

Московский государственный технический университет им.  
Н.Э. Баумана

Факультет “Радиотехнический”  
Кафедра ИУ5 “Системы обработки информации и управления”

Отчет по лаб 3  
“Функциональные возможности языка Python.”  
**Базовые компоненты интернет технологий**

Вариант 8

Подготовил:  
Студент группы РТ5-31Б  
Коваленко В.И.

Проверил:  
Доцент кафедры ИУ5  
Гапанюк Ю.Е.

12 Декабря 2021г.

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете `lab_python_fr`. Решение каждой задачи должно располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача 1 (файл `field.py`)

Необходимо реализовать генератор `field`. Генератор `field` последовательно выдает значения ключей словаря. 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'}
```

В качестве первого аргумента генератор принимает список словарей, дальше через `*args` генератор принимает неограниченное количество аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно `None`, то элемент пропускается.

Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно `None`, то оно пропускается. Если все поля содержат значения `None`, то пропускается элемент целиком.

Шаблон для реализации генератора:

```
# 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
    # Необходимо реализовать генератор
```

### Листинг

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            if args[0] in item and item[args[0]] is not None:
                yield item[args[0]]

    else:
        for item in items:
            dictionary = {}
            for value in args:
                if value in item and item[value] is not None:
                    dictionary[value] = item[value]
            if len(dictionary) != 0:
                yield dictionary

def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    a = field(goods, 'title')
    print(next(a))
    print(next(a))

if __name__ == "__main__":
    main()
```

```
Ковер
Диван для отдыха
kovalenkov@MacBook-Air-Vlad Лаб 3 %
```

## Задача 2 (файл gen\_random.py)

Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Шаблон для реализации генератора:

```
# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
def gen_random(num_count, begin, end):
    pass
    # Необходимо реализовать генератор
```

### Листинг

```
import random

def gen_random(num_count, begin, end):
    cur = 0 #текущее значение
    while cur < num_count:
        cur += 1
        yield random.randint(begin, end)
```

## Задача 3 (файл unique.py)

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

Конструктор итератора также принимает на вход именованный bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

При реализации необходимо использовать конструкцию \*\*kwargs.

Итератор должен поддерживать работу как со списками, так и с генераторами.

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(data) будет последовательно возвращать только 1, 2 и 3.

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

Шаблон для реализации класса-итератора:

```
# Итератор для удаления дубликатов
```

```
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 одинаковыми  
строки в разном регистре
```

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

```
        # ignore_case = False, Абв и АБВ - одинаковые строки, одна из  
которых удалится
```

```
        # По-умолчанию ignore_case = False
```

```
        pass
```

```
    def __next__(self):
```

```
        # Нужно реализовать __next__
```

```
        pass
```

```
    def __iter__(self):
```

```
        return self
```

### Листинг

```
from gen_random import gen_random
```

```
class Unique(object):
```

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

```
        self.used_elements = set()
```

```

self.data = items
self.ignore_case = False
if len(kwargs) > 0:
    self.ignore_case = kwargs['ignore_case']

def __next__(self):
    it = iter(self.data)
    while True:
        try:
            cur = next(it)
        except StopIteration:
            raise StopIteration
        else:
            if self.ignore_case is True and isinstance(cur, str):
                cur = cur.lower()
            if cur not in self.used_elements:
                self.used_elements.add(cur)
            return cur

def __iter__(self):
    return self

def main():
    data = gen_random(3, 1, 2)
    iter = Unique(data, ignore_case=True)
    for i in iter:
        print(i)
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    iter = Unique(data, ignore_case=False)
    for i in iter:
        print(i)

if __name__ == "__main__":
    main()

```

```

2
a
A
b
B
kovalenkov@MacBook-Air-Vlad lab 3 %

```

#### Задача 4 (файл sort.py)

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted. Пример:

```
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
```

```
Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
```

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

Без использования lambda-функции.

Шаблон реализации:

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
```

```
if __name__ == '__main__':  
    result = ...  
    print(result)
```

```
    result_with_lambda = ...  
    print(result_with_lambda)
```

#### Листинг

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
```

```
if __name__ == '__main__':  
    result = sorted(data, key=abs, reverse=True)  
    print(result)
```

```
    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
    print(result_with_lambda)
```

```
[123, 100, -100, -30, 4, -4, 1, -1, 0]  
[123, 100, -100, -30, 4, -4, 1, -1, 0]  
kovalenkov@MacBook-Air-Vlad Лаб 3 %
```

#### Задача 5 (файл print\_result.py)

Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.

Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводиться в столбик через знак равенства.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора
```

```
@print_result
def test_1():
    return 1
```

```
@print_result
def test_2():
    return 'iu5'
```

```
@print_result
def test_3():
    return {'a': 1, 'b': 2}
```

```
@print_result
def test_4():
    return [1, 2]
```

```
if __name__ == '__main__':
    print('!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат выполнения:

```
test_1
1
test_2
iu5
test_3
a = 1
```



```
b = 2
test_4
1
2
```

## Листинг

```
import functools
#декораторы

def print_result(func):
    @functools.wraps(func)
    def wrapped(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if type(result) is list:
            print(*result, sep='\n')
        elif type(result) is dict:
            for k, v in result.items():
                print('{} = {}'.format(k, v))
        else:
            print(result)
        return result

    return wrapped

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu5'

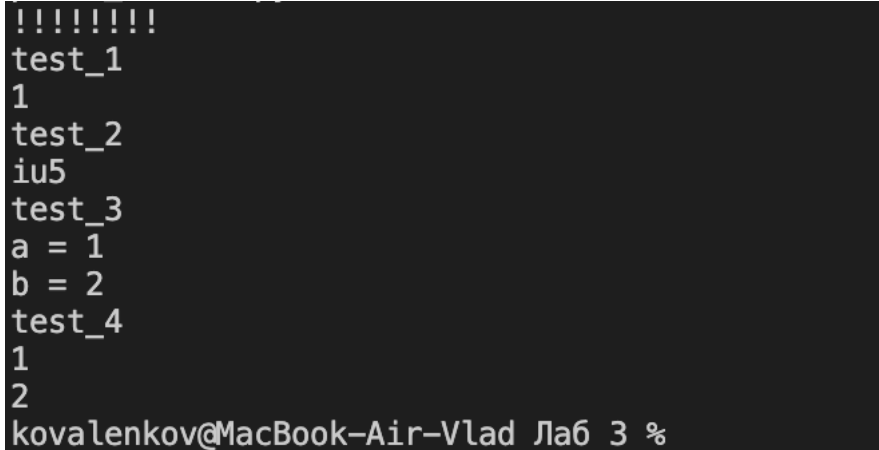
@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 2]
```

```

if __name__ == '__main__':
    print('!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()

```



```

!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
kovalenkov@MacBook-Air-Vlad Лаб 3 %

```

#### Задача 6 (файл cm\_timer.py)

Необходимо написать контекстные менеджеры `cm_timer_1` и `cm_timer_2`, которые считают время работы блока кода и выводят его на экран. Пример:

```

with cm_timer_1():
    sleep(5.5)

```

После завершения блока кода в консоль должно вывестись `time: 5.5` (реальное время может несколько отличаться).

`cm_timer_1` и `cm_timer_2` реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки `contextlib`).

#### Листинг

```

import time
from contextlib import contextmanager

```

```

class cm_timer_1:
    def __init__(self):
        self.start = time.time()

    def __enter__(self):

```

```

        return self

    def __exit__(self, *args):
        print('time: {}'.format(time.time() - self.start))

    @contextmanager
    def cm_timer_2():
        cur = time.time()
        yield cur
        print('time: {}'.format(time.time() - cur))

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(5.5)

    with cm_timer_2():
        time.sleep(5.5)

```

```

time: 5.502991199493408
time: 5.511736869812012
kovalenkov@MacBook-Air-Vlad Лаб 3 %

```

## Задача 7 (файл process\_data.py)

В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.

В файле data\_light.json содержится фрагмент списка вакансий.

Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print\_result печатается результат, а контекстный менеджер cm\_timer\_1 выводит время работы цепочки функций.

Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.

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. Для модификации используйте функцию map.

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. Используйте zip для обработки пары специальность — зарплата.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

# Необходимо в переменную path сохранить путь к файлу, который был
передан при запуске сценария

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

# Далее необходимо реализовать все функции по заданию, заменив `raise
NotImplemented`
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
# В реализации функции f4 может быть до 3 строк

@print_result
def f1(arg):
    raise NotImplementedError

@print_result
def f2(arg):
    raise NotImplementedError

@print_result
```

```
def f3(arg):  
    raise NotImplemented
```

```
@print_result  
def f4(arg):  
    raise NotImplemented
```

```
if __name__ == '__main__':  
    with cm_timer_1():  
        f4(f3(f2(f1(data))))
```

### Листинг

```
import json  
from print_result import print_result  
import unique  
from gen_random import gen_random  
from cm_timer import cm_timer_1  
# Сделаем другие необходимые импорты  
  
path =  
"/Users/kovalenkov/Documents/GitHub/BKIT_2021/notebooks/fp/files/data_light.  
json"  
  
with open(path, "r", encoding='utf8') as f:  
    data = json.load(f)  
    args = (job["job-name"] for job in data)  
  
@print_result  
def f1(args):  
    return sorted(unique.Unique(args, ignore_case=False).data)  
  
@print_result  
def f2(arg):  
    return list(filter(lambda x: x.lower().startswith("программист") is True, arg))  
  
@print_result  
def f3(arg):  
    return list(map(lambda x: x + ' с опытом Python', arg))  
  
@print_result
```

```
def f4(arg):  
    return list(zip(arg, list(gen_random(len(arg), 100000, 200000))))
```

```
if __name__ == '__main__':  
    with cm_timer_1():  
        f4(f3(f2(f1(args))))
```

```
('Программист с опытом Python', 120598)  
( 'Программист с опытом Python', 182416)  
( 'Программист с опытом Python', 107780)  
( 'Программист / Senior Developer с опытом Python', 191892)  
( 'Программист 1С с опытом Python', 106344)  
( 'Программист 1С с опытом Python', 113642)  
( 'Программист C# с опытом Python', 111572)  
( 'Программист C++ с опытом Python', 197427)  
( 'Программист C++/C#/Java с опытом Python', 199160)  
( 'Программист/ Junior Developer с опытом Python', 135617)  
( 'Программист/ технический специалист с опытом Python', 142139)  
( 'Программист-разработчик информационных систем с опытом Python', 155203)  
( 'программист с опытом Python', 115281)  
( 'программист с опытом Python', 177291)  
( 'программист с опытом Python', 115359)  
( 'программист 1С с опытом Python', 165026)  
time: 0.09711194038391113  
kovalenkov@MacBook-Air-Vlad Лаб 3 %
```