Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Разработка интернет приложений»

Отчет по лабораторной работе №3
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-52Б
Васильченко Дарья

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2021 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 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.

- 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.

Текст программы

```
def field(items, *args):
    assert len(args) > 0
    result = []
    if len(args) > 1:
        i = -1
        for d in items:
            result.append({})
            i += 1
            for arg in args:
                val = d.get(arg)
                if val:
                    result[i].update({arg: val})
    else:
        for arg in args:
```

```
            for d in items:
                val = d.get(arg)
                if val:
                    result.append(val)
    return result


def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    print(*field(goods, 'title'), sep=", ")


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

Экранная форма с примером выполнения программы:

```
Ковер, Диван для отдыