Московский государственный технический университет им. Н.Э.Баумана

Кафедра «Системы обработки информации и управления»

Утверждаю: _____ Согласовано: _____

"___"_____2016 г. "___"_____2016 г.

«Введение в Python»

Отчет по лабораторной работе №4

(вид документа)

писчая бумага формата А4

(вид носителя)

_____

(количество листов)

Исполнитель: студент группы РТ5-51

_____ Семенов А.А.

"_____"_____ 2016 г.

Москва – 2016

Задача 1 (ex_1.py)

Необходимо реализовать генераторы field и gen_random

Генератор field последовательно выдает значения ключей словарей массива

```
from librip.gen import field
goods = [
  {'title': 'Ковер', 'price': 2000, 'color': 'green'},
  {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
  {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
  {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]
print(list(field(goods,'title','price')))
```

```
import random
# Генератор вычленения полей из массива словарей
# Пример:
# goods = [
#    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
# ]
# field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'
# field(goods, 'title', 'price') должен выдавать {'title': 'Ковер',
'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
def field(items, *args):
  assert len(args) > 0
  if len(args) == 1:
    arg=args[0]
    for item in items:
      if arg is not None:
        if arg in item:
          yield item[arg]
  else:
    for item in items:
      r={}
      for arg in args:
```

```
            if arg is not None:
                if arg in item:
                    r[arg]=item[arg]
        yield r
```

Вывод:

[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}]

Задача 2 (ex_2.py)

Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной bool-параметр ignore_case , в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False

```
#!/usr/bin/env python3

from librip.gen import gen_random

from librip.iterators import Unique

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

data2 = gen_random(1, 3, 10)

un=Unique(data2)

print(list(un))

# Реализация задания 2
```

```
# Генератор списка случайных чисел

# Пример:

# gen_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1

# Hint: реализация занимает 2 строки

def gen_random(begin, end, num_count):

  i=0

  while i<num_count:

    yield random.randint(begin, end)

    i+=1

    # Необходимо реализовать генератор
```

```
class Unique(object):

  def __init__(self, items, **kwargs):

    # Нужно реализовать конструктор

    # В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore_case,

    # в зависимости от значения которого будут считаться одинаковые строки в разном регистре

    # Например: ignore_case = True, Абв и АБВ разные строки

    #           ignore_case = False, Абв и АБВ одинаковые строки, одна из них удалится

    # По-умолчанию ignore_case = False
```

```
    self.IGNORE_CASE = kwargs['ignore_case'] if 'ignore_case' in
kwargs.keys() else False
    self.ITEMS = list(items)
    self.PASSED = set()
  def __next__(self):
    # Нужно реализовать __next__
    while True:
      if self.INDEX == len(self.ITEMS) - 1:
        raise StopIteration
      self.INDEX += 1
      val = str(self.ITEMS[self.INDEX])
      val2 = val if self.IGNORE_CASE else val.lower()
      if val2 not in self.PASSED:
        self.PASSED.add(val2)
        return val
  def __iter__(self):
    self.INDEX = -1
    return self
```

Вывод:

['1', '2', '3']

[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

Задача 3 (ex_3.py)

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив, отсортированный по модулю. Сортировку осуществлять с помощью функции sorted

```
#!/usr/bin/env python3
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
print(sorted(data,key=lambda key:abs(key)))
# Реализация задания 3
```

Вывод:

[0, 1, -1, 4, -4, -30, 100, -100, 123]

Задача 4 (ex_4.py)

Необходимо реализовать декоратор print_result , который выводит на экран результат выполнения функции. Файл ex_4.py не нужно изменять. Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции, печатать результат и возвращать значение. Если функция вернула список (list), то значения должны выводиться в столбик. Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равно

```
from librip.decorators import print_result

# Необходимо верно реализовать print_result

# и задание будет выполнено

@print_result
def test_1():
  return 1
@print_result
def test_2():
  return 'iu'
@print_result
def test_3():
  return {'a': 1, 'b': 2}
@print_result
def test_4():
  return [1, 2]
test_1()
test_2()
test_3()
test_4()
```

```
def print_result(func):
  def my_decorate(*args):
    if len(args) > 0:
      res=func(args[0])
    else:
      res = func()
    if type(res) is list:
```

```
      st="\n".join([str(x) for x in res ])
    elif type(res) is dict:
      st="\n".join([str(x)+"="+str(res[x]) for x in res])
    else:
      st=res
    mas=(func.__name__,st)
    print ('\n'.join([str(x) for x in mas]))
    return res
  return my_decorate
```

