

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

**Факультет «Радиотехнический»  
Кафедра ИУ5 «Системы обработки информации и управления»**

**Курс «Базовые компоненты интернет-технологий»**

**Отчёт по лабораторной работе №3  
«Функциональные возможности языка Python»**

**Выполнил:**

**студент группы РТ5-31Б  
Андреев Виктор**

**Подпись и дата:**

**Проверил:**

**преподаватель каф. ИУ5  
Гапанюк Ю.Е.**

**Подпись и дата:**

**Москва, 2021 г**

## Описание задания

### 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
```

## 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
```

## 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
```

#### 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.

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)
```

#### 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

```

## 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`).

## 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 NotImplemented

@print_result
def f2(arg):
    raise NotImplemented

@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))))
```

# Текст программы

## field.py:

```
def field(items, *args):
    assert len(args) > 0 # аргументов (ключей) должно быть больше нуля
    if len(args) == 1: # если введён один аргумент (ключ)
        for d in items: # перебор предметов
            note = d.get(args[0]) # получение значения по ключу
            if note is not None: # если значение существует
                yield note # возврат генератора
    else:
        for d in items: # перебор предметов
            dictionary = dict() # определение словаря
            for key in args: # перебор ключей
                note = d.get(key) # получение значения по ключу
                if note is not None: # если значение существует
                    dictionary[key] = note # запоминание значения в словарь
            if len(dictionary) != 0: # если длина полученного словаря не равна нулю
                yield dictionary # возврат генератора

if __name__ == '__main__':

    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': 'Шкаф', 'price': None, 'color': 'brown'},
        {'title': 'Кресло', 'price': 8000, 'color': None},
        {'title': None, 'price': 404, 'color': 'white'}
    ]

    arr1 = list()
    arr2 = list()

    for i in field(goods, 'title'): # вывод названий
        arr1.append(i)
    print(arr1)

    for i in field(goods, 'title', 'price', 'color'): # вывод всей информации
        arr2.append(i)
    print(arr2)
```

## get\_random.py:

```
import random # модуль для получения случайного значения

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end) # возврат генератора случайного значения

if __name__ == '__main__':

    data = gen_random(5, 1, 3) # массив из генераторов случайных значений

    print(list(data)) # вывод случайных значений
```



## unique.py:

```
from get_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.used_elements = set() # множество для сохранения использованных элементов
        self.data = items # изначальный массив данных
        self.ignore_case = False # значение ignore_case (по умолчанию False)
        if len(kwargs) > 0:
            self.ignore_case = kwargs['ignore_case'] # изменение значения ignore_case

    def __next__(self):
        it = iter(self.data) # итератор массива данных
        while True:
            try:
                current = next(it) # получение следующего значения
            except StopIteration: # завершение цикла при возникновении исключения
                raise StopIteration
            else:
                if self.ignore_case is True and isinstance(current, str): # проверка на
ignore_case
                    current = current.lower()
                if current not in self.used_elements: # проверка на наличие элемента во
множестве элементов
                    self.used_elements.add(current) # добавление элемента во множество
                    return current # возврат текущего элемента

    def __iter__(self):
        return self

if __name__ == '__main__':
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data2 = gen_random(10, 1, 3)
    data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print(list(Unique(data1)))
    print(list(Unique(data2)))
    print(list(Unique(data3)))
    print(list(Unique(data3, ignore_case=True)))
```

## sort.py:

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

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True) # сортировка по абсолютному значению
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True) # ... с
использованием lambda-функции
    print(result_with_lambda)
```

## print\_result.py:

```
def print_result(func_to_decorate):
    def decorated_func(*args, **kwargs):
        incoming = func_to_decorate(*args, **kwargs) # получение значения функции
        print(func_to_decorate.__name__) # вывод названия функции
        if isinstance(incoming, list): # если значение - список
            for i in incoming:
                print(i)
        elif isinstance(incoming, dict): # если значение - словарь
            for i in incoming:
                print(str(i) + " = " + str(incoming[i]))
        else: # иное значение
            print(incoming)
        return incoming

    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__':
    test_1()
    test_2()
    test_3()
    test_4()
```

## cm\_timer.py:

```
import time
from contextlib import contextmanager

class cm_timer_1:

    def __init__(self):
        self.startTime = time.time() # запоминание времени начала исполнения

