МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Н. Э. БАУМАНА

ФАКУЛЬТЕТ
«ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА
«АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ»

Лабораторная работа №3

по учебной дисциплине
«Разработка интернет приложений»

на тему
«Функциональные

возможности языка Python»

Вариант №1

Группа: ИУ5Ц-73Б

Студент: Каунов А. А.

Преподаватель: Гапанюк Ю. Е.

Москва, 2021

1. Описание задания

Изучить возможности функционального программирования в языке Python:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. 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 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(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 = False, Абв и АБВ - разные строки
        #           ignore_case = True, 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)

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.

2. Текст программы

Файл main.py

```
from Menu.Menu import menu, run, pause

MenuMain = menu((
    "1. Задача 1. Реализация генератора field;",
    "2. Задача 2. Реализация генератора случайных чисел в диапазоне;",
    "3. Задача 3. Реализация итератора;",
    "4. Задача 4. Сортировка массива;",
    "5. Задача 5. Реализация декоратора функций;",
    "6. Задача 6. Контекстные менеджеры;",
    "7. Задача 7. Итоговая задача;",
    "0. Exit"
))

# Обеспечиваем "вечную работу", пока пользователь не захочет выйти
cont = True
while cont:

    # Печать пунктов меню
    MenuMain.print()
```

```
    # Осуществляем выполнение пунктов меню
    for case in MenuMain.input():
        if case(1): run("Tasks/field.py")
        if case(2): run("Tasks/gen_random.py")
        if case(3): run("Tasks/unique.py")
        if case(4): run("Tasks/sort.py")
        if case(5): run("Tasks/print_result.py")
        if case(6): run("Tasks/cm_timer.py")
        if case(7): run("Tasks/process_data.py")

        if case(0): cont=0
        # default
        if case():
            print('Вы хотите выйти?')
            print('Нажмите "0" !')

    # Окончание действия меню
    MenuMain.end()
```

Файл Menu.py

```
from Menu.switch import switch
from subprocess import call
from shutil import get_terminal_size

# Класс для обеспечения функционала Switch-Case
class menu(object):

    def __init__(self, options):
        self.options = options

    def input(self):
        while True:
            try:
                print()
                self.value = int(input('| Введите пункт меню: '))
                print()
                print()
                print("|***|***|***|")
                print()
            except BaseException:
                print('Вы не выбрали пункт меню!')
                print('Повторите ввод!')
                print()
                continue
            return switch(self.value)

    def print(self, options=None, indent=0):
        if not isinstance(options, tuple):
            options = self.options
        #deep - глубина списка. Нужна для регулирования вертикальных отступов
        deep = indent
        for item in options:
            if type(item) == tuple:
                deep = self.print(item, indent+1)
                if (deep >= indent):
                    deep=0
                    print()
            else:
                print("|"+"-"*indent+"| "+item)
        return deep
```

```
    def end(cont):
        print()
        print("|***|***|***|")
        print()
        pause()
        clear()
# ***

# Функция для обеспечения вызова другого файла
# Запускаем другой файл (в том же терминале)
def run(runfile):
    call(["python", runfile])


# Функция обеспечения паузы
def pause():
    programPause = input("Press the <ENTER> key to continue...")

def clear():
    print("\n" * get_terminal_size().lines, end='')
```

Файл field.py

```
goods = [
    {'title': 'Ковер',              'price': 2000,  'color': 'green'},
    {'title': 'Диван для отдыха',   'price': 5800,  'color': 'black'},
    {'title': 'Софа',               'price': 3000,  'color': 'blue'},
    {'title': 'Телевизор',          'price': 4000,  'color': 'black'},
    {'title': 'Атомная бомба',      'price': 6000,  'color': 'new sun'},
    {'title': 'Зачёт (бесценно)',                   'color': 'rainbow'},
]

def field(items, *args):
    assert len(args) > 0, 'Список аргументов не должен быть пустым'
    for i in items:
        arr = {}
        for j in args:
            if j in i:
                arr.update({j: i[j]})
        if len(args) == 1:
            if args[0] in arr:
                print(arr[args[0]])
        else:
            print(arr)

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

Файл random.py

```
import random

def gen_random(num_count, begin, end):
    i = 0
    arr = []
    while i < num_count:
        arr.append(random.randint(begin, end))
        i+=1
    return arr

if __name__ == '__main__':
    print("Рандомные числа:", gen_random(5,0,10))
```

Файл unique.py