Вывод:

test_1

1

test_2

iu

test_3

a=1

b=2

test_4

1

2

Задача 5 (ex_5.py)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран

```
from time import sleep

from librip.ctxmngrs import timer

with timer():

  sleep(5.5)
```

```
from time import *

class timer:

  def __enter__(self):

    self.time=time()

  def __exit__(self, exc_type, exc_val, exc_tb):

    print (time()-self.time)
```

Вывод:

5.503533124923706

Задача 6 (ex_6.py)

Мы написали все инструменты для работы с данными. Применим их на реальном примере, который мог возникнуть в жизни. В репозитории находится файл data_light.json . Он содержит облегченный список вакансий в России в формате json (ссылку на полную версию размером ~ 1 Гб. в формате xml можно найти в файле README.md). Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д. В ex_6.py дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print_result печатается результат, а контекстный менеджер timer выводит время работы цепочки функций. Задача реализовать все 4 функции по заданию, ничего не изменяя в файле-шаблоне. Функции f1-f3 должны быть реализованы в 1 строку, функция f4 может состоять максимум из 3 строк.

Что функции должны делать:

1. Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.

2. Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами нужно получить все специальности, связанные с программированием

3. Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python

4. Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб

```
#!/usr/bin/env python3

import json

import sys

from librip.ctxmngrs import timer

from librip.decorators import print_result

from librip.gen import field, gen_random

from librip.iterators import Unique as unique

path = 'data_light.json'

# Здесь необходимо в переменную path получить

# путь до файла, который был передан при запуске

with open(path) as f:

  data = json.load(f)

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`

# Важно!

# Функции с 1 по 3 дожны быть реализованы в одну строку

# В реализации функции 4 может быть до 3 строк

# При этом строки должны быть не длиннее 80 символов

@print_result

def f1(arg):

  return list(unique(list(field(arg, "job-name")), ignore_case=True))

@print_result

def f2(arg):

  return [x for x in arg if x.upper().startswith("ПРОГРАММИСТ")]

@print_result

def f3(arg):

  return ["{} с опытом Python". format(x) for x in arg]

@print_result

def f4(arg):

  rand = gen_random(100000, 200000, len(arg))

  return ["{}, зарплата {} руб.".format(x[0],x[1]) for x in zip(arg,rand)]
```

```
with timer():
  f4(f3(f2(f1(data))))
```

Вывод:

F1 очень большой, пропустим его…

f2

Программист

Программист C++/C#/Java

программист

Программист 1С

Программистр-разработчик информационных систем

Программист C++

Программист/ Junior Developer

Программист / Senior Developer

Программист/ технический специалист

программист 1С

Программист C#

f3

Программист с опытом Python

Программист C++/C#/Java с опытом Python

программист с опытом Python

Программист 1С с опытом Python

Программистр-разработчик информационных систем с опытом Python

Программист C++ с опытом Python

Программист/ Junior Developer с опытом Python

Программист / Senior Developer с опытом Python

Программист/ технический специалист с опытом Python

программист 1С с опытом Python

Программист C# с опытом Python

f4

Программист с опытом Python, зарплата 164900 руб.

Программист C++/C#/Java с опытом Python, зарплата 197854 руб.

программист с опытом Python, зарплата 125830 руб.

Программист 1С с опытом Python, зарплата 167544 руб.

Программистр-разработчик информационных систем с опытом Python, зарплата 175924 руб.

Программист C++ с опытом Python, зарплата 110101 руб.

Программист/ Junior Developer с опытом Python, зарплата 189670 руб.

Программист / Senior Developer с опытом Python, зарплата 118814 руб.

Программист/ технический специалист с опытом Python, зарплата 151313 руб.

программист 1С с опытом Python, зарплата 140785 руб.

Программист C# с опытом Python, зарплата 139587 руб.

0.0299530029296875