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

    def __exit__(self, exp_type, exp_value, traceback):
        print("time: {}".format(time.time() - self.startTime)) # вывод времени исполнения

@contextmanager
def cm_timer_2():
    try:
        startTime = time.time() # запоминание времени начала исполнения
        yield startTime
```

```

    finally:
        print("time: {}".format(time.time() - startTime)) # вывод времени исполнения

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(5.5)
    with cm_timer_2():
        time.sleep(5.5)

```

## process\_data.py:

```

import json
import sys
from print_result import print_result
from cm_timer import cm_timer_1
from unique import Unique
from field import field
from get_random import gen_random

# Сделаем другие необходимые импорты

path = r'/Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/data_light.json'

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

with open(path, 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

@print_result
def f1(arg): # функция для возврата массива, отсортированного по названию работы
    return sorted(list(Unique(field(arg, "job-name"), ignore_case=True)), key=str.lower)

@print_result
def f2(arg): # функция для возврата массива с работами, начинающихся с "программист"
    return list(filter(lambda string: str.startswith(str.lower(string), "программист"), arg))

@print_result
def f3(arg): # функция для модификации элементов массива
    return list(map(lambda s: s + " с опытом Python", arg))

@print_result
def f4(arg): # функция для добавления зарплаты
    return dict(zip(arg, list("зарплата {}".format(val) for val in gen_random(len(arg),
100000, 200000))))

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

```

# Примеры работы программы

```
def field(items, *args):
    assert len(args) > 0 # аргументов (ключей) должно быть больше нуля
    if len(args) == 1: # если введён один аргумент (ключ)
        for d in items: # перебор предметов
            note = d.get(args[0]) # получение значения по ключу
            if note is not None: # если значение существует
                yield note # возврат генератора
    else:
        for d in items: # перебор предметов
            dictionary = dict() # определение словаря
            for key in args: # перебор ключей
                note = d.get(key) # получение значения по ключу
                if note is not None: # если значение существует
                    dictionary[key] = note # запоминание значения в словарь
            if len(dictionary) != 0: # если длина полученного словаря не равна нулю
                yield dictionary # возврат генератора

if __name__ == '__main__':
    goods = [
        {'title': 'Ковёр', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': 'Шкаф', 'price': None, 'color': 'brown'},
        {'title': 'Кресло', 'price': 8000, 'color': None},
        {'title': None, 'price': 404, 'color': 'white'}
    ]

    arr1 = list()
    arr2 = list()

    for i in field(goods, 'title'): # вывод названий
        arr1.append(i)
        print(arr1)

    for i in field(goods, 'title', 'price', 'color'): # вывод всей информации
        arr2.append(i)
        print(arr2)
```

Ln: 1 Col: 0

```
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/field.py
['Ковёр', 'Диван для отдыха', 'Шкаф', 'Кресло']
[{'title': 'Ковёр', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}, {'title': 'Шкаф', 'color': 'brown'}, {'title': 'Кресло', 'price': 8000}, {'price': 404, 'color': 'white'}]
>>> |
```

Ln: 8 Col: 4

```
import random # модуль для получения случайного значения

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end) # возврат генератора случайного значения

if __name__ == '__main__':
    data = gen_random(5, 1, 3) # массив из генераторов случайных значений
    print(list(data)) # вывод случайных значений
```

Ln: 11 Col: 48

```
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/get_random.py
[1, 1, 3, 2, 1]
>>> |
```

Ln: 7 Col: 4

```

from get_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.used_elements = set() # множество для сохранения использованных элементов
        self.data = items # изначальный массив данных
        self.ignore_case = False # значение ignore_case (по умолчанию False)
        if len(kwargs) > 0:
            self.ignore_case = kwargs['ignore_case'] # изменение значения ignore_case

    def __next__(self):
        it = iter(self.data) # итератор массива данных
        while True:
            try:
                current = next(it) # получение следующего значения
            except StopIteration: # завершение цикла при возникновении исключения
                raise StopIteration
            else:
                if self.ignore_case is True and isinstance(current, str): # проверка на ignore_case
                    current = current.lower()
                if current not in self.used_elements: # проверка на наличие элемента во множестве
                    self.used_elements.add(current) # добавление элемента во множество
                    return current # возврат текущего элемента

    def __iter__(self):
        return self

if __name__ == '__main__':
    data1 = [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data2 = gen_random(10, 1, 3)
    data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print(list(Unique(data1)))
    print(list(Unique(data2)))
    print(list(Unique(data3)))
    print(list(Unique(data3, ignore_case=True)))

```

Ln: 16 Col: 67

