Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Лабораторная работа №3-4 по БКИТ

“Функциональные возможности языка Python.**”**

Выполнил:
студент группы ИУ5-31Б
Ашуров Г. В.

Проверил:
Гапанюк Юрий Евгеньевич

2022 г.

Задание:

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.

Код:

```
goods = [
    {'title': 'Ковер', 'price': 1500, 'color': 'green', 'amount': 200},
    {'title': 'Диван для отдыха', 'price': 5000, 'color': 'yellow', 'amount': 50},
    {'title': 'Стол маленький', 'price': 2500, 'color': 'white', 'amount': 100},
    {'title': 'Ваза для цветов', 'price': 1000, 'color': 'blue', 'amount': 350},
]


def field(items, *args):
    try:
        assert len(args) > 0
        r = [{} for i in range(len(items))]
        for i in range(len(items)):
            for key in items[i]:
                if key in args:
                    r[i][key] = items[i][key]
        return r
    except:
        return "Not list of dicts as argument passed"
```

Результат:

```
PS C:\Users\Георгий\Desktop\myBKIT\BKIT\lab3-4> & C:/Users/Георгий/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:
/Users/Георгий/Desktop/myBKIT/BKIT/lab3-4/field.py
[{'title': 'Ковер', 'price': 1500}, {'title': 'Диван для отдыха', 'price': 5000}, {'title': 'Стол маленький', 'price': 2
500}, {'title': 'Ваза для цветов', 'price': 1000}]
```

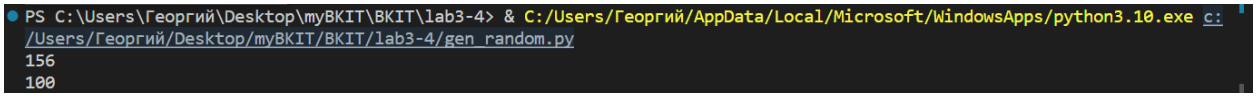
Задача 2 (файл gen_random.py)

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.
Код:

```
from random import randint

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield randint(begin, end)
```

Результат:



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.

Код:

```
class Unique(object):

    def __init__(self, items, **kwargs):
        self.r = []

        for key, value in kwargs.items():
```

```
            if key == 'ignore_case' and value == True:
                items = [i.lower() for i in items]

        for i in items:
            if i not in self.r:
                self.r.append(i)
        pass

    def __next__(self):
        try:
            x = self.r[self.begin]
            self.begin += 1
            return x
        except:
            raise StopIteration

    def __iter__(self):
        self.begin = 0
        return self

a = [1, 4, 87, 3, 5, 7, 2, 4, 6, 4, 3, 6, 3, 4, 2]
b = ['A', 'a', 'B', 'b']
for i in Unique(b):
    print(i)
```

Результат:

```
/Users/Георгий/Desktop/myBKIT/BKIT/lab3-4/unique.py
A
a
B
b
```

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.

Код:

```
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 a: -abs(a))
    print(result_with_lambda)
```

Результат:

```
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 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 выводить в столбик через знак равенства.

Код:

```
def print_result(f):
    def wrapper(a):
        print(f.__name__)
        res = f(a)
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for k,v in res.items():
                print(k, '=', v)
        elif type(res) == int or type(res) == str:
            print(res)
        elif type(res) == zip:
            for name, number in res:
                print(name, number)
        else:
            print("nothing to show")
        return res
```

```
    return wrapper

def print_result1(f):
    print(f.__name__)
    res = f()
    if type(res) == list:
        for i in res:
            print(i)
    elif type(res) == dict:
        for k,v in res.items():
            print(k, '=', v)
    elif type(res) == int or type(res) == str:
        print(res)
    else:
        print("nothing to show")
    return res


@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]
```

Результат:

```
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.

```
import time
from contextlib import contextmanager

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

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

with cm_timer_1() as c:
    time.sleep(2.6)

with cm_timer_2() as c:
    time.sleep(2.6)
```

Результат:

```
2.6042401790618896
2.6114015579223633
```

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.

Код:

```
import json
from unique import Unique
from field import field
from operator import concat
from gen_random import gen_random
from print_result import print_result
from cm_timer import cm_timer_1
```

```python
@print_result
def f1(a):
    return Unique([i['job-name'] for i in field(data, 'job-name')], ignore_case=True)

@print_result
def f2(a):
    return filter(lambda a: a.startswith('программист'), a)


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

@print_result
def f4(a):
    c = zip(a, gen_random(len(a), 100000, 200000))
    return c


with open('data_light.json',"r", encoding="utf-8") as f:
    data = json.loads(f.read())
    with cm_timer_1():
        (f4(f3(f2(f1(data)))))
```

Результат:

```
программист c++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист c# с опытом Python
f4
программист с опытом Python 167518
программист c++/c#/java с опытом Python 118406
программист 1с с опытом Python 101562
программистр-разработчик  информационных систем с опытом Python 111968
программист c++ с опытом Python 135838
программист/ junior developer с опытом Python 142235
программист / senior developer с опытом Python 175073
программист/ технический специалист с опытом Python 193376
программист c# с опытом Python 170321
```