

**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 ИУ5-34Б:
Мамоу Асман
Подпись и дата:

проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е
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.

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(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.

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, 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.

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

```
goods = [
    {'title': 'Cover', 'price': 2000, 'color': 'green'},
    {'title': 'Divan dlya otidixa', 'price': 5300, 'color': 'black'}
]
def field(items, *args):
    res = []
    for i in range(0, len(items)):
        for j in range(0, len(args)):
            if items[i].get(args[j]) != None:
                res.append(items[i].get(args[j]))
    return res
print(field(goods, 'title', 'price'))
```

Gen_random.py

```
import random
def gen_random(num_count, begin, end):
    for i in range(0, num_count):
        yield (random.randrange(begin, end))
print(list(gen_random(5,1,10)))
```

Unique.py

```
import Gen_random
class Unique(object):
    def __init__(self, items, **kwargs):
        self.arr = []
        for i in items:
            if len(kwargs) > 0 and kwargs["ignore_case"]:
                if i.lower() not in self.arr:
                    self.arr.append(i.lower())
            else:
                if i not in self.arr:
                    self.arr.append(i)

    def __next__(self):
        item = self.arr[0]
        del self.arr[0]
```



```

        return item

    def __iter__(self):
        return self
if __name__ == '__main__':
    print('TEST1')
    D = [2,2,2,2,2,1,1,1]
    t = Unique(D)
    print(t.__next__())
    print(t.__next__())
    print('TEST2')
    D = ['A','a','B','b']
    t = Unique(D,ignore_case=True)
    print(t.__next__())
    print(t.__next__())
    print('TEST3')
    D = Gen_random.gen_random(10,1,3)
    t = Unique(D)
    print(t.__next__())
    print(t.__next__())

```

Sort.py

```

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),reverse=True)
    print(result_with_lambda)

```

Print_result.py

```

def print_result(func):
    def f(*args,**kwargs):
        print(func.__name__)
        output = func(*args,**kwargs)
        if type(output) == list:
            for i in output:
                print(i)
        elif type(output) == dict:
            for key in output.keys():
                print('{} = {}'.format(key,output[key]))
        else:
            print(output)
        return output
    return f

@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

```

from contextlib import contextmanager
import time as t

@contextmanager
def cm_timer_1():
    start = t.time()
    yield
    print("time: ", t.time() - start)

class cm_timer_2:
    def __init__(self):
        self.start = True
    def __enter__(self):
        self.start = t.time()
        return self.start
    def __exit__(self, exc_type, exc_val, exc_tb):
        print("time: ", t.time() - self.start)

if __name__ == "__main__":
    with cm_timer_1():
        t.sleep(5.5)
    with cm_timer_2():
        t.sleep(5.5)

```

Proccess_data.py

```

import json
from print_result import print_result
from cm_timer import cm_timer_1
from random import randrange
from unique import Unique
from Field import field
import sys

path='data_light.json'
with open(path, encoding = "utf8") as f:
    data = json.load(f)

@print_result
def fl(arg):
    return sorted(list(set([el['job-name'] for el in arg])), key=lambda x:
x.lower())

```

```

@print_result
def f2(arg):
    return [x for x in arg if x.split()[0].lower() == 'программист']

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

@print_result
def f4(arg):
    a = list(map(lambda x: x + ' с зарплатой ', arg))
    b = []
    for i in range(len(arg)):
        b.append(str(randrange(100000, 200000)))
    return list(zip(a, b))

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

```

Результаты кода

```

C:\Users\asman\PycharmProjects\LAB_3-4\venv\Scripts\python.exe C:/Users/asman/PycharmProjects/LAB_3-4/Field.py
['Cover', 2000, 'Divan dlya otdixa', 5300]

```

Process finished with exit code 0

```

C:\Users\asman\PycharmProjects\LAB_3-4\venv\Scripts\python.exe C:/Users/asman/PycharmProjects/LAB_3-4/Gen_random.py
[4, 9, 4, 6, 7]

```

Process finished with exit code 0

```

C:\Users\asman\PycharmProjects\LAB_3-4\venv\Scripts\python.exe C:/Users/asman/PycharmProjects/LAB_3-4/unique.py
[3, 2, 7, 4, 7]
TEST1
2
1
TEST2
a
b
TEST3
1
2

```

Process finished with exit code 0

```

C:\Users\asman\PycharmProjects\LAB_3-4\venv\Scripts\python.exe C:/Users/asman/PycharmProjects/LAB_3-4/sort.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

```
C:\Users\asman\PycharmProjects\LAB_3-4\venv\Scripts\python.exe C:/Users/asman/PycharmProjects/LAB_3-4/print_result.py
test_1
1
test_2
iv5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

```
C:\Users\asman\PycharmProjects\LAB_3-4\venv\Scripts\python.exe C:/Users/asman/PycharmProjects/LAB_3-4/cm_timer.py
time: 5.513701915740967
time: 5.506955623626709

Process finished with exit code 0
```

```
Программист C++/C#/Java с опытом Python
f4
('программист с опытом Python с зарплатой ', '190363')
('Программист с опытом Python с зарплатой ', '102989')
('Программист / Senior Developer с опытом Python с зарплатой ', '124101')
('Программист 1С с опытом Python с зарплатой ', '182142')
('программист 1С с опытом Python с зарплатой ', '104375')
('Программист C# с опытом Python с зарплатой ', '185376')
('Программист C++ с опытом Python с зарплатой ', '186212')
('Программист C++/C#/Java с опытом Python с зарплатой ', '143609')
time: 0.05137300491333008

Process finished with exit code 0
```