arg2)	__new__ вызывается при создании объекта
__init__(self [,...])	instance = MyClass(arg1, arg2)	__init__ вызывается при инициализации объекта
__cmp__(self, other)	self == other, self > other и т.д.	Вызывается при любых сравнениях
__pos__(self)	+self	Унарный плюс
__neg__(self)	-self	Унарный минус
__invert__(self)	~self	Побитовая инверсия
__index__(self)	x[self]	Вызывается при использовании объекта в качестве индекса
__nonzero__(self)	bool(self)	Булевское значение объекта
__getattr__(self, name)	self.name # имени не существует	Доступ к несуществующему атрибуту
__setattr__(self, name, val)	self.name = val	Присваивание атрибуту
__delattr__(self, name)	del self.name	Удаление атрибута
__getattribute__(self, name)	self.name	Доступ к любому атрибуту
__getitem__(self, key)	self[key]	Доступ к элементу по индексу
__setitem__(self, key, val)	self[key] = val	Присвоение элементу значения по индексу
__delitem__(self, key)	del self[key]	Удаление элемента по индексу
__iter__(self)	for x in self	Итерация по объекту как по последовательности
__contains__(self, value)	value in self, value not in self	Проверка на вхождение в объект
__call__(self [,...])	self(args)	"Вызов" объекта
__enter__(self)	with self as x:	with-менеджеры контекста (вход)
__exit__(self, exc, val, trace)	with self as x:	with-менеджеры контекста (завершение)
__getstate__(self)	pickle.dump(pkl_file, self)	Pickle-сериализация
__setstate__(self)	data = pickle.load(pkl_file)	Pickle-десериализация

Классы можно создавать непосредственно во время выполнения скрипта. Например:

```
Person = type('Person', (object,),
              {'name': 'vasya',
               'age': 21,
               'calc': lambda self: self.age + 1})
p = Person()
print(p.name)
print(p.calc())

# это эквивалентно написанию такого класса:
#
# class Person(object):
#     def __init__(self):
#         self.name = 'vasya'
#         self.age = 21
#
#     def calc(self):
#         self.age += 1
#
```

Для поддержки типов-перечислений в Python разработан класс Enum в модуле enum. Пример использования этого класса:

```
from enum import Enum, unique

@unique
class OperatingSystem(Enum):
    WINDOWS = 0
    UNIX = 1
    MACOS = 2

  OperatingSystem.UNIX
```

Специальный атрибут __slots__ позволяет явно указать в коде класса, какие атрибуты ожидаются у объекта класса, что приводит к более экономному расходу памяти и более быстрому доступу к указанным атрибутам.

По умолчанию, объекты классов хранят атрибуты в словарях. Память расходуется неэффективно, когда нужно хранить всего несколько атрибутов. Особенно это становится критичным в случае хранения большого числа таких «скромных» объектов.

Объявление в классе __slots__ резервирует ровно столько памяти под атрибуты, сколько нужно для их хранения. К примеру, атрибуты классов библиотеки SQLAlchemy эффективно хранятся в памяти благодаря тому, что реализованы через __slots__.

Пример класса, в котором указан атрибут __slots__:

```
class SmallObject(object):

    __slots__ = ['width', 'height', 'path']

    def __init__(self):
        self.width = 10
        self.height = 10
        self.path = 'default'

    @property
    def size(self):
        return self.width * self.height


obj = SmallObject()
print(obj.path)
print(obj.size)
```

Лабораторная работа №4

Тема: Прикладные пакеты Python.

Цель: Научиться проектировать и создавать одно- и многопоточные скрипты с графическим интерфейсом, позволяющие работать с базами данных, сетью и слабоструктурированной информацией.

Темы для предварительной проработки [УСТНО]:

- Работа с бинарными данными в Python.
- Пакеты Python для работы с файлами форматов json и xml, базами данных и объектно-реляционными отображениями.
- Пакеты requests, BeautifulSoup.
- Пакеты для научно-исследовательской деятельности: sciPy, numPy, symPy, pandas.

Индивидуальное задание [КОД]:

1. Напишите скрипт, читающий во всех mp3-файлах указанной директории ID3v1-теги и выводящий информацию о каждом файле в виде: [имя исполнителя] - [название трека] - [название альбома]. Если пользователь при вызове скрипта задает ключ -d, то выведите для каждого файла также 16-ричный дамп тега. Скрипт должен также автоматически проставить номера треков и жанр (номер жанра задается в параметре командной строки), если они не проставлены. Используйте модуль struct.
   ID3v1-заголовки располагаются в последних 128 байтах mp3-файла. Структура заголовка отражена в табл. 2.

Таблица 2 – Структура ID3v1-заголовка mp3-файла

Поле	Длина (байт)	Описание
header	3	3 символа: "TAG"
title	30	30 символов названия трека
artist	30	30 символов имени исполнителя
album	30	30 символов названия альбома
year	4	4 символа года издания
comment	28 или 30	Комментарий
zero-byte	1	Если в теге хранится номер трека, то этот байт зарезервирован под 0.
track	1	Номер трека альбома или 0. Имеет смысл, если предыдущий байт равен 0
genre	1	Индекс в списке жанров или 255.

2. Напишите скрипт для информационной системы библиотеки. База данных библиотеки включает таблицы «Авторы» с полями «id», «имя», «страна», «годы жизни», и «Книги» с полями «id автора», «название», «количество страниц», «издательство», «год издания»). Необходимо производить авторизацию пользователей, логины и пароли которых хранятся в отдельной таблице. Пароли должны храниться в зашифрованном виде (например, хэш SHA-1 или MD5). В программе должны быть окна для отображения информации о всех книгах и авторах, окно добавления книги/автора. Реализуйте также возможность сохранения информации о выделенном авторе в файле в формате json или XML (по выбору пользователя). При добавлении нового автора в базу допускается не заполнять поля в соответствующем окне, а распарсить файл, указанный пользователем (файл необходимо заранее создать и заполнить информацией вручную, в текстовом редакторе). Для *преобразования в формат XML и json* напишите собственный код; *парсинг* можно делать с помощью сторонних библиотек. Форматы файлов:

```
JSON:                          XML:
{
  "name": "L.N.Tolstoi",       <author>
  "country": "Russia",           <name>L.N.Tolstoi</Name>
  "years": [1828, 1910]          <country>Russia</Country>
}                                <years born="1828" died="1910"/>
                               </author>
```

3. Выполните задание № 2 средствами SQLAlchemy, включая создание и редактирование таблиц, а также выполнение таких запросов, как:
   - вывод фамилий всех авторов, родившихся в диапазоне между X и Y годами (задайте программно числа X и Y);
   - вывод всех книг, написанных авторами из России;
   - вывод всех книг с количеством страниц более N;
   - вывод всех авторов с числом книг более N.
4. Выполните задание № 3, используя в качестве базы данных NoSql-технологию MongoDB.
5. Напишите приложение для загрузки файлов из интернета. В главном окне должно быть три текстовых поля, в которые можно вводить URL файла на закачку; под каждым из текстовых полей должны быть индикаторы загрузки и рядом поля с процентом загрузки каждого файла. Необходимо организовать возможность качать от одного до трех файлов параллельно (использовать потоки обязательно, файлы загружать фрагментами по 4 Кб). Загрузка должна инициироваться нажатием кнопки «Start downloading!». По окончанию загрузки последнего файла должно появиться окно со столбчатой диаграммой со значениями времени загрузки каждого

файла в формате «2s 322ms» и размерами файлов (используйте библиотеку matplotlib).

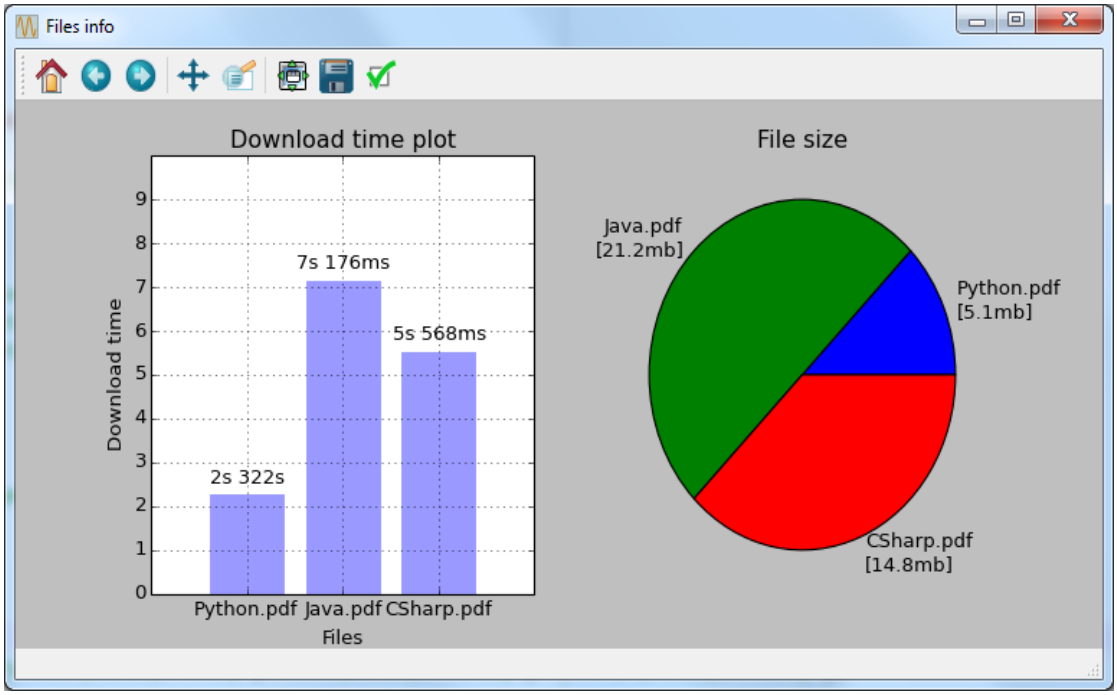

Рисунок 5 – Вид окна загрузки файлов

6. С помощью модуля numPy реализуйте следующие операции: 1) умножение произвольных матриц A (размерности 3x5) и B (5x2); 2) умножение матрицы (5x3) на трехмерный вектор; 3) решение произвольной системы линейных уравнений; 4) расчет определителя матрицы; 5) получение обратной и транспонированной матриц. Также продемонстрируйте на примере матрицы 5x5 тот факт, что определитель равен произведению собственных значений матрицы.
7. Выберите произвольную дифференцируемую и интегрируемую функцию одной переменной. С помощью модуля symPy найдите и отобразите ее производную и интеграл в аналитическом и графическом виде. Напишите код для решения произвольного нелинейного урванения и системы нелинейных уравнений.
8. Скачайте файл с информацией о всех государствах мира по адресу: https://github.com/mledoze/countries/blob/master/dist/countries.csv. С помощью модуля pandas отобразите: 1) 10 самых маленьких и самых больших стран мира по территории; 2) 10 самых маленьких и самых больших стран мира по населению; 3) все франкоязычные страны мира; 4) только островные государства; 5) все страны, находящиеся в южном полушарии. Сгруппируйте страны по первой букве; по населению; по территории. Программно сохраните в таблицу Excel все страны с выборочной информацией: название, столица, население, территория, валюта, широта, долгота.

Контрольные вопросы [ОТЧЕТ]:

1. Какие возможности предоставляет Python для работы с произвольными бинарными данными?
2. Какова типовая схема действий при работе с базами данных в Python?
3. Что такое объектно-реляционные отображения? Какие есть средства в Python по работе с ними?
4. Какие возможности предоставляет Python для работы с сетью?
5. Каковы особенности работы с потоками в Python?
6. Какой функционал предоставляют пакеты sciPy, numPy, symPy, pandas?

Краткая теоретическая справка.

