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

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

Отчет по лабораторной работе №3
«Основные конструкции языка Python»
по дисциплине
«Парадигмы и конструкции языков программирования»

Выполнил:
студент группы ИУ5-33Б:
Буйдина К.А.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2023 г.

Описание задания:

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.

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 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 выводить 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)

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:

1. **cm_timer**

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start = time.time()
        return self

    def __exit__(self, *args):
        self.end = time.time()
        self.interval = self.end - self.start
        print(f"Time: {self.interval} sec")
print("1st")
```

```
# Использование контекстного менеджера

'''with cm_timer_1():
   time.sleep(5.5)'''

print("2nd")

@contextmanager
def cm_timer_2():
    start_t = time.time()
    yield #приостанавливает свое выполнение, передавая управление обратно в
блок with
    end_t = time.time()
    print(f"Time: {end_t - start_t} sec")

'''with cm_timer_2():
   time.sleep(5.5)'''
```

2. field

```
# 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
    for d in items:
        for k in args:
            if d[k] != None:
                print(d[k])


goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Диван для кроликов', 'color': None, 'price': None}
]

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

3. gen_random

```
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
import random


def gen_random(num_count, begin, end):
    # Необходимо реализовать генератор
    a = []
    while num_count > 0:
        b = random.randint(begin,end)
        a.append(b)
        num_count -= 1
    return a
```

```
def gen_random1(num_count, begin, end):
    while num_count > 0:
        yield random.randint(begin, end) #сохраняет состояние функции до сл
вызова
        num_count -= 1

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

4. print_result

```
def print_result1(func):
    def wrapper():

        print(func.__name__)
        res = func()
        if isinstance(res, list):
            for i in res:
                print(i)
        elif isinstance(res, dict):
            for k in res:
                print(f'{k} = {res[k]}')
        else:
            print(res)
        return res
    return wrapper


def print_result(func):
    def wrapper(arg):

        res = func(arg)
        if isinstance(res, list):
            for i in res:
                print(i)
        elif isinstance(res, dict):
            for k in res:
                print(f'{k} = {res[k]}')
        else:
            print(res)

        return res
    return wrapper

@print_result1
def test_1():
    return 1


@print_result1
def test_2():
    return 'iu5'


@print_result1
def test_3():
    return {'a': 1, 'b': 2}
```

```
@print_result1
def test_4():
    return [1, 2]


'''if __name__ == '__main__':
    test_1()
    test_2()
    test_3()
    test_4()'''
```

5. process_data

```
import json
import sys
from print_result import print_result
import cm_timer
from unique import Unique
import re
import gen_random
# Сделаем другие необходимые импорты

path = 'data_light.json'
#path = 'test_3.json'

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

'''with open(path) as f:
    data = json.load(f)'''
with open(path, 'r', encoding='utf-8') as f:
    data = json.load(f)
    print(data[0])

# 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):
    return  [element for element in Unique([item["job-name"] for item in arg
            if "job-name" in item],ignore_case=True )]
@print_result
def f2(arg):
    return [x
            for x in arg
            if re.match("^программист",x) ]
@print_result
def f3(arg):
    return list(map(lambda x: x + " с опытом Python", arg))

@print_result
def f4(arg):
    return [f"{e}, зарплата {y} руб." for e,y in zip(arg,
gen_random.gen_random1(len(arg),100000,200000))]

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

6. sort

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

def sort_x(x): return abs(x)

if __name__ == '__main__':
    result = sorted(data,key=sort_x,reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)
```

7. lab_python_oop.color

```
# Итератор для удаления дубликатов
import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.items_iter = set()
        self.items = iter(items)
        self.ignore_case = kwargs.get('ignore_case', False)

    def __next__(self):
        while True:
            try:
                current = next(self.items)
            except StopIteration:
                raise StopIteration
            if self.ignore_case and isinstance(current, str):
                current = current.lower()
            if current not in self.items_iter:
                self.items_iter.add(current)
                return current

    def __iter__(self):
        return self


'''
data_list = [
    {0: [1, 1, 1, 1, 1, 2, 2, 2, 2, 2] },
    {0: ["a","A","b","B","a","A","b","B"]},
    {1: ["a","A","b","B","a","A","b","B"]},
    {0 : gen_random.gen_random1(10, 1, 3) }
]

