

Утверждаю: _____

"__" _____ 2016 г.

Согласовано: _____

"__" _____ 2016 г.

«Введение в python»

Отчет по лабораторной работе №2

(вид документа)

писчая бумага формата А4

(вид носителя)

(количество листов)

Исполнитель: студент группы РТ5-51

_____ Коньшин К.И.

"__" _____ 2016 г.

Задание

Важно выполнять все задачи последовательно . С 1 по 5 задачу формируется модуль `librip`, с помощью которого будет выполняться задание 6 на реальных данных из жизни. Весь вывод на экран (даже в столбик) необходимо реализовывать одной строкой.

Подготовительный этап 1. Зайти на [github.com](https://github.com/iu5team/ex-lab4) и выполнить fork проекта с заготовленной структурой <https://github.com/iu5team/ex-lab4>

2. Переименовать репозиторий в `lab_4` 3. Выполнить `git clone` проекта из вашего репозитория

Задача 1 (`ex_1.py`)

Необходимо реализовать генераторы `field` и `gen_random`. Генератор `field` последовательно выдает значения ключей словарей массива
Пример:

```
goods = [ {'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для  
отдыха', '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'}
```

1. В качестве первого аргумента генератор принимает `list` , дальше через `*args` генератор принимает неограниченное кол-во аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно `None` , то элемент пропускается
3. Если передано несколько аргументов, то последовательно выдаются словари, если поле равно `None` , то оно пропускается, если все поля `None` , то пропускается целиком весь элемент

Генератор `gen_random` последовательно выдает заданное количество случайных чисел в заданном диапазоне Пример: `gen_random(1, 3, 5)` должен выдать 5 чисел от 1 до 3, т.е. примерно 2, 2, 3, 2, 1

В `ex_1.py` нужно вывести на экран то, что они выдают *одной строкой*. Генераторы должны располагаться в `librip/gen.py`

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`

Пример:

`data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]` `Unique(data)` будет последовательно возвращать только 1 и 2
`data = gen_random(1, 3, 10)` `unique(gen_random(1, 3, 10))` будет последовательно возвращать только 1, 2 и 3
`data = ['a', 'A', 'b', 'B']` `Unique(data)` будет последовательно возвращать только a, A, b, B
`data = ['a', 'A', 'b', 'B']` `Unique(data, ignore_case=True)` будет последовательно возвращать только a, b

В `ex_2.py` нужно вывести на экран то, что они выдают *одной строкой*. **Важно** продемонстрировать работу как с массивами, так и с генераторами (`gen_random`). Итератор должен располагаться в `librip/iterators.py`

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

Пример: data = [4, -30, 100, -100, 123, 1, 0, -1, -4] 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 выводиться в столбик через знак равно Пример:

```
@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()
На консоль выведется:
test_1
1
test_2
iu
test_3
a=1
b=2
test_4
1
```

2 2 Декоратор должен располагаться в `librip/ decorators .py`

Задача 5 (`ex_5.py`)

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

```
with timer():  
    sleep(5.5)
```

После завершения блока должно вывестись в консоль примерно 5.5

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.*

Исходный код

Файл ex_1.py

```
1  #!/usr/bin/env python3
2  from librip.gen import *
3  from librip.iterators import *
4
5  goods = [
6      {'title': 'Ковер', 'price': 2000, 'color': 'green'},
7      {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
8      {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
9      {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
10 ]
11 # Реализация задания 1
12 print(list(field(goods, 'title', 'color')))
13 print(list(gen_random(1, 4, 5)))
14
```

Файл ex_2.py

```

1  #!/usr/bin/env python3
2  from librip.gen import gen_random
3  from librip.iterators import Unique
4
5  data1 = [1, 1, 1, 1, 2, 1, 1, 2, 12, 12, 1, 211]
6  data2 = gen_random(1, 3, 10)
7  data3 = ['K', 'O', 'S', 'T', 'Y', 'A', '', 'g', 'o', 'd']
8  # Реализация задания 2
9
10 print(list(Unique(data1)))
11
12 print(list(Unique(list(data2))))
13
14 print(list(Unique(data3, ignore_case=True)))
15 |

```

Файл ex_3.py

```

1  #!/usr/bin/env python3
2
3  data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
4  # Реализация задания 3
5
6  print(sorted(data, key=lambda m: abs(m)))
7

```

Файл ex_4.py

```

1  from librip.decorators import print_result
2
3  # Необходимо верно реализовать print_result
4  # и задание будет выполнено
5
6  @print_result
7  def test_1():
8      return 1
9
10
11  @print_result
12  def test_2():
13      return 'iu'
14
15
16  @print_result
17  def test_3():
18      return {'a': 1, 'b': 2}
19
20
21  @print_result
22  def test_4():
23      return [1, 2]
24
25
26  test_1()
27  test_2()
28  test_3()
29  test_4()
30

```

Файл ex_5.py

```

1  from time import sleep
2  from librip.ctxmgrs import timer
3
4  with timer():
5      sleep(5.5)
6

```

Файл ex_6.py


```

1  #!/usr/bin/env python3
2  import json
3  import sys
4  from librip.ctxmgrs import timer
5  from librip.decorators import print_result
6  from librip.gen import field, gen_random
7  from librip.iterators import Unique as unique
8
9  assert len(sys.argv) > 0
10 path = sys.argv[1]
11
12 # Здесь необходимо в переменную path получить
13 # путь до файла, который был передан при запуске
14
15 with open(path) as f:
16     data = json.load(f)
17
18
19 # Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`
20 # Важно!
21 # Функции с 1 по 3 должны быть реализованы в одну строку
22 # В реализации функции 4 может быть до 3 строк
23 # При этом строки должны быть не длиннее 80 символов
24
25 @print_result
26 def f1(arg):
27     return list(unique(list(field(arg, "job-name")), ignore_case=True))
28
29 @print_result
30 def f2(arg):
31     return list(filter(lambda x: str(x).startswith('Программист'), arg))
32
33 @print_result
34 def f3(arg):
35     return list(map(lambda x: "{} с опытом Python".format(x), arg))
36
37 @print_result
38 def f4(arg):
39     return list(map(lambda x: "{} , зарплата {}".format(*list(x)), zip(arg, gen_random(100000, 200000, len(arg)))))
40
41
42 with timer():
43     f4(f3(f2(f1(data))))
44

```

Файл librip/ctxmgrs.py

```

1  import time
2  # Здесь необходимо реализовать
3  # контекстный менеджер timer
4  # Он не принимает аргументов, после выполнения блока он должен вывести время выполнения в секундах
5  # Пример использования
6  # with timer():
7  #     sleep(5.5)
8  #
9  # После завершения блока должно вывестись в консоль примерно 5.5
10 class timer:
11     t = 0
12     def __enter__(self):
13         self.t = time.clock()
14     def __exit__(self, exp_type, exp_value, traceback):
15         print(time.clock() - self.t)
16

```

Файл librip/decorators.py

```

1 # Здесь необходимо реализовать декоратор, print_result который принимает на вход функцию,
2 # вызывает её, печатает в консоль имя функции, печатает результат и возвращает значение
3 # Если функция вернула список (list), то значения должны выводиться в столбик
4 # Если функция вернула словарь (dict), то ключи и значения должны выводиться в столбик через знак равно
5
6 def print_result(func):
7     def wap(*args):
8         print(func.__name__)
9         if len(args) == 0:
10             cont = func()
11         else:
12             cont = func(args[0])
13
14         if (type(cont) == list):
15             print("\n".join([str(x) for x in cont]))
16         elif (type(cont) == dict):
17             print("\n".join([str(x) + " = " + str(cont[x]) for x in cont]))
18         else:
19             print(cont)
20
21         return cont
22
23     return wap
24