Работа с базами данных в Python производится как через SQL-запросы, так и через объектно-реляционные отображения (Object-Relational Mapping, ORM) SQLAlchemy, Django ORM и другие. Типовая схема выполнения запросов не зависит от СУБД и включает такие блоки, как: 1) установка соединения с БД; 2) получение курсора; 3) функция execute(), непосредственно выполняющая SQL-запрос. Пример кода работы с БД, содержащей две таблицы «Альбом» и «Исполнитель»:

```
import sqlite3
# в этом примере используется SQLite;
# альтернативные коннекторы для других СУБД:
# import psycopg2 для PostgreSQL,
# import pymysql для MySql

DBname = r"db\sqlite3\catalog.db"

# 1) установка соединения
db = sqlite3.connect(DBname)

with db:
     # 2) получение курсора
    cursor = db.cursor()

     # 3) непосредственное выполнение SQL-запроса…
    cursor.execute("SELECT id, name from performers")
     # … и заполнение данных результатами выполнения запроса
    performers = cursor.fetchall()

    for performer in performers:
        print(performer)
         cursor.execute("""
                  SELECT performers.name, albums.name,
                  albums.release_year FROM albums JOIN
                  performers ON albums.perfID=performers.id
                         """)
        albums = cursor.fetchall()
        for album in albums:
            print(album)
```

С помощью модулей requests и BeautifulSoup можно осуществлять простой разбор HTML-страниц. Например, скрипт, находящий на странице все ссылки на mp3-файлы и загружающий их на клиентский компьютер, выглядит так:

```
import requests
from bs4 import BeautifulSoup

WEBSITE = 'http://www.somesite.foo/audio/'
content = requests.get(WEBSITE)

soup = BeautifulSoup(content.text, 'html.parser')

links = soup.find_all('a')
links = filter(lambda a: a.get('href'), links)
links = filter(lambda a: a.get('href').endswith('mp3')
                 and not a.get('href').startswith('?'), links)

for link in links:
    print('Downloading: {}'.format(link.get('href')))
    download_file(link.get('href'))
```

В функции загрузки файла download_file() также используется модуль requests и, в частности, функция iter_content():

```
import requests

DOWNLOAD_FOLDER = r'C:\User\Downloads'

def download_file(url):
    local_path = os.path.join(DOWNLOAD_FOLDER,
                              url.split('/')[-1])

    r = requests.get(url, stream=True)

    with open(local_path, 'wb') as f:
        for chunk in r.iter_content(chunk_size=4096):
            if chunk:
                f.write(chunk)

    return local_path
```

Язык Python очень популярен в научных кругах по всему миру, благодаря таким библиотекам, как numPy (линейная алгебра), sciPy (численные методы, обработка сигналов и др.), symPy (символьные вычисления), pandas (статистика и анализ данных) и matplotlib (визуализация данных). Ниже приведен список только некоторых возможностей, предоставляемых этими библиотеками.

```
np.array([2, 3, 4])              # инициализация вектора напрямую
np.empty(20, dtype=np.float32)   # вектор 20 вещественных чисел
np.zeros(200)                    # инициализация 200 нулями
np.ones((3,3), dtype=np.int32)   # создание матрицы 3x3 из единиц
np.eye(200)                      # создание единичной матрицы
np.zeros_like(a)                 # матрица нулей формы матрицы a
np.linspace(0., 10., 100)        # вектор 100 точек от 0 до 10
np.arange(0, 100, 2)             # точки от 0 до 99 с шагом 2
np.logspace(-5, 2, 100)          # 100 точек на логарифмической
                                 # шкале от 1e-5 до 1e2
np.copy(a)              # копирование матрицы
a.shape                 # кортеж с длинами всех измерений матрицы
a.ndim                  # количество измерений
a.sort(axis=1)          # сортировка вектора
a.flatten()             # формирование одномерного вектора
a.conj()                # комплексно-сопряженная матрица
a.astype(np.int16)      # преобразование к типу int
a.tolist()              # преобразование вектора в список
np.argmax(a, axis=1)    # индекс максимального элемента
np.cumsum(a)            # кумулятивная сумма
np.any(a)               # True, если хоть один элемент равен True
np.all(a)               # True, если все элементы равны True
np.where(cond)          # индексы элементов, для которых
                        # выполняется условие cond
np.where(cond, x, y)    # элементы от x до y, для которых
                        # выполняется условие cond
np.dot(a, b)                  # матричное произведение
np.sum(a, axis=1)             # сумма по измерению 1
a[None, :] * b[:, None]       # внешнее произведение матриц
np.outer(a, b)                # внешнее произведение матриц
np.sum(a * a.T)               # норма матрицы

plot(x,y, '-o', c='red', lw=2, label='bla') # вывод графика
scatter(x,y, s=20, c=color)                 # диаграмма разброса

# 3D-график на плоскости (быстрая версия)
pcolormesh(xx, yy, zz, shading='gouraud')

# 3D-график на плоскости (более медленная версия)
colormesh(xx, yy, zz, norm=norm)

contour(xx, yy, zz, cmap='jet')             # контурный график
n, bins, patch = hist(x, 50)                # гистограмма
imshow(matrix, origin='lower',              # вывод изображения
          extent=(x1, x2, y1, y2))
text(x, y, string, fontsize=12, color='m')  # текстовая надпись
```

