

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



Лабораторные работы по курсу:

«Разработка Интернет Приложений»

ЛР3. Python-классы

Исполнитель:

Студент группы РТ5-51

Умряев Д.Т.

Преподаватель:

Гапанюк Ю. Е.

« ____ » _____



Москва 2017 г.

Цель работы:

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`. Итератор **не должен модифицировать** возвращаемые значения.

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

МГТУ им. Н. Э. Баумана, кафедра
ИУ5, курс РИП ЛР №4: Python,
функциональные возможности

t

e

s

t

—

2

i

u

t

e

s

t

—

3

a

=

1

b

=

2

t

e

s

t

—

4

1

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:

```
wit
```

```
h
```

```
t
```

```
i
```

```
m
```

```
e
```

```
r
```

```
(
```

```
)
```

```
:
```

```
s
```

```
l
```

```
e
```

```
e
```

```
p
```

```
(
```

```
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**. Используйте наработки из предыдущих заданий.
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. Для фильтрации используйте функцию `filter`.
3. Функция `f3` должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: ***Программист С# с опытом Python***. Для модификации используйте функцию `map`.
4. Функция `f4` должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: ***Программист С# с опытом Python, зарплата 137287 руб.*** Используйте `zip` для обработки пары специальность — зарплата.

Листинг:

ex_1.py

```
#!/usr/bin/env python3
from librip.gens import field
from librip.gens import gen_random

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'},
    {'title': 'Балалайка', 'price': None, 'color': 'white'},
    {'title': None, 'price': 9, 'color': 'white'},
    {'title': None, 'price': None, 'color': 'white'},
]

# Реализация задания 1
print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
print(list(gen_random(1, 3, 5)))
```

ex_2.py

```
#!/usr/bin/env python3
from librip.gens 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)
data3 = ['a', 'A', 'b', 'B']
data4 = ['Aab', 'aAB', 'BBB', 'bbb', 'bbb']

print(list(Unique(data1)))
print(list(Unique(data2)))

print(list(Unique(data3, ignore_case=False)))
```

```
print(list(Unique(data4, ignore_case=False)))
print(list(Unique(data4, ignore_case=True)))
```

ex_3.py

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
list(map(lambda x: print(x, end=' '), (sorted(data, key=abs))))
```

ex_4.py

```
from librip.decorators import print_result
```

```
@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()
```

ex_5.py

```
from time import sleep
from librip.ctxmgrs import timer

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

ex_6.py

```
import json
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as unique
```

```

path = r"data_light_cp1251.json"

with open(path) as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(unique((field(arg, 'job-name')), ignore_case=True), key=str.lower)

@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith('Программист'), arg))

@print_result
def f3(arg):
    return list(map(lambda x: x + ' с опытом работы Python', arg))

@print_result
def f4(arg):
    return list('{} , зарплата {} руб'.format(x, y) for x, y in zip(arg, list(gen_random(100000, 200000, len(arg)))))

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

```

ctxmgrs.py

```

import contextlib
import time

@contextlib.contextmanager
def timer():
    t1 = time.time()
    yield
    t2 = time.time()
    print(t2-t1)

```

decorators.py

```

def print_result(func_to_decorate):
    def decorated_func(*args):
        result = func_to_decorate(*args)
        print(func_to_decorate.__name__)
        if type(result) in (str, int):
            print(result)
        if type(result) is list:
            print(*result, sep='\n')
        if type(result) is dict:
            for x, y in result.items():
                print('{} = {}'.format(x, y))
    return decorated_func

```



```
    return result
```

```
    return decorated_func
```

gens.py

```
import random
```

```
def field(items_list, *args):
    if args is None:
        print("Вы не передали названия полей")
        return

    if len(items_list) == 0:
        print("Вы передали пустой лист")
        return

    if len(args) == 1:
        for item_dict in items_list:
            if item_dict[args[0]] is not None:
                yield item_dict[args[0]]
    else:
        for item_dict in items_list:
            dictionary = {}
            for arg in args:
                if item_dict.get(arg) is not None:
                    dictionary[arg] = item_dict.get(arg)

            if dictionary != {}:
                yield dictionary

def gen_random(begin, end, num_count):
    for i in range(num_count):
        yield random.randint(begin, end)
```

iterators.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        if type(items) is list:
            self.item = iter(items)
        elif str(type(items)) == '<class \'generator\>':
            self.item = items
        else:
            raise TypeError("Класс Unique принимает на вход list или generator, а не " +
str(type(items)))
        self.unique_item_list = []
        self.ignore_case = kwargs.get('ignore_case')

    def _is_unique(self, item):
        if self.ignore_case:
            item = str(item)
            item = item.lower()
        if item in self.unique_item_list:
```

```

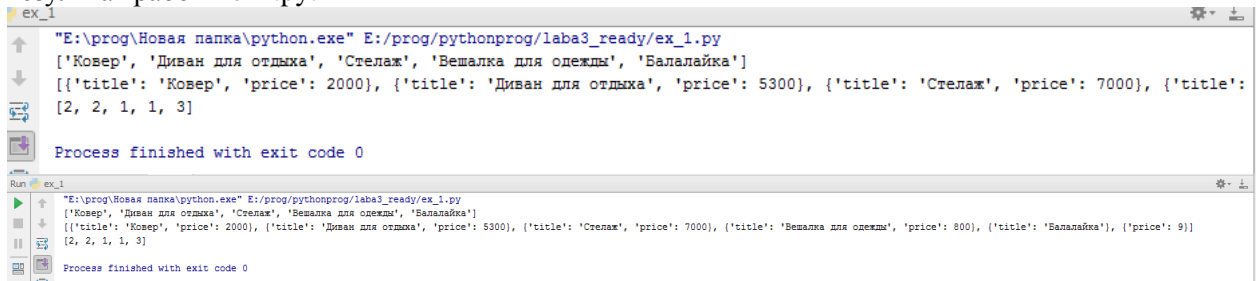
        return False
    else:
        self.unique_item_list.append(item)
        return True

    def __next__(self) :
        value = next(self.item)
        while not self._is_unique(value):
            value = next(self.item)
        return value

    def __iter__(self):
        return self