```
#from Tasks.gen_random import gen_random

# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, ignore_case=False):
        self.data = items
        self.iteritems = iter(self.data)
        self.data = set(self.data)
        self.iteritems = iter(self.data)
        if ignore_case == True:
            try:
                self.data = map(lambda x:x.lower(),self.data)
                self.data = set(self.data)
                self.iteritems = iter(self.data)
            except BaseException:
                pass
    def __str__(self):
        return self.data

    def __iter__(self):
        return self.iteritems

data = ['a', 'A', 'a', 'a', 'A', 'B', 'b']
u = Unique(data, ignore_case=False)

if __name__ == '__main__':
    for i in u:
        print(i)
```

Файл sort.py

```
import math

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

print( sorted(data, key=lambda x: math.fabs(x), reverse=True) )

def funcSort(x):
    return math.fabs(x)

print( sorted(data, key=funcSort, reverse=True) )
```

Файл print_result.py

```
def print_result(result):
    def wrapper():
        print(result.__name__)
        res = result()
        if isinstance(res,list):
            for i in res:
                print(i)
        elif isinstance(res,dict):
            for i in res:
                print(i, ' = ', res[i])
        else: print(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]

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

Файл cm_timer.py

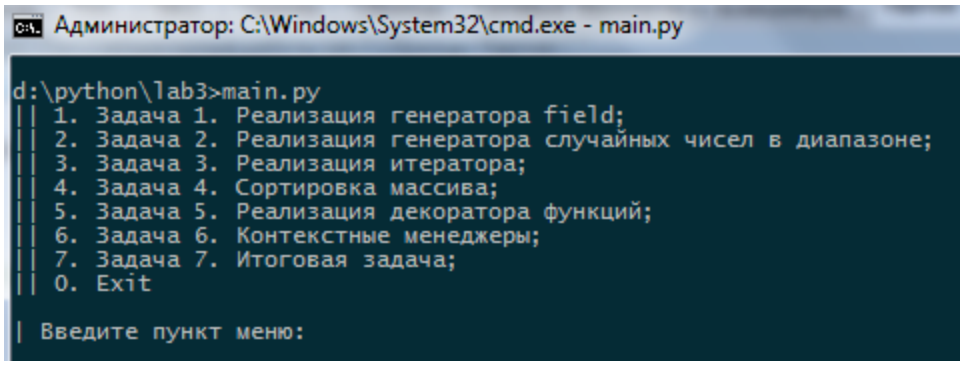
```
import time
from contextlib import contextmanager

class cm_timer_1():
    def __enter__(self):
        self.start = time.time()
    def __exit__(self, type, value, traceback):
        print(time.time() - self.start)

@contextmanager
def cm_timer_2():
    start = time.time()
    yield
    print(time.time() - start)

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(3)

    with cm_timer_2():
        time.sleep(2)
```

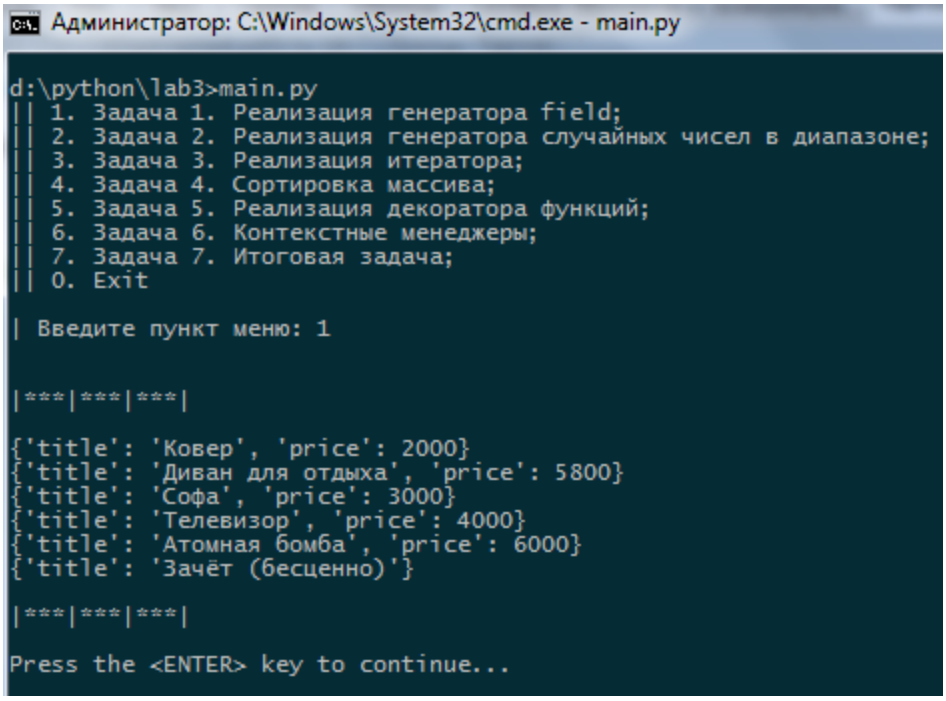
Файл process_data.py

```
import json
import sys
from Tasks.cm_timer import cm_timer_1
from Tasks.gen_random import gen_random
from Tasks.unique import Unique

def print_result(result):
    def wrapper(data):
        res = result(data)
        print(res)
        return res
    return wrapper

path = 'Tasks/datalight.json'

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

# @print_result
```

```
def f1(arg):
    arr = []
    for i in arg:
        arr.append(i['job-name'])
    res = Unique(arr, ignore_case=True)
    return res.data

# @print_result
def f2(arg):
    return list(filter(lambda x: x.lower().find('программист') != -1, arg))

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

@print_result
def f4(arg):
    return list(map(lambda x: x + ', зарплата {} руб.'.format(gen_random(1, 100000,
250000)[0]), arg))

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

3. Вывод программы

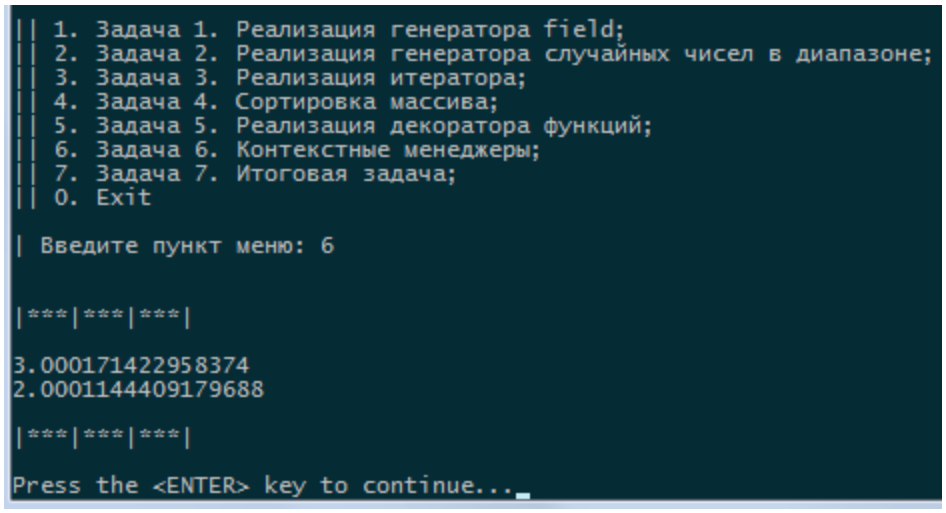




```
|| 1. Задача 1. Реализация генератора field;
|| 2. Задача 2. Реализация генератора случайных чисел в диапазоне;
|| 3. Задача 3. Реализация итератора;
|| 4. Задача 4. Сортировка массива;
|| 5. Задача 5. Реализация декоратора функций;
|| 6. Задача 6. Контекстные менеджеры;
|| 7. Задача 7. Итоговая задача;
|| 0. Exit

| Введите пункт меню: 2


|***|***|***|

Рандомные числа: [6, 5, 2, 6, 0]

|***|***|***|

Press the <ENTER> key to continue...
```

```
|| 1. Задача 1. Реализация генератора field;
|| 2. Задача 2. Реализация генератора случайных чисел в диапазоне;
|| 3. Задача 3. Реализация итератора;
|| 4. Задача 4. Сортировка массива;
|| 5. Задача 5. Реализация декоратора функций;
|| 6. Задача 6. Контекстные менеджеры;
|| 7. Задача 7. Итоговая задача;
|| 0. Exit

| Введите пункт меню: 3


|***|***|***|

B
b
a
A

|***|***|***|

Press the <ENTER> key to continue...
```

```
|| 1. Задача 1. Реализация генератора field;
|| 2. Задача 2. Реализация генератора случайных чисел в диапазоне;
|| 3. Задача 3. Реализация итератора;
|| 4. Задача 4. Сортировка массива;
|| 5. Задача 5. Реализация декоратора функций;
|| 6. Задача 6. Контекстные менеджеры;
|| 7. Задача 7. Итоговая задача;
|| 0. Exit

| Введите пункт меню: 5


|***|***|***|

test_1
1
test_2
iu5
test_3
a  =  1
b  =  2
test_4
1
2

|***|***|***|

Press the <ENTER> key to continue...
```


|| 1. Задача 1. Реализация генератора field;
|| 2. Задача 2. Реализация генератора случайных чисел в диапазоне;
|| 3. Задача 3. Реализация итератора;
|| 4. Задача 4. Сортировка массива;
|| 5. Задача 5. Реализация декоратора функций;
|| 6. Задача 6. Контекстные менеджеры;
|| 7. Задача 7. Итоговая задача;
|| 0. Exit
| Введите пункт меню: 6
|***|***|***|
3.000171422958374
2.0001144409179688
|***|***|***|
Press the <ENTER> key to continue...