```

Файл librip/gen.py

```

1 import random
2
3
4 # Генератор вычленения полей из массива словарей
5 # Пример:
6 # goods = [
7 #     {'title': 'Ковер', 'price': 2000, 'color': 'green'},
8 #     {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
9 # ]
10 # field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'
11 # field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
12
13 def field(items, *args):
14     lenargs = len(args)
15     assert lenargs > 0
16     # Необходимо реализовать генератор
17     for el in items:
18         buff = {}
19         if (lenargs == 1):
20             if el[args[0]] != None:
21                 yield el[args[0]]
22         else:
23             for ar in args:
24                 if el[ar] != None:
25                     buff[ar] = el[ar]
26             yield buff
27
28
29 # Генератор списка случайных чисел
30 # Пример:
31 # gen_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1
32 # Hint: реализация занимает 2 строки
33 def gen_random(begin, end, num_count):
34     for i in range(num_count):
35         yield random.randint(begin, end)
36     # Необходимо реализовать генератор
37

```

Файл librip/iterators.py

```

1 # Итератор для удаления дубликатов
2 class Unique(object):
3     ignore_case = False
4     i = 0
5     buff = []
6
7     def __init__(self, items, **kwargs):
8         # Нужно реализовать конструктор
9         # В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore_case,
10        # в зависимости от значения которого будут считаться одинаковые строки в разном регистре
11        # Например: ignore_case = True, Абв и АБВ разные строки
12        # ignore_case = False, Абв и АБВ одинаковые строки, одна из них удалится
13        # По-умолчанию ignore_case = False
14        self.ignore_case = False
15        if (len(kwargs) > 0 and kwargs['ignore_case'] != None) and (type(kwargs['ignore_case']) == bool):
16            self.ignore_case = kwargs['ignore_case']
17        self.arr = items
18    def __next__(self):
19        while self.i < len(self.arr):
20            temp = str(self.arr[self.i])
21            if (self.ignore_case):
22                temp = temp.lower()
23            if not temp in self.buff:
24                self.buff.append(temp)
25                return self.arr[self.i]
26            self.i += 1
27            raise StopIteration()
28
29
30    def __iter__(self):
31        return self
32

```

Результаты выполнения

Файл ex_1.py, ex_2.py, ex_3.py, ex_4.py, ex_5.py

```

MacBook-Pro-Konsin-3:~ konsinkonstantin$ python3 lab_4/ex_1.py
[{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}, {'title': 'Средаж', 'color': 'white'}, {'title': 'Вешалка для одежды', 'color': 'white'}]
[2, 4, 1, 1, 2]
MacBook-Pro-Konsin-3:~ konsinkonstantin$ python3 lab_4/ex_2.py
[1, 2, 12, 211]
[3]
['K', 'O', 'S', 'T', 'Y', 'A', '', 'g', 'd']
MacBook-Pro-Konsin-3:~ konsinkonstantin$ python3 lab_4/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
MacBook-Pro-Konsin-3:~ konsinkonstantin$ python3 lab_4/ex_4.py
test_1
1
test_2
iu
test_3
b = 2
a = 1
test_4
1
2
MacBook-Pro-Konsin-3:~ konsinkonstantin$ python3 lab_4/ex_5.py
6.00000000000004494e-05
MacBook-Pro-Konsin-3:~ konsinkonstantin$

```

Файл ex_6.py

Разработчик импульсных источников питания
Механик по эксплуатации транспортного отдела
Инженер-технолог по покраске
Бетонщик – арматурщик
главный инженер финансово-экономического отдела
Секретарь судебного заседания в аппарате мирового судьи Железнодорожного судебного района гор
ода Ростова-на-Дону
варщик зефира
варщик мармеладных изделий
Оператор склада
Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
Заведующий музеем в д.Копорье
Документовед
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических си
стем
Менеджер (в промышленности)
f2
Программист
Программист C++/C#/Java
Программист 1C
Программист-разработчик информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1C с опытом Python
Программист-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 C# с опытом Python
f4
Программист с опытом Python, зарплата 184288
Программист C++/C#/Java с опытом Python, зарплата 180019
Программист 1C с опытом Python, зарплата 114639
Программист-разработчик информационных систем с опытом Python, зарплата 190030
Программист C++ с опытом Python, зарплата 172873
Программист/ Junior Developer с опытом Python, зарплата 136429
Программист / Senior Developer с опытом Python, зарплата 107469
Программист/ технический специалист с опытом Python, зарплата 186839
Программист C# с опытом Python, зарплата 126976
0.09451099999999998