```

Результат работы ex1.py:



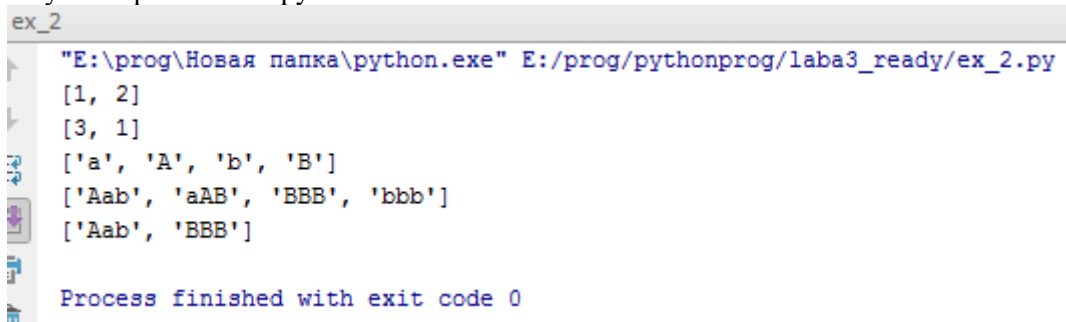
```

ex_1
"E:\prog\Новая папка\python.exe" E:/prog/pythonprog/laba3_ready/ex_1.py
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды', 'Балалайка']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title':
[2, 2, 1, 1, 3]

Process finished with exit code 0

```

Результат работы ex2.py:



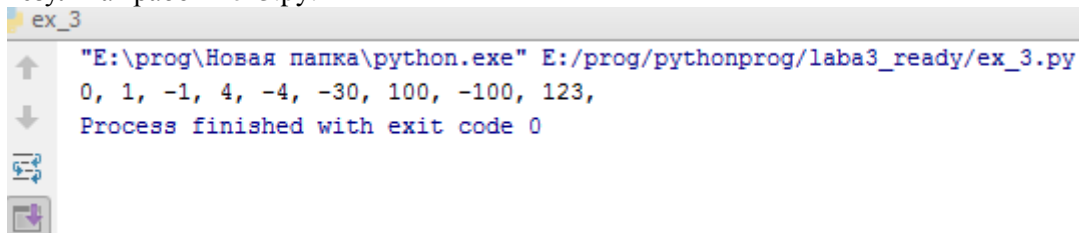
```

ex_2
"E:\prog\Новая папка\python.exe" E:/prog/pythonprog/laba3_ready/ex_2.py
[1, 2]
[3, 1]
['a', 'A', 'b', 'B']
['Aab', 'aAB', 'BBB', 'bbb']
['Aab', 'BBB']

Process finished with exit code 0

```

Результат работы ex3.py:

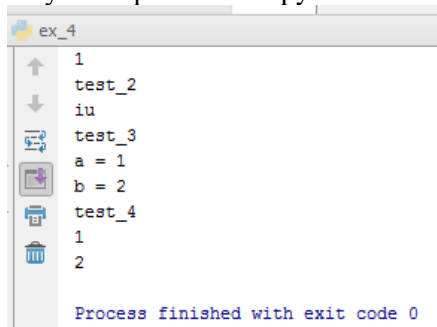


```

ex_3
"E:\prog\Новая папка\python.exe" E:/prog/pythonprog/laba3_ready/ex_3.py
0, 1, -1, 4, -4, -30, 100, -100, 123,
Process finished with exit code 0

```

Результат работы ex4.py:



```

ex_4
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0

```

Результат работы ex5.py:

```
ex_5
"Е:\prog\Новая папка\python.exe" E:/prog/pythonprog/lab3_ready/ex_5.py
5.501314878463745
Process finished with exit code 0
```

Результат работы ex6.py:

```
x_6
Юрисконсульт
юрисконсульт 2 категории
Юрисконсульт. Контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский - разговорный)
Юрист волонтер
Юристконсульт
f2
Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
f3
Программист с опытом работы Python
Программист / Senior Developer с опытом работы Python
Программист 1C с опытом работы Python
Программист C# с опытом работы Python
Программист C++ с опытом работы Python
Программист C++/C#/Java с опытом работы Python
Программист/ Junior Developer с опытом работы Python
Программист/ технический специалист с опытом работы Python
Программист-разработчик информационных систем с опытом работы Python
f4
Программист с опытом работы Python, зарплата 184465 руб
Программист / Senior Developer с опытом работы Python, зарплата 177711 руб
Программист 1C с опытом работы Python, зарплата 199048 руб
Программист C# с опытом работы Python, зарплата 194596 руб
Программист C++ с опытом работы Python, зарплата 140648 руб
Программист C++/C#/Java с опытом работы Python, зарплата 116683 руб
Программист/ Junior Developer с опытом работы Python, зарплата 113813 руб
Программист/ технический специалист с опытом работы Python, зарплата 148735 руб
Программист-разработчик информационных систем с опытом работы Python, зарплата 182102 руб
0.20701146125793457
Process finished with exit code 0
```