for data_dict in data_list:
    for key in data_dict:
        data = data_dict[key]
        unique_iterator = Unique(data, ignore_case=key)
        print(f"Unique elements for key {key}:")
        for element in unique_iterator:
            print(element)'''
```

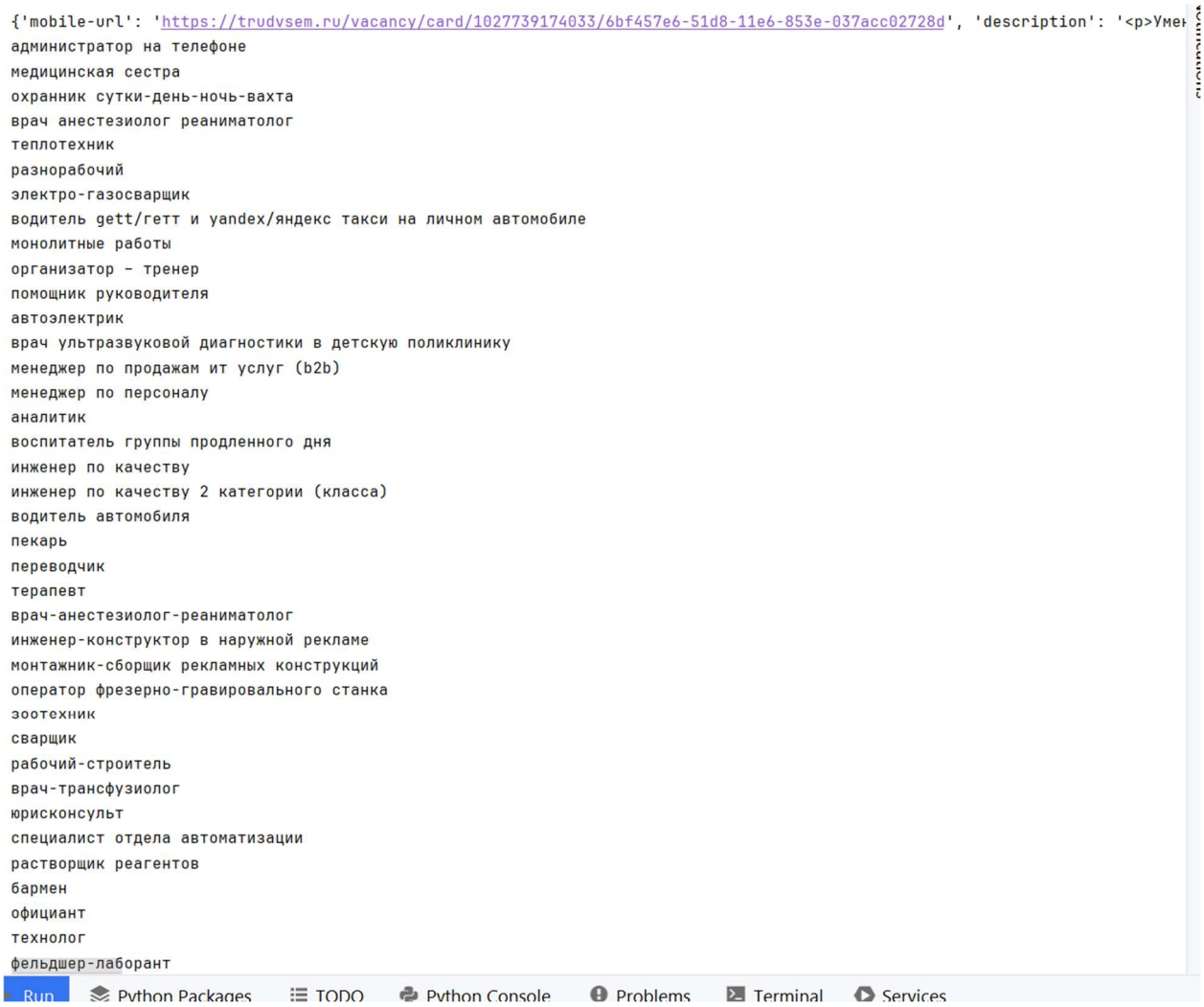
Экранные формы с примерами выполнения программы:

```
Ковер
Диван для отдыха
Диван для кроликов

Ковер
2000
Диван для отдыха
5300
Диван для кроликов

Process finished with exit code 0
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```



Так как в выводе много строк и они не могут быть запечатлены на скрине, была подготовлена тестовая выборка данных, которая демонстрирует работу кода:

```
1st
2nd
{'mobile-url': 'https://trudvsem.ru/vacancy/card/1157154009143/4c136616-0632-11e6-a218-4376a32b3f4
программист c++/c#/java
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
программист
программист c++/c#/java
программист
программист c++/c#/java с опытом Python
программист с опытом Python
программист c++/c#/java с опытом Python, зарплата 163684 руб.
программист с опытом Python, зарплата 132026 руб.
Time: 0.0 sec
```