ПЕРЕЧЕНЬ ЭКЗАМЕНАЦИОННЫХ ВОПРОСОВ

1. Особенности интерпретатора CPython. Память, переменные, объекты.
2. PEP8: стандарт оформления кода на языке Python. Идиомы Python.
3. Изменяемые типы данных Python.
4. Неизменяемые типы данных Python.
5. Генерация и перехват исключений на языке Python.
6. Структурирование кода на языке Python. Модули и пакеты.
7. Пользовательские функции. Области видимости и правило LEGB.
8. Итераторы и генераторы. Модуль itertools.
9. Декораторы. Модуль functools.
10. Анонимные функции. Функции map, filter, reduce.
11. Работа с файлами и файловой системой в Python.
12. Работа с процессами. Модуль subprocess.
13. Классы и объекты. Свойства, конструкторы, статические члены.
14. Перегрузка операторов в классах Python.
15. Менеджеры контекста.
16. Скрипты с графическим интерфейсом пользователя в Python.
17. Работа с базами данных в Python.
18. Работа с сетью.
19. Многопоточность в Python. Глобальная блокировка интерпретатора.
20. Пакеты Python для научно-исследовательской деятельности.

Пример экзаменационного билета

ГОУ ВПО «Донецкий национальный университет»

Образовательно-квалификационный уровень Академический магистр
Направление подготовки 10.04.01 Информационная безопасность
Учебная дисциплина Методы и алгоритмы программирования на Python Семестр 1

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 7

1. Анонимные функции. Функции map, filter, reduce.
2. Даны два списка по 12 элементов от 1 до 99. Напишите код вывода на консоль третьего списка, в котором сначала идут однозначные числа из первого списка, затем однозначные из второго, затем двузначные элементы из первого, затем двузначные из второго.
3. Напишите код классов «Треклист» и «Песня» (данные: название песни, время звучания); корректно свяжите их. В треклисте перегрузите оператор сложения (он должен выполнять слияние двух треклистов) и оператора in (вхождения песни с определенным названием в треклист). Если при создании песни указывается пустая строка, генерируйте исключение. Продемонстрируйте в коде работу классов и перехват исключения.

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Лутц М. Изучаем Python / М. Лутц. – М.: Символ-плюс, 2008. – 848 с.
2. Саммерфильд М. Программирование на Python 3. Подробное руководство / М. Саммерфильд. – М.: Символ-плюс, 2009. – 608 с.
3. Дронов В. Простой Python. Python 3 и PyQt 5. Разработка приложений / В. Дронов, Н. Прохоренок. – СПб.: БХВ-Петербург, 2016. – 832 с.
4. Рамальо Л. Python. К вершинам мастерства / Л. Рамальо. – М.: ДМК Пресс, 2016. – 768 с.
5. Любанович Б. Простой Python. Современный стиль программирования / Б. Любанович. – СПб.: Питер, 2016. – 480 с.
6. Чан У.Дж. Python. Создание приложений / У. Дж. Чан. – М.: Вильямс, 2016. – 816 с.
7. Слаткин Б. Секреты Python. 59 рекомендаций по написанию эффективного кода / Б. Слаткин. – М.: Вильямс, 2016. – 272 с.
8. Лутц М. Программирование на Python / М. Лутц. – М.: Символ-плюс, 2002. – 1136 с.
9. Официальный сайт Python. URL: http://www.python.org (дата обращения 03.12.2016).
10. Курс «Программирование на языке Python», автор – Шарий Т.В. URL: https://github.com/ar1st0crat/PythonCourse (дата обращения 03.12.2016).