

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

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

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе № 3-4  
«Функциональные возможности языка Python»

Выполнил:  
студент группы ИУ5-34Б  
Бромберг Е.А.

Проверил:  
преподаватель каф. ИУ5  
Нардид А.Н.

Москва, 2024 г.

## **Постановка задачи**

### **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.

## Текст программы

```
def field(items, *args):
    assert len(args) > 0
    for i in range(len(items)):
        if len(args) == 1:
            if items[i][args[0]] != None:
                yield "{}{}".format(items[i][args[0]])
        else:
            flag = True
            map = dict()
            for j in range(len(args)):
                if items[i][args[j]] != None:
                    flag = False
                    map[args[j]] = items[i][args[j]]
            if not flag:
                yield map

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
]
print("Task 1: field(goods, 'title')")
test = field(goods, 'title')
print(*test, sep=', ')
print()
print("Task 2: field(goods, 'title', 'price')")
test = field(goods, 'title', 'price')
print(*test, sep=', ')
```

## Экранные формы с примерами выполнения программы

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python field.py
Task 1: field(goods, 'title')
'Ковер', 'Диван для отдыха'

Task 2: field(goods, 'title', 'price')
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}

(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```

## 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

### Текст программы

```
import random

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

gen = gen_random(5, 1, 3)
for i in gen:
    print(i)
```

### Экранные формы с примерами выполнения программы

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python gen_random.py
1
2
3
1
2
```

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python gen_random.py
1
2
2
1
3
```

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python gen_random.py
3
1
3
3
3

(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```

### 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.

### Тексты программы

```
from gen_random import gen_random

# 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
        self.used_elements = set()
        self.data = list(items)
        self.index = 0
        if len(kwargs) != 0:
            self.ignore_case = kwargs['ignore_case']
        else:
            self.ignore_case = False

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            else:
                current = self.data[self.index]
                self.index = self.index + 1
                flag = True
                if type(current) == str and self.ignore_case == True:
                    for i in self.used_elements:
                        if type(i) == str and i.lower() == current.lower():
                            flag = False
                            break
                if current not in self.used_elements and flag:
                    self.used_elements.add(current)
                    return current

    def __iter__(self):
        return self

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
for i in Unique(data, ignore_case=True):
    print(i)
```

## Экранные формы с примерами выполнения программы

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python unique.py  
1  
2
```

```
data = gen_random(10, 1, 3)
```

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python unique.py  
2  
3  
1
```

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

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python unique.py  
a  
A  
b  
B
```

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'] + ignore_case=True
```

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python unique.py  
a  
b  
  
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```

## Задача 4 (файл sort.py)

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`. 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:

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


## Текст программы

```
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))
    print(result_with_lambda)
```

## Экранные формы с примерами выполнения программы

 Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```



## 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.

### Текст программы

```
# Здесь должна быть реализация декоратора
def print_result(func_to_decorate):
    def decorated_func():
        result = func_to_decorate()
        print(func_to_decorate.__name__)
        if type(result) == list:
            for i in result:
                print(i)
        elif type(result) == dict:
            for i in result.keys():
                print('{} = {}'.format(i, result[i]))
        else:
            print(result)
        return result
    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()
```

## Экранные формы с примерами выполнения программы

Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python print_result.py
!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```

## 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`).

## Текст программы

```
import time  
from contextlib import contextmanager  
  
class cm_timer_1:  
    def __init__(self):  
        self.start = 0  
  
    def __enter__(self):  
        self.start = time.time()  
        return self.start  
  
    def __exit__(self, exp_type, exp_value, traceback):  
        if exp_type is not None:  
            print(exp_type, exp_value, traceback)  
        else:  
            print('time: {}'.format(time.time() - self.start))  
  
@contextmanager  
def cm_timer_2():  
    start = time.time()  
    yield start  
    print('time: {}'.format(time.time() - start))  
  
with cm_timer_1():  
    time.sleep(5.5)
```


## Экранные формы с примерами выполнения программы

`cm_timer_1`

 Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python cm_timer.py  
time: 5.507068395614624
```

cm\_timer\_2

 Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python cm_timer.py  
time: 5.502106428146362  
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```

## 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 С# 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 С# с опытом Python, зарплата 137287 руб. Используйте `zip` для обработки пары специальность — зарплата.

## Текст программы

```
import json
import sys
from unique import Unique
from gen_random import gen_random
from cm_timer import cm_timer_1

path = './data_light.json'

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

with open(path, "r", encoding="utf-8") 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

def print_result(func_to_decorate):
    def decorated_func(arg):
        result = func_to_decorate(arg)
        print()
        if type(result) == list:
            for i in result:
                print(i)
        elif type(result) == dict:
            for i in result.keys():
```

```

        print('{} = {}'.format(i, result[i]))
    else:
        print(result)
    return result
return decorated_func

def field(items, *args):
    assert len(args) > 0
    for i in range(len(items)):
        if len(args) == 1:
            if items[i][args[0]] != None:
                yield "{}".format(items[i][args[0]])
        else:
            flag = True
            map = dict()
            for j in range(len(args)):
                if items[i][args[j]] != None:
                    flag = False
                    map[args[j]] = items[i][args[j]]
            if not flag:
                yield map

@print_result
def f1(arg):
    return sorted(list(Unique(field(arg, "job-name"), ignore_case=True)))

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

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

@print_result
def f4(arg):
    return [i + ", зарплата {} руб.".format(j) for i, j in zip(arg,
gen_random(len(arg), 100_000, 200_000))]

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

```

## Экранные формы с примерами выполнения программы

cmd Командная строка

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>python process_data.py
```

```
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
web-программист
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
Агент по недвижимости
Агент по недвижимости (стажер)
Агент по недвижимости / Риэлтор
```

...

```
электросварщик
энтомолог
юрисконсульт 2 категории
```

```
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик информационных систем
```

```
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С 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
```

```
Программист с опытом Python, зарплата 177167 руб.
Программист / Senior Developer с опытом Python, зарплата 135410 руб.
Программист 1С с опытом Python, зарплата 136888 руб.
Программист C# с опытом Python, зарплата 123820 руб.
Программист C++ с опытом Python, зарплата 124350 руб.
Программист C++/C#/Java с опытом Python, зарплата 138134 руб.
Программист/ Junior Developer с опытом Python, зарплата 151950 руб.
Программист/ технический специалист с опытом Python, зарплата 111568 руб.
Программистр-разработчик информационных систем с опытом Python, зарплата 180689 руб.
time: 13.21373987197876
```

```
(lab_python_fp) C:\Users\user\PycharmProjects\lab_python_fp>
```