

**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 лабораторной работе №3–4
« Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-35Б
Коньгина Дарья
Подпись и дата:

Проверил:
Преподаватель каф. ИУ5
Нардид А.Н.
Подпись и дата:

Москва, 2022 г.

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
    if len(args) >= 1:
        print("---field---")
    d = dict()
    l = []
    for i in range(len(items)):
        if len(args)==1:
            if items[i].get(args[0]) != None:
                l.append(items[i].get(args[0]))
        else:
            c = 0
            for j in range(len(args)):
```

```

        if items[i].get(args[j]) != None:
            c+=1
        if c>0:
            l.append(items[i])

    return l

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'price': 5300, 'color': 'black'},
    {'vov': 58, 'sfs': 'efa'}
]

def main1():

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

if __name__ == '__main__':
    main1()

```

Экранные формы с примерами выполнения программы для задачи 1:

```

python
"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\Л3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/Л3/main.py"

---field---
['Ковер', 'Диван для отдыха']
---field---
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}, {'price': 5300, 'color': 'black'}]

Process finished with exit code 0

```

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):
    print("---gen_random---")
    l = [random.randint(begin, end) for _ in range(num_count)]
    return l

def main2():

    print(gen_random(5, 3, 10))

if __name__ == '__main__':
    main2()

```

Экранные формы с примерами выполнения программы для задачи 2:

```
python
"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\Л3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/Л3/main.py"

--gen_random--
[4, 5, 10, 6, 4]

Process finished with exit code 0
```

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(10, 1, 3)`

`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.

Текст программы:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # Например: ignore_case = True, Абв и АБВ - разные строки
        # По-умолчанию ignore_case = False -> Абв и АБВ - одинаковые строки
        #print('len(kwargs) =', len(kwargs))
        len1 = len(kwargs)
        global Ff
        if len1==0:
            Ff = False
        else:
            for key in kwargs:
                Ff = kwargs[key]
        self.used_elements = set()
        self.data = items
        self.index = 0

    def __iter__(self):
        # метод __iter__() - "превращать" итерируемый объект в итератор.
        # Если в цикл for передается уже итератор, то метод __iter__() этого
        # объекта должен возвращать сам объект
```

```

        return self

def __next__(self): #выдача очередного элемента
    while True:
        if self.index >= len(self.data):
            raise StopIteration
        else:
            current = self.data[self.index]
            self.index = self.index + 1
            #print("!!!!!!!!!!!!!!!!!!!!")
            if current not in self.used_elements:
                if Ff==False:
                    #print("CHECK", current, "-", current.lower())
                    if type(current) is not int:
                        if current.lower() not in self.used_elements:
                            self.used_elements.add(current.lower())
                            return current
                    else:
                        if current not in self.used_elements:
                            self.used_elements.add(current)
                            return current
                else:
                    self.used_elements.add(current)
                    return current

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = ['a', 'A', 'b', 'B', 'A', 'Abc', 'AbC']

def main3():
    for i in Unique(data1):
        print(i)
    print('//-----//')
    for i in Unique(data2):
        print(i)
    print('//-----//')
    for i in Unique(data2, ignore_case=True):
        print(i)

if __name__ == '__main__':
    main3()

```

Экранные формы с примерами выполнения программы для задачи 3:

```

"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\Л3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/Л3/main.py"

1
2
//-----//
a
b
Abc
//-----//
a
A
b
B
Abc
AbC

Process finished with exit code 0

```

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]

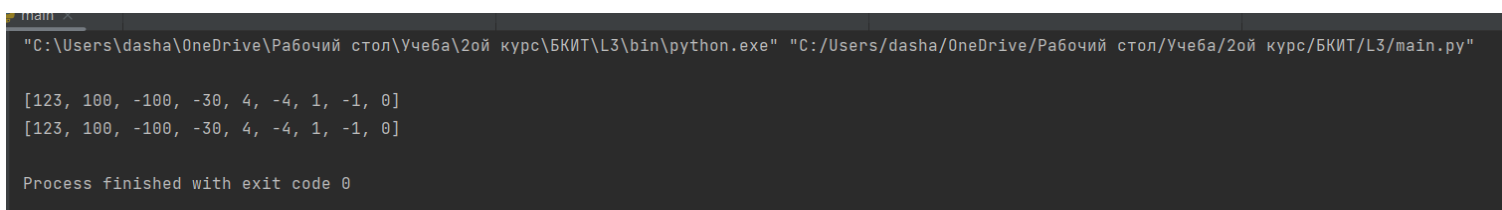
Необходимо решить задачу двумя способами:

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

Текст программы:

```
def result(data):  
    return sorted(data, key=abs, reverse=True)  
  
def result_with_lambda(data):  
    return sorted(data, key=lambda i: abs(i), reverse=True)  
  
data4 = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
def main4():  
  
    print(result(data4))  
    print(result_with_lambda(data4))  
  
if __name__ == '__main__':  
    main4()
```

Экранные формы с примерами выполнения программы для задачи 4:



```
main <-  
"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\Л3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/Л3/main.py"  
  
[123, 100, -100, -30, 4, -4, 1, -1, 0]  
[123, 100, -100, -30, 4, -4, 1, -1, 0]  
  
Process finished with exit code 0
```

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:

```
def print_result(func):
    #def wrapper(l=[], *args, **kwargs):
    def wrapper():
        print("FunctionName =", func.__name__)
        # res = func(l, *args, **kwargs)
        res = func()
        print("type =", type(res))
        if type(res) is list:
            print(*res, sep='\n')
        elif type(res) is dict:
            for key in res:
                print(key, "=", res[key])
        return res
    return wrapper

@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]

def main5():

    test_1()
    test_2()
    test_3()
    test_4()

if __name__ == '__main__':
    main5()
```

Экранные формы с примерами выполнения программы для задачи 5:

```
"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\Л3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/Л3/main.py"

FunctionName = test_1
type = <class 'int'>
FunctionName = test_2
type = <class 'str'>
FunctionName = test_3
type = <class 'dict'>
a = 1
b = 2
FunctionName = test_4
type = <class 'list'>
1
2

Process finished with exit code 0
```

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):
        pass
    def __enter__(self): # создание и возвращение объекта базы данных
        print("cm_timer_1 -> time_1")
        self.time1 = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb): # закрытие подключения
        print("cm_timer_1 -> time_2")
        self.time2 = time.time()
        t = self.time2-self.time1
        print(t)

# Yield — это ключевое слово в Python,
# которое используется для возврата из функции с сохранением состояния ее
# локальных переменных,
# и при повторном вызове такой функции выполнение продолжается с оператора
# yield,
# на котором ее работа была прервана.
@contextmanager
def cm_timer_2():
    print("cm_timer_2 -> time_1")
    t1 = time.time()
    yield
    print("cm_timer_2 -> time_2")
    t2 = time.time()
    t = t2-t1
    print(t)

def main6():
    with cm_timer_1():
        time.sleep(5.5)
    print('//-----//')
    with cm_timer_2():
        time.sleep(5.5)

if __name__ == '__main__':
    main6()
```

Экранные формы с примерами выполнения программы для задачи 6:


```
"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\Л3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/Л3/main.py"

cm_timer_1 -> time_1
cm_timer_1 -> time_2
5.511589288711548
//-----//
cm_timer_2 -> time_1
cm_timer_2 -> time_2
5.51489877708057

Process finished with exit code 0
```

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
import time
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.sort import result
from lab_python_fp.sort import result_with_lambda
from lab_python_fp.print_result import test_1, test_2, test_3, test_4, print_result
from lab_python_fp.cm_timer import cm_timer_1
```

```

from lab_python_fp.cm_timer import cm_timer_2

with open('C:\\Users\\dasha\\data_light.json', encoding="utf8") as f:
    data = json.load(f)

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

#@print_result
def f2(arg) -> list:
    #return list(filter(lambda s: s[0:len('программист')]=='Программист',
sorted(Unique(field(data, 'job-name'), ignore_case=True))))
    return list(filter(lambda s: s[0:len('программист')]=='Программист', arg))

#@print_result
def f3(arg) -> list:
    return list(map(lambda s: s+' с опытом работы Python', arg))

@print_result
def f4(arg)->list:
    return list(zip(arg, ['зарплата '+str(s)+' руб.' for s in gen_random(len(arg),
100000, 200000)]))

def main7():
    with cm_timer_1():
        f4(f3(f2(f1(data))))

if __name__ == '__main__':
    main7()

```

Экранные формы с примерами выполнения программы для задачи 7:

```

"C:\Users\dasha\OneDrive\Рабочий стол\Учеба\2ой курс\БКИТ\L3\bin\python.exe" "C:/Users/dasha/OneDrive/Рабочий стол/Учеба/2ой курс/БКИТ/L3/main.py"

cm_timer_1 -> time_1
type = <class 'list'>
('Программист с опытом работы Python', 'зарплата 199630 руб.')
('Программист / Senior Developer с опытом работы Python', 'зарплата 179743 руб.')
('Программист 1С с опытом работы Python', 'зарплата 174017 руб.')
('Программист C# с опытом работы Python', 'зарплата 151464 руб.')
('Программист C++ с опытом работы Python', 'зарплата 119560 руб.')
('Программист C++/C#/Java с опытом работы Python', 'зарплата 153028 руб.')
('Программист/ Junior Developer с опытом работы Python', 'зарплата 153880 руб.')
('Программист/ технический специалист с опытом работы Python', 'зарплата 165484 руб.')
('Программист-разработчик информационных систем с опытом работы Python', 'зарплата 127234 руб.')
cm_timer_1 -> time_2
0.00626373291015625

Process finished with exit code 0

```

