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 находит элементы-дубликаты и заменяет их на случайные уникальные значения. Например, при входных данных 1 2 3 4 4, вывод 1 2 3 130 -10. Используйте модуль random.
4. 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).
5. 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.
6. 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*».

7. Напишите программу, позволяющую ввести с клавиатуры текст предложения и вывести на консоль все символы, которые входят в этот текст ровно по одному разу.
8. 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.
9. 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.).
10. Написать скрипт, который проверяет, можно ли из элементов списка составить ряд Фибоначчи. Если можно, то вывести элементы списка в отсортированном виде.
11. 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`».
12. Напишите скрипт, позволяющий определить надежность вводимого пользователем пароля. Это задание является творческим: алгоритм определения надежности разработайте самостоятельно.
13. 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
```

14. 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=4, overlap=0.5):
print(frame)
# выводит 1 2 3 4
          3 4 5 6
          5 6 7 8
          7 8 9 10
          9 10
```

15. 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 sum 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)
```

16. 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']
```

17. Напишите параметризированный декоратор pre_process, который осуществляет предварительную обработку (цифровую фильтрацию) списка по алгоритму: $s[i] = s[i] – a \cdot s[i–1]$. 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)
```

18. Напишите декоратор, который числовой результат функции преобразует в экспоненциальную записи.
19. 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.
20. Написать скрипт, который расшифровывает и зашифровывает

подаваемую строку. При расшифровке строки используется последовательность сигналов, а при шифровке - генерируется. Использовать модуль itertools.
Ввод: HWelJlopWoOlrld 1,0,1,1,0,1,1,0,1,1,0,0,1,1,1
Вывод: HelloWorld

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.

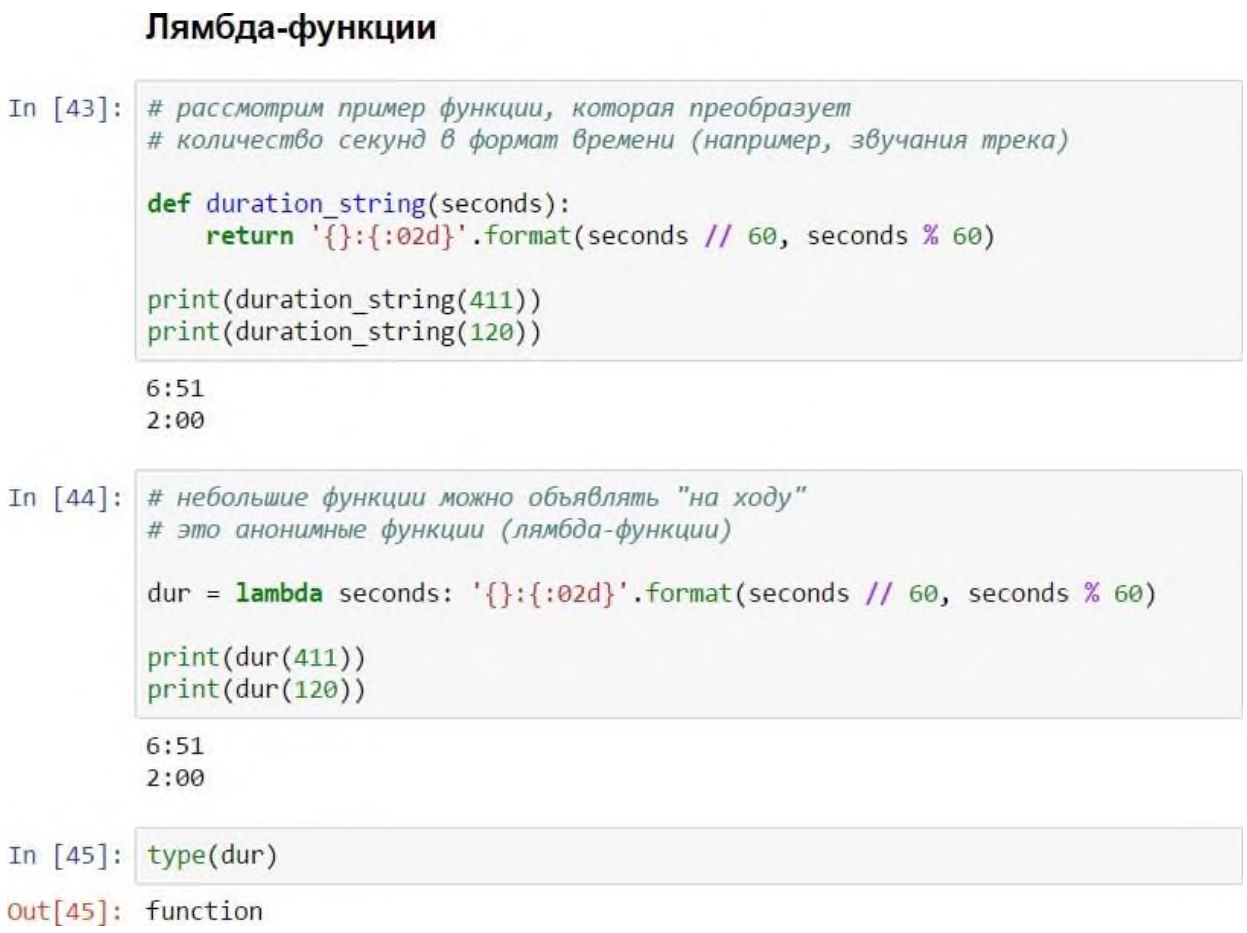

Рисунок 1 – Фрагмент тетради jupyter notebook

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
```