```

Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/unique.py
[1, 2]
[1, 3, 2]
['a', 'A', 'b', 'B']
['a', 'b']
>>>

```

Ln: 10 Col: 4

```

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

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True) # сортировка по абсолютному значению
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True) # ... с использованием лямбда-функции
    print(result_with_lambda)

```

Ln: 8 Col: 29

```

Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
>>>

```

Ln: 8 Col: 4

```

def print_result(func_to_decorate):
    def decorated_func(*args, **kwargs):
        incoming = func_to_decorate(*args, **kwargs) # получение значения функции
        print(func_to_decorate.__name__) # вывод названия функции
        if isinstance(incoming, list): # если значение - список
            for i in incoming:
                print(i)
        elif isinstance(incoming, dict): # если значение - словарь
            for i in incoming:
                print(str(i) + " = " + str(incoming[i]))
        else: # иное значение
            print(incoming)
        return incoming

    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__':
    test_1()
    test_2()
    test_3()
    test_4()

```

Ln: 1 Col: 0

```

Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
>>>

```

Ln: 16 Col: 4

```

import time
from contextlib import contextmanager

class cm_timer_1:

    def __init__(self):
        self.startTime = time.time() # запоминание времени начала исполнения

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

    def __exit__(self, exp_type, exp_value, traceback):
        print("time: {}".format(time.time() - self.startTime)) # вывод времени исполнения

@contextmanager
def cm_timer_2():
    try:
        startTime = time.time() # запоминание времени начала исполнения
        yield startTime
    finally:
        print("time: {}".format(time.time() - startTime)) # вывод времени исполнения

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(5.5)
    with cm_timer_2():
        time.sleep(5.5)

```

Ln: 20 Col: 12

```

Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/cm_timer.py
time: 5.504944801330566
time: 5.50499415397644
>>>

```

Ln: 8 Col: 4

```

import json
import sys
from print_result import print_result
from cm_timer import cm_timer_1
from unique import Unique
from field import field
from get_random import gen_random

# Сделаем другие необходимые импорты

path = r'/Users/viktorandreev/Desktop/Programming/BKIT/lab3_code/data_light.json'

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

with open(path, 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

@print_result
def f1(arg): # функция для возврата массива, отсортированного по названию работы
    return sorted(list(Unique(field(arg, "job-name"), ignore_case=True)), key=str.lower)

@print_result
def f2(arg): # функция для возврата массива с работами, начинающихся с "программист"
    return list(filter(lambda string: str.startswith(str.lower(string), "программист"), arg))

@print_result
def f3(arg): # функция для модификации элементов массива
    return list(map(lambda s: s + " с опытом Python", arg))

@print_result
def f4(arg): # функция для добавления зарплаты
    return dict(zip(arg, list("{} руб.".format(val) for val in gen_random(len(arg), 10000

```

Ln: 1 Col: 0

```

энтомолог
юристконсульт
юристконсульт 2 категории
юристконсульт. контрактный управляющий
юрист
юрист (специалист по сопровождению международных договоров, английский - разгово
рный)
юрист волонтер
юристконсульт
f2
программист
программист / senior developer
программист 1с
программист с#
программист с++
программист с++/c#/java
программист/ junior developer
программист/ технический специалист
программист-разработчик информационных систем
f3
программист с опытом Python
программист / senior developer с опытом Python
программист 1с с опытом Python
программист с# с опытом Python
программист с++ с опытом Python
программист с++/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 122064 руб.
программист / senior developer с опытом Python = зарплата 196216 руб.
программист 1с с опытом Python = зарплата 107854 руб.
программист с# с опытом Python = зарплата 149969 руб.
программист с++ с опытом Python = зарплата 159929 руб.
программист с++/c#/java с опытом Python = зарплата 175704 руб.
программист/ junior developer с опытом Python = зарплата 110498 руб.
программист/ технический специалист с опытом Python = зарплата 158443 руб.
программист-разработчик информационных систем с опытом Python = зарплата 1861
02 руб.
time: 15.324515104293823
>>>

```

Ln: 2304 Col: 4