

**Московский государственный технический  
университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3  
«Функциональные возможности языка Python»

Выполнил:  
студент группы ИУ5-54Б  
Киреев А.А.  
Подпись и дата:

Москва, 2021 г.

**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.

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.

**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.

**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.

#### **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`.

#### **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.

#### **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.

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`).

#### **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.

### Текст программы.

файл “field.py”

```
def field(items, *args):          #items - список словарей, args - аргументы -
    #ключи словаря

    assert len(args) > 0
    if len(args) == 1:            #Если передан один элемент - выдаются только
    #соответствующее значение для ключа словаря
        for i in items:
            if args[0] in i.keys() and not i[args[0]] is None:          #Проверка
    #на наличие ключа в данном словаре и на отсутствие значения None для
    #поля
                yield i[args[0]]
    else:                          #Если передано несколько аргументов
        for i in items:
            tmp_dict = {}          #Временный локальный словарь для заполнения и
    #вывода
            for key in args:
                if key in i.keys() and not i[key] is None:
                    tmp_dict[key] = i[key]
            if len(tmp_dict) > 0:
```

```

        yield tmp_dict

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    print(str(list(field(goods, 'title')))[1:-1])
    print(str(list(field(goods, 'title', 'price')))[1:-1])

```

файл “gen\_random.py”

```

import random

def gen_random(num_count, begin, end):
    #Генератор определенного
    количества случайных чисел
    for i in range(num_count):
        yield random.randint(begin, end)    #Использование пакета для
    генерации случайных переменных и метода для целых чисел

if __name__ == '__main__':
    print(str(list(gen_random(5, 1, 4)))[1:-1])

```

файл “unique.py”

```

import random

class Unique:

    def __init__(self, items, **kwargs):
        #Конструктор класса
        self.used_el = set()    #Множество уже использованных
    элементов
        self.data = list(items)    #Список с данными для
    проверки
        self.index = 0    #Индекс для прохода по списку
    с данными
        if 'ignore_case' in kwargs.keys() and kwargs['ignore_case'] ==
    True:    #Проверка на учет регистра
            self.ignore_case = True
        else:
            self.ignore_case = False

    def __next__(self):
        #Метод следующего элемента
        while True:
            if self.index >= len(self.data):
                #Если еще есть данные в
    списке
                raise StopIteration

```

```

        cur = self.data[self.index]
        self.index += 1
        if ((self.ignore_case or not isinstance(cur, str)) and cur not in
self.used_el):    #Проверка является ли cur элементом класса и был ли он уже
использован
            self.used_el.add(cur)
            return cur
        elif (not self.ignore_case and isinstance(cur, str)    #Учет
регистра
                and cur.upper() not in self.used_el
                and cur.lower() not in self.used_el):
            self.used_el.add(cur.upper())
            self.used_el.add(cur.lower())
            return cur

    def __iter__(self):    #Метод итератор
        return self

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

if __name__ == '__main__':
    data_int = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data_rand = gen_random(10, 3, 10)
    data_str = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print('Отбор уникальных чисел: ', str(list(Unique(data_int)))[1:-1]))
    print('Отбор уникальных случайных чисел: ', str(list(Unique(data_rand)))[1:-
1]))
    print('Отбор уникальных строк без игнорирования регистра по умолчанию: ',
str(list(Unique(data_str)))[1:-1]))
    print('Отбор уникальных строк (регистр учитывается): ',
str(list(Unique(data_str, ignore_case=True)))[1:-1]))
    print('Отбор уникальных строк (регистр НЕ учитывается): ',
str(list(Unique(data_str, ignore_case=False)))[1:-1]))

```

файл “sort.py”

```

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

if __name__ == '__main__':
    print('Без использования lambda-функции: ', sorted(data, key=abs,
reverse=True))
    print('С использованием lambda-функции: ', sorted(data, key = lambda x: x if
x >= 0 else -x, reverse=True))

```

файл “print\_result.py”

```
def print_result(func):          #Декоратор для print_result
    def decorated_func(*args):
        print(func.__name__)
        return_value = func(*args) #Вызов исходной функции
        if isinstance(return_value, list): #Проверка является ли результат
СПИСКОМ
            for value in return_value:
                print(str(value))
            elif isinstance(return_value, dict): #Проверка является ли результат
словарем
                for key in return_value.keys():
                    print(str(key) + ' = ' + str(return_value[key]))

            else:
                print(return_value) #Если не словарь и не список, то переменная
        return return_value
    return decorated_func

@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()
```

файл “cm\_timer.py”

```
import time
from contextlib import contextmanager

class cm_timer_1:      #Таймер на основе класса

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(cm_timer_1.__name__, time.time() - self.start_time)

@contextmanager
def cm_timer_2():      #Таймер на основе готовой библиотеки
    start_time = time.time()
    yield
    print(cm_timer_2.__name__, time.time() - start_time)

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

    with cm_timer_2():
        time.sleep(5.5)
```

файл “process\_data.py”

```
import json
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

path = '../lab3/data_light.json'

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

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

@print_result
def f1(arg):
```



```

        return sorted(Unique(field(arg, 'job-name')))

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

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

@print_result
def f4(arg):
    gen_salary = list(gen_random(len(arg), 100000, 200000))
    work_and_salary = list(zip(arg, gen_salary))
    return list(map(lambda x: x[0] + ', зарплата ' + str(x[1]) + ' руб',
work_and_salary))

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

```

## Результаты работы программы:

### “field.py”

```

(venv) andrey@andrey-Aspire-E5-575G:~/Документы/lab3$ cd /home/andrey/Документы/lab3 ; /usr/bin/env /home/andrey/Документ
ы/lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/
launcher 43085 -- /home/andrey/Документы/lab3/lab_python_fp/field.py
'Ковер', 'Диван для отдыха'
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

```

### “gen\_random.py”

```

(venv) andrey@andrey-Aspire-E5-575G:~/Документы/lab3$ cd /home/andrey/Документы/lab3 ; /usr/bin/env /home/andrey/Документ
ы/lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/
launcher 38675 -- /home/andrey/Документы/lab3/lab_python_fp/gen_random.py.py
2, 4, 2, 1, 3
(venv) andrey@andrey-Aspire-E5-575G:~/Документы/lab3$ cd /home/andrey/Документы/lab3 ; /usr/bin/env /home/andrey/Документ
ы/lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/
launcher 34167 -- /home/andrey/Документы/lab3/lab_python_fp/gen_random.py.py
2, 4, 2, 1, 2

```

### “unique.py”

```

(venv) andrey@andrey-Aspire-E5-575G:~/Документы/lab3$ cd /home/andrey/Документы/lab3 ; /usr/bin/env /home/andrey/Документ
ы/lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/
launcher 41927 -- /home/andrey/Документы/lab3/lab_python_fp/unique.py
Отбор уникальных чисел: 1, 2
Отбор уникальных случайных чисел: 10, 7, 5, 8, 4, 6
Отбор уникальных строк без игнорирования регистра по умолчанию: 'a', 'b'
Отбор уникальных строк (регистр учитывается): 'a', 'A', 'b', 'B'
Отбор уникальных строк (регистр НЕ учитывается): 'a', 'b'

```

### “sort.py”

```

(venv) andrey@andrey-Aspire-E5-575G:~/Документы/lab3$ cd /home/andrey/Документы/lab3 ; /usr/bin/env /home/andrey/Документ
ы/lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/
launcher 33887 -- /home/andrey/Документы/lab3/lab_python_fp/sort.py
Без использования lambda-функции: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
С использованием lambda-функции: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

```

## “print\_result.py”

```
(venv) andrey@andrey-Aspire-E5-575G:~/Документы/Lab3$ cd /home/andrey/Документы/Lab3 ; /usr/bin/env /home/andrey/Документы/Lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/launcher 46817 -- /home/andrey/Документы/Lab3/lab_python_fp/print_result.py

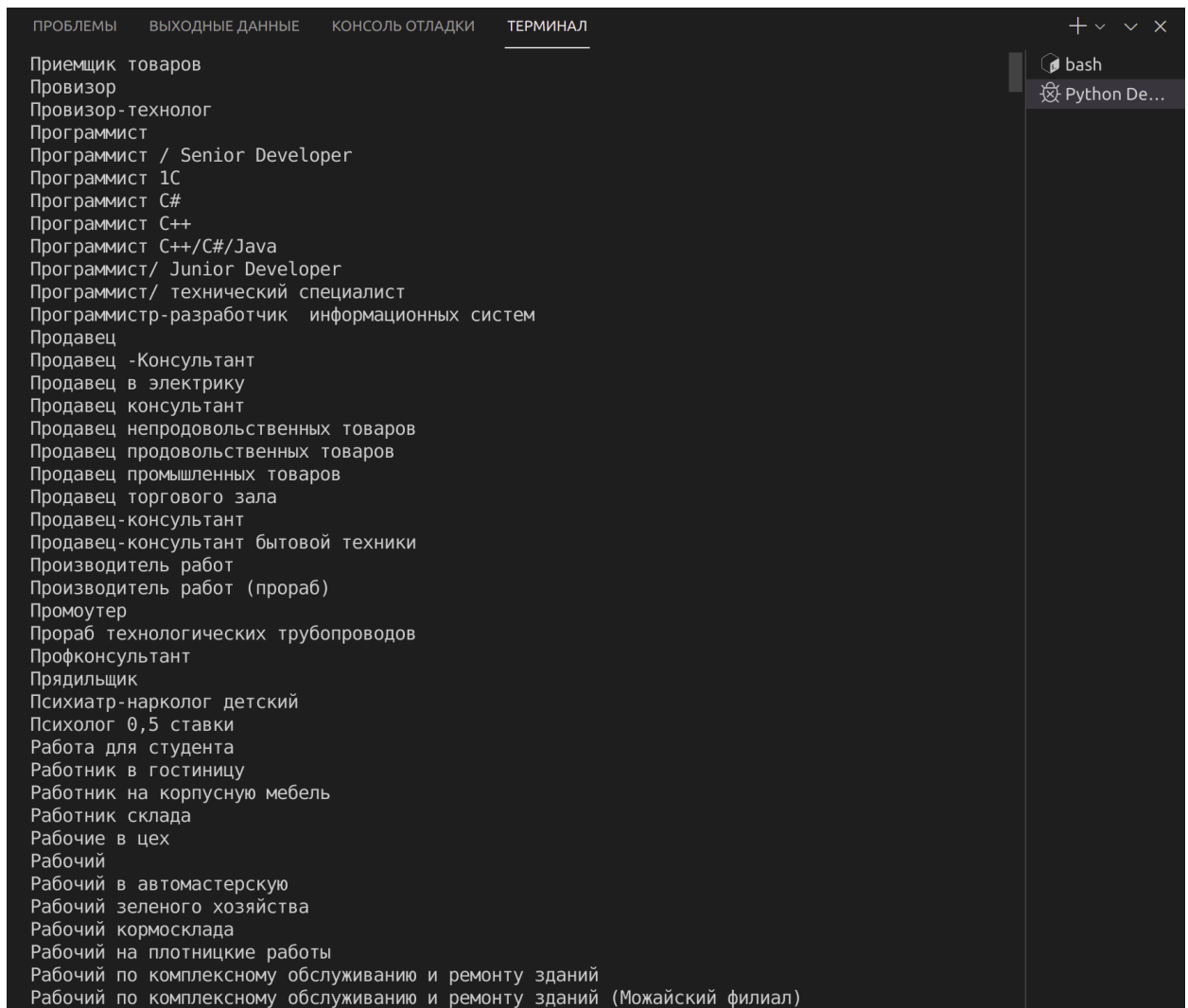
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

## “cm\_timer.py”

```
(venv) andrey@andrey-Aspire-E5-575G:~/Документы/Lab3$ cd /home/andrey/Документы/Lab3 ; /usr/bin/env /home/andrey/Документы/Lab3/venv/bin/python /home/andrey/.vscode/extensions/ms-python.python-2021.10.1365161279/pythonFiles/lib/python/debugpy/launcher 37993 -- /home/andrey/Документы/Lab3/lab_python_fp/cm_timer.py
cm_timer 1 5.506101131439209
cm_timer 2 5.506587505340576
```

## “process\_data.py”

(часть вывода)



The screenshot shows a VS Code interface with a terminal window open. The terminal displays a list of professions in Russian, one per line. The list includes various roles such as 'Приемщик товаров', 'Провизор', 'Программист', 'Продавец', 'Работник склада', and 'Рабочий'. The terminal window has tabs for 'bash' and 'Python De...'. The list of professions is as follows:

- Приемщик товаров
- Провизор
- Провизор-технолог
- Программист
- Программист / Senior Developer
- Программист 1C
- Программист C#
- Программист C++
- Программист C++/C#/Java
- Программист/ Junior Developer
- Программист/ технический специалист
- Программист-разработчик информационных систем
- Продавец
- Продавец -Консультант
- Продавец в электрику
- Продавец консультант
- Продавец непродовольственных товаров
- Продавец продовольственных товаров
- Продавец промышленных товаров
- Продавец торгового зала
- Продавец-консультант
- Продавец-консультант бытовой техники
- Производитель работ
- Производитель работ (прораб)
- Промоутер
- Прораб технологических трубопроводов
- Профконсультант
- Прядильщик
- Психиатр-нарколог детский
- Психолог 0,5 ставки
- Работа для студента
- Работник в гостиницу
- Работник на корпусную мебель
- Работник склада
- Рабочие в цех
- Рабочий
- Рабочий в автомастерскую
- Рабочий зеленого хозяйства
- Рабочий кормосклада
- Рабочий на плотницкие работы
- Рабочий по комплексному обслуживанию и ремонту зданий
- Рабочий по комплексному обслуживанию и ремонту зданий (Можайский филиал)

```
ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ
+  v  v  x

шиномонтаж
шлифовщик 5 разряда
шлифовщик механического цеха
эколог
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер станционного телевизионного оборудования
электросварщик
энтомолог
юрисконсульт 2 категории
f2
Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-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
Программистр-разработчик информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 138957 руб
Программист / Senior Developer с опытом Python, зарплата 108079 руб
Программист 1C с опытом Python, зарплата 167458 руб
Программист C# с опытом Python, зарплата 159693 руб
Программист C++ с опытом Python, зарплата 172098 руб
Программист C++/C#/Java с опытом Python, зарплата 192027 руб
Программист/ Junior Developer с опытом Python, зарплата 192547 руб
Программист/ технический специалист с опытом Python, зарплата 160889 руб
Программистр-разработчик информационных систем с опытом Python, зарплата 116628 руб
cm timer 1 0.30886220932006836
(venv) andrey@andrey-Aspire-E5-575G:~/Документы/lab3$
```