

Московский государственный технический университет им.
Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет технологий»

Отчет по лабораторным работам №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-32Б: Кузьмин Я.А.

Руководитель:
преподаватель каф. ИУ5 Гапанюк Ю.Е.

Москва, 2022 г.

Цель лабораторной работы:

Изучение возможностей функционального программирования в языке Python.

Задание лабораторной работы: Задание лабораторной работы состоит из решения нескольких задач.

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

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задание №1:

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

Текст программы: `lab_python_fp/field.py`

Пример:

`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]

    return [ dict[arg] for dict in items if arg in dict and dict[arg] != None]
else:
    res = [{ arg:dict[arg] for arg in args if arg in dict and dict[arg] != None}
    for dict in items ]
    return list(filter(lambda dict: len(dict) != 0, res))

if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]

    print(field(goods, 'title', 'price'))
    print(field(goods, 'title'))

```

Примеры выполнения:

```

yarik_tri@LAPTOP-MN0K6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 field.py
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
['Ковер', 'Диван для отдыха']

```

Задание №2:

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.

Текст программы: `lab_python_fp/gen_random.py`

```

from random import randint

```

```

# 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

```

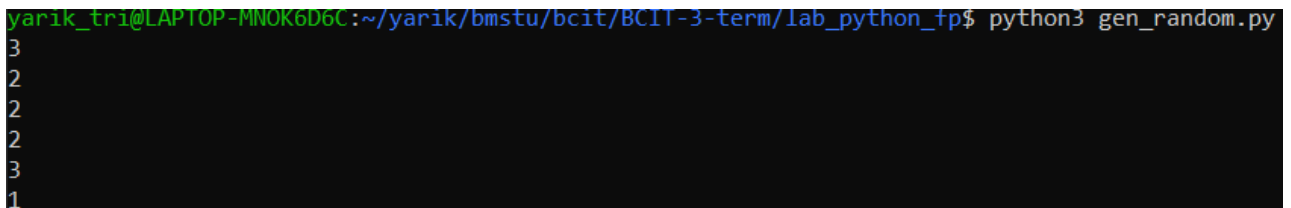
```
def gen_random_1(num_count, begin, end):  
    return (randint(begin, end) for i in range(num_count))
```

```
def gen_random_2(num_count, begin,  
end):  
    for i in range(num_count):  
yield randint(begin, end)
```

```
gen_random = gen_random_1
```

```
if __name__ == "__main__":  
for i in gen_random(6, 1, 3):  
    print(i)
```

Примеры выполнения:



```
yarik_tri@LAPTOP-MNOK6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 gen_random.py  
3  
2  
2  
2  
3  
1
```

Задание №3:

- 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.

Текст программы: lab_python_fp/unique.py

```
from gen_random import gen_random
```

```
class Unique(object):  
    def  
__init__(self, items, **kwargs):
```

```

self.used_items = set()
self.iterator = iter(items)

    if 'ignore_case' in kwargs and kwargs['ignore_case'] == True:
        self.ignore_case = True
    else:
        self.ignore_case = False

    def
__next__(self):
    while True:
        item = None
        try:
            item =
next(self.iterator)
        except
StopIteration:
            raise
StopIteration

        item_original = item
    if self.ignore_case:
        item = str(item).lower()

        if item not in self.used_items:
            self.used_items.add(item)
        return item_original

    def __iter__(self):
        return self

if __name__ == "__main__":
    # data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 'A', 3, 'a', '1567', 1567, 'AAb',
'AAB', 'aAb', 'aab']
    data =
gen_random(100, 1, 3)

    for i in Unique(data, ignore_case=True):
        print(i)

```

Примеры выполнения:

```
yarik_tri@LAPTOP-MNOK6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 unique.py
3
2
1
```

Задание №4:

Дан массив 1, содержащий положительные и отрицательные числа.

Необходимо одной строкой кода вывести на экран массив 2, который содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции `sorted`.

Необходимо решить задачу двумя способами:

1. С использованием `lambda`-функции.
2. Без использования `lambda`-функции.

Текст программы: `lab_python_fp/sort.py`

```
data = [4, -30, 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 i: abs(i), reverse
    = True)
    print(result_with_lambda)
```

Примеры выполнения:

```
yarik_tri@LAPTOP-MNOK6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 sort.py
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
```

Задание №5:

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 выводить в столбик через знак равенства.

Текст программы: lab_python_fp/print_result.py

```
# Здесь должна быть реализация декоратора

def print_result(func):
    def wrapper(*args,
    **kwargs):
        print(func.__name__)
        res
        = func(*args, **kwargs)

        if isinstance(res, list) or isinstance(res,
        tuple):
            for i in res:
                print(i)
            elif isinstance(res, dict):
                for key, value in res.items():
                    print('{} = {}'.format(key, value))
                else:
                    print(res)

        return res
    return wrapper
```

```
@print_result
def test_1(a, b):
    return a + b
```

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

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

```
@print_result
```

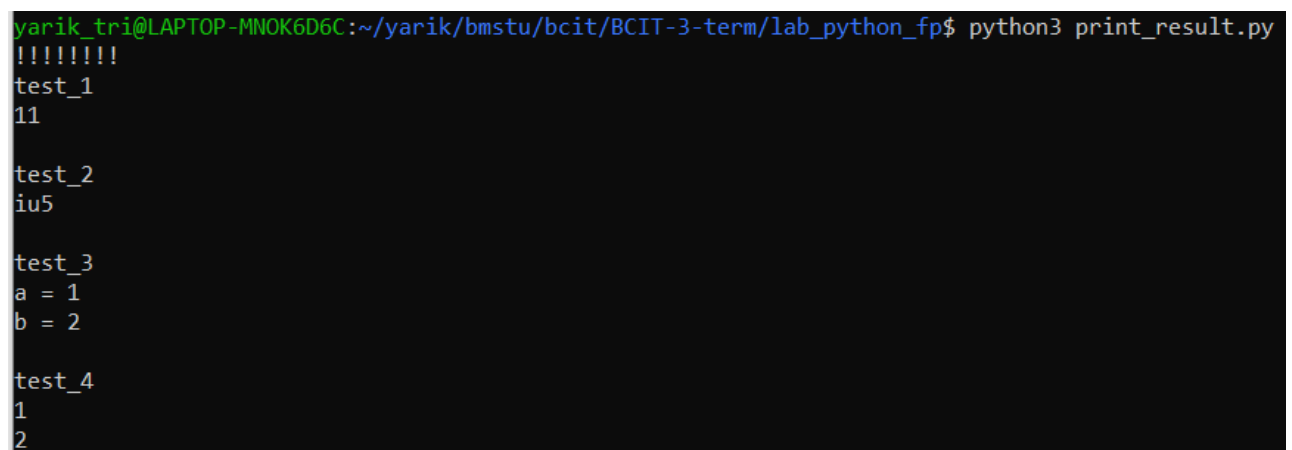
```

def test_4():
    return [1, 2]

if __name__ == '__main__':
    print('!!!!!!!')
    test_1(5, 6)
    print()
    test_2()
    print()
    test_3()
    print()
    test_4()

```

Примеры выполнения:



```

yarik_tri@LAPTOP-MNOK6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 print_result.py
!!!!!!!
test_1
11
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

```

Задание №6:

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

После завершения блока кода в консоль должно вывестись `time: 2.5` (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`).

Текст программы: `lab_python_fp/cm_timer.py`

```

import time from contextlib import
contextmanager

```



```

class cm_timer_1:
    def __init__(self):
        self.start = 0

    def __enter__(self):
        self.start = time.time()

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            end = time.time()
            t = end - self.start
            print("time = {}".format(t))

```

```

@contextmanager
def cm_timer_2():
    start = time.time()
    yield None
    end = time.time()
    t = end - start
    print("time = {}".format(t))

```

```

if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(1.5)
        with cm_timer_2():
            time.sleep(1.3)

```

Примеры выполнения:

```

yarik_tri@LAPTOP-MNOK6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 cm_timer.py
time = 1.500636100769043
time = 1.3015244007110596

```

Задание №7:

- 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.

Текст программы:

```
lab_python_fp/process_data.py
```

```
import json
```

```
import sys
```

```
from field import field
```

```
from gen_random import gen_random
```

```
from unique import Unique from
```

```
print_result import print_result
```

```
from cm_timer import cm_timer_1
```

```
try:
```

```
    path = sys.argv[1] except:
```

```
    path = "data_light.json"
```

```
finally:    print(path)
```

```
with open(path, 'r', encoding='utf8') as f:
```

```
    data = json.load(f)
```

```
@print_result
```

```
def f1(arg):
```

```
    return sorted([i for i in Unique(field(data, 'job-name'),  
ignore_case=True)],
```

```
                    key = lambda x: x.lower())
```

```
@print_result
```

```
def f2(arg):
```

```
    return list(filter(lambda str:  
str.lower().startswith('программист'), arg))
```

```
@print_result def
```

```
f3(arg):    return
```

```
list(map(lambda
```

```
str: str + ' с
```

```
опытом Python',
```

```
arg))
```

```
@print_result
```

```
def f4(arg):
```

```

    job_salary = zip(arg, gen_random(len(arg), 100000, 200000))

    return [{"{}", зарплата {} руб.".format(job, salary) for job, salary
in job_salary]

```

```

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

```

Примеры выполнения:

```

yarik_tri@LAPTOP-MNOK6D6C:~/yarik/bmstu/bcit/BCIT-3-term/lab_python_fp$ python3 process_data.py
../data_light.json
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестящик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
автослесарь
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости

```

Далее следует ещё огромное множество вакансий
После чего следующий результат функций f2, f3, f4:

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 Python
Программист / Senior Developer с опытом Python
Программист 1C с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист-разработчик информационных систем с опытом Python

f4

Программист с опытом Python, зарплата 133735 руб.
Программист / Senior Developer с опытом Python, зарплата 108933 руб.
Программист 1C с опытом Python, зарплата 121630 руб.
Программист C# с опытом Python, зарплата 193725 руб.
Программист C++ с опытом Python, зарплата 160649 руб.
Программист C++/C#/Java с опытом Python, зарплата 177618 руб.
Программист/ Junior Developer с опытом Python, зарплата 175752 руб.
Программист/ технический специалист с опытом Python, зарплата 142327 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 181747 руб.
time = 0.03422212600708008