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

«Функциональные возможности языка Python.»

Выполнил: Тянутов Александр Дмитриевич

студент группы : ИУ5-31Б

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

.

Подпись и дата:

Москва, 2022 г.

Постановка задачи
Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. Решение каждой задачи должно раполагаться 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 количествово 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.
Пример:
gen_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

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. 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.

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 выводить 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

cm_timer.py

```
from time import perf_counter
from time import sleep
from contextlib import contextmanager
from unique import Unique

class cm_timer1:
    def __enter__(self):
        self.t = perf_counter()
    def __exit__(self, type, value, tranceback):
        ex_time = perf_counter() - self.t
        print('exec time = ', round(ex_time, 4))
        return ex_time

@contextmanager
def cm_timer2() -> float:
    start = perf_counter()
    try:
        yield perf_counter() - start
    finally:
        print('exec time = ', round(perf_counter() - start, 4))

if __name__ == '__main__':
    with cm_timer1():
        sleep(1)
    with cm_timer2():
        sleep(2)
```

field.py

```
from ast import arg


def field(items:list[dict], *args):
    """function for getting value by key

    Args:
        items (list[dict[str, Any]]):
    """
    assert len(args) > 0
    for item in items:
        if len(args) == 1 and args[0] in item.keys() and item[args[0]] is not None:
            yield item[args[0]]
        elif len(args) > 1:
```

```
        d = {x:item[x] for x in args if x in item.keys() and item[x] is not None}
        if bool(d):
            yield d


if __name__=='__main__':
    goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green', 'shape': None},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': None, 'price': None, 'color': None, 'shape': None},
    {'title': 'Стул', 'price': '2500', 'color': 'black', 'shape': 'round'}
    ]
    k = 'shape'
    m = list(field(goods, k))
    for g in field(goods, k):
        print(g)
```

gen_random.py

```
from ctypes import BigEndianStructure
import random


def gen_random(count, begin_num, end_num):
    """return a random generated number

    Args:
        count (int): amount of numbers
        begin_num (int): begin of random number
        end_num (int): end of random number

    Yields:
        int: random number
    """
    for i in range(0, count):
        # return random.randint(begin_num, end_num)
        yield random.randint(begin_num, end_num)

if __name__ == '__main__':
    gen_f = gen_random(5, 1, 3)
    for i in gen_f:
        print(i)
```

print_result.py

```
def print_result(func):
    def inner(*args, **kwargs):
        res = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(res, dict):
            for k, v in res.items():
                print(k, '=', v)
        elif isinstance(res, list):
            for i in res:
                print(i)
        else:
            print(res)
        return res
    return inner


@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()
```

process_data.py

```
from enum import unique
import json
```

```
import os
from logging import exception
from operator import contains
from cm_timer import cm_timer1
from gen_random import gen_random
from sort import sort_py
from unique import Unique
from unique import print_test
from print_result import print_result
from field import field


@print_result
def f1(arg):
    return sort_py(list(Unique(field(arg, 'job-name'), True)))

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


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


@print_result
def f4(arg):
    zip_str = zip(arg, gen_random(len(arg), 100000, 200000))
    str_ans = ['{}, зарплата {} руб.'.format(a, b) for a,b in zip_str]
    return str_ans

def main():
    path = os.path.dirname(__file__) + '\data_light.json'
    with open(path, encoding='utf-8') as f:
        data = json.load(f)
    return f4(f3(f2(f1(data))))

if __name__ == '__main__':
    with cm_timer1():
        main()
```

sort.py

```
from typing import Any
```

```
def sort_py_abs(items:list[int]):
    return sorted(items, key=abs,  reverse=True)

def sort_py(items:list[Any]):
    return sorted(items)

sort_py_lambda = lambda x: sorted(x, key=abs, reverse=True)

if __name__=='__main__':
    data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
    res = sort_py_abs(data)
    print(res)
    res_lambda = sort_py_lambda(data)
    print(res_lambda)
    data = ['d', 'a', 'c']
    res = sort_py(data)
```

unique.py

```
from gen_random import gen_random

class Unique(object):
    def __init__(self, items, ignore_case=False):
        self.c = 0
        self.ignore_case=ignore_case
        self.items = self.get_unique(items, ignore_case)


    def get_unique(self, items, ignore_case):
        un = set()
        lst=[]
        if items is None:
            return
        for x in items:
            if isinstance(x, str):
                if ignore_case:
                    x = x.lower()
                if x not in un:
                    un.add(x)
                    lst.append(x)
            elif not isinstance(x, str) and x not in un:
                un.add(x)
                lst.append(x)
```

```
        return lst

    def __next__(self):
        if self.c < len(self.items):
            x = self.items[self.c]
            self.c += 1
            return x
        else:
            raise StopIteration
    def __iter__(self):
        return self

def print_test(test):
    print('-'*50)
    for i in test:
        print(i)

import os

if __name__ == '__main__':
    data =[1,3,4,3, 'S', True]
    test = Unique(data)
    print_test(test)
    data = gen_random(10, 1, 3)
    test = Unique(data)
    print_test(test)
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    test = Unique(data, True)
    data = ['as', 'aS', 'Aa', 'AA', 'aA', 'aa', 5]
    test = Unique(data, True)
    print_test(test)
```

lab_3_test.py

```
import io
import contextlib
import json
import os
from operator import ge
from random import random
from unittest import TestCase, main
from field import field
from print_result import print_result
from unique import Unique
from gen_random import gen_random
from random import randint
from process_data import f4, f3, f2, f1

class fireld_test(TestCase):
    def __init__(self, methodName: str = ...) -> None:
        super().__init__(methodName)
        self.goods = [
            {'title': 'Ковер', 'price': 2000, 'color': 'green', 'shape': None},
            {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
            {'title': None, 'price': None, 'color': None, 'shape': None},
            {'title': 'Стул', 'price': '2500', 'color': 'black', 'shape': 'round'}
            ]
    def test_default_case(self):
        goods1 = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
        ]
        key_word = 'title'
        gen_field = iter(field(goods1, key_word))
        for i in goods1:
            self.assertEqual(next(gen_field), i[key_word])
    def test_none_case(self):
        key_words = ('title', 'shape')
        gen_field = iter(field(self.goods, *key_words))
        eq = {key_words[0]:self.goods[0][key_words[0]]}
        self.assertEqual(next(gen_field), eq)
        eq = {key_words[0]: self.goods[1][key_words[0]]}
        self.assertEqual(next(gen_field), eq)
        eq = {key_words[0]:self.goods[3][key_words[0]],
key_words[1]:self.goods[3][key_words[1]]}
        self.assertEqual(next(gen_field), eq)
    def test_none_size_case(self):
        key_words = ('title', 'shape')
```

```
        gen_field = list(field(self.goods, *key_words))
        self.assertEqual(len(gen_field), 3)
        key_words = 'shape'
        gen_field = list(field(self.goods, key_words))
        self.assertEqual(len(gen_field), 1)
    def test_not_existing_keys_case(self):
        key_words = 'kek'
        gen_field = list(field(self.goods, *key_words))
        self.assertEqual(len(gen_field), 0)
        key_words = ('kke', 'lel')
        gen_field = list(field(self.goods, *key_words))
        self.assertEqual(len(gen_field), 0)

@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]

class print_result_test(TestCase):
    def __init__(self, methodName: str = ...) -> None:
        super().__init__(methodName)

    def test_test_case(self):
        captured_out = io.StringIO()
        with contextlib.redirect_stdout(captured_out):
            a = test_1()
            b = test_2()
            c = test_3()
            d = test_4()
        captures_string = captured_out.getvalue()
        compare_string='test_1\n{}\ntest_2\n{}\ntest_3\n{}\ntest_4\n{}\n'.format('1', 'iu5',
'a = 1\nb = 2', '1\n2')
        self.assertEqual(captures_string, compare_string)
        self.assertEqual(a, 1)
```

```
        self.assertEqual(b, 'iu5')
        self.assertEqual(c, {'a': 1, 'b': 2})
        self.assertEqual(d, [1, 2])

class test_unique(TestCase):
    def __init__(self, methodName: str = ...) -> None:
        super().__init__(methodName)
    def test_list_int_case(self):
        data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
        ret = list(Unique(data))
        self.assertEqual(ret, [1, 2])
    def test_random_case(self):
        for i in range(100):
            data = list(gen_random(randint(1, 20), randint(1, 20), randint(20, 100)))
            ret = sorted(list(Unique(data)))
            self.assertEqual(ret, sorted(list(set(data))))
    def test_list_str_ignore_false_case(self):
        data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        ret = list(Unique(data))
        self.assertEqual(ret, ['a', 'A', 'b', 'B'])
    def test_list_str_ignore_true_case(self):
        data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        ret = list(Unique(data, True))
        self.assertEqual(ret, ['a', 'b'])
    def test_list_int_str_ignore_true_case(self):
        data = [1, 2, 3, 4, 3, 's', 'S']
        ret = list(Unique(data, True))
        self.assertEqual(ret, [1, 2, 3, 4, 's'])
    def test_list_int_str_ignore_false_case(self):
        data = [1, 2, 3, 4, 3, 's', 'S']
        ret = list(Unique(data, False))
        self.assertEqual(ret, [1, 2, 3, 4, 's', 'S'])
    def test_list_str_dc_ignore_false_case(self):
        data = ['as', 'aS', 'Aa', 'AA', 'aA', 'aa', 5, 'aA']
        ret = list(Unique(data))
        self.assertEqual(ret, ['as', 'aS', 'Aa', 'AA', 'aA', 'aa', 5])
    def test_list_str_dc_ignore_true_case(self):
        data = ['as', 'aS', 'Aa', 'AA', 'aA', 'aa', 5, 'aA']
        ret = list(Unique(data, True))
        self.assertEqual(ret, ['as', 'aa', 5])

class process_data_test(TestCase):
    def __init__(self, methodName: str = ...) -> None:
        super().__init__(methodName)
    def test_simple_output_check(self):
```

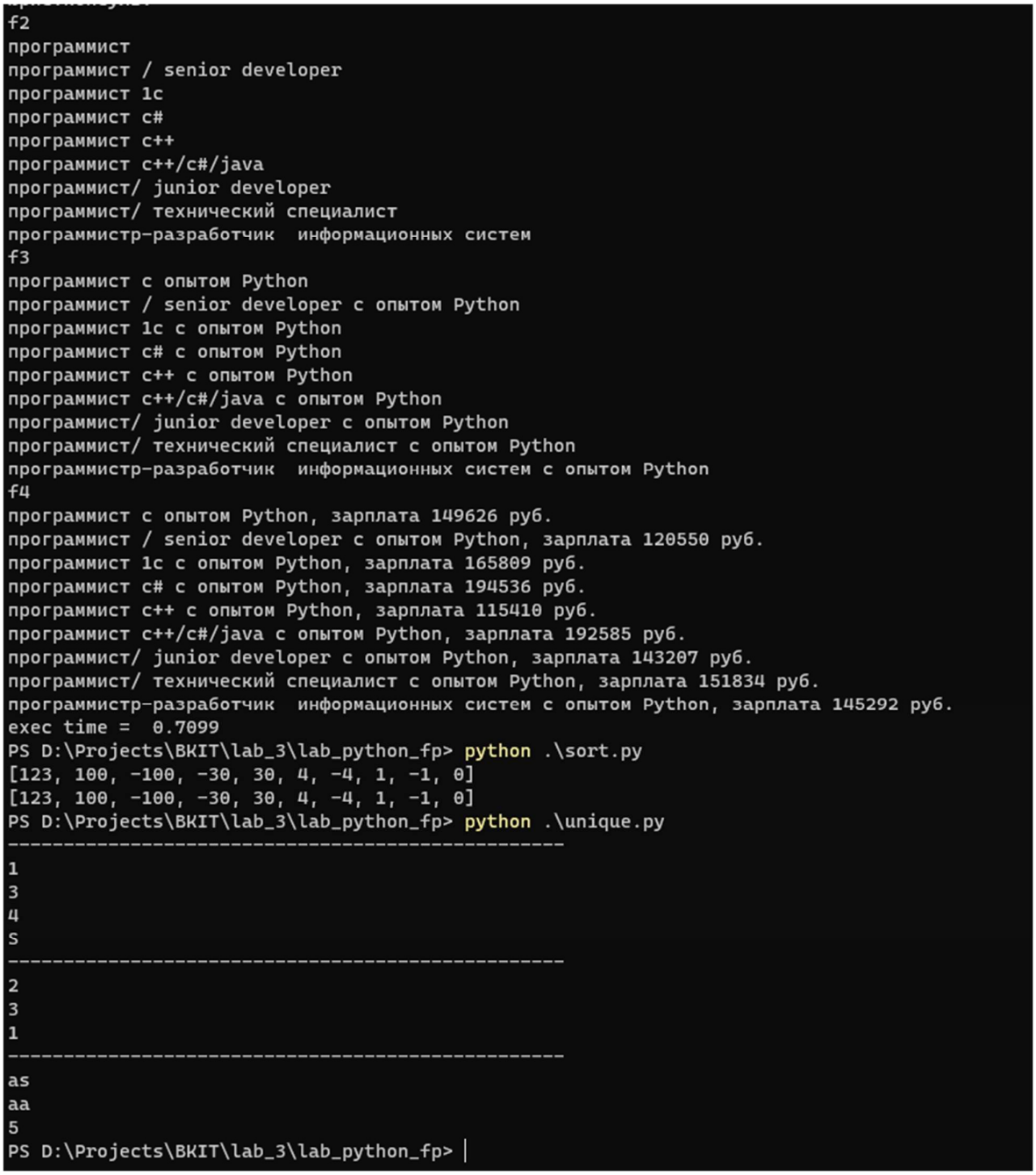
```
        path = os.path.dirname(__file__) + '\data_light.json'
        with open(path, encoding='utf-8') as f:
            data = json.load(f)
        captured_out = io.StringIO()
        with contextlib.redirect_stdout(captured_out):
            ret = f4(f3(f2(f1(data))))
        self.assertEqual(len(ret), 9)

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

Анализ результатов

```
PS D:\Projects\BKIT\lab_3\lab_python_fp> python .\cm_timer.py
exec time =  1.0068
exec time =  2.0065
PS D:\Projects\BKIT\lab_3\lab_python_fp> python .\field.py
round
PS D:\Projects\BKIT\lab_3\lab_python_fp> python .\gen_random.py
1
2
3
3
3
PS D:\Projects\BKIT\lab_3\lab_python_fp> python .\print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS D:\Projects\BKIT\lab_3\lab_python_fp> python .\process_data.py
f1
1c программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
```

Далее следует большой вывод process_data.py его сокращен до вывода функций f2 f3 f4



Вывод

Освоил функциональные возможности языка python. Научился создавать генераторы, декораторы и освоил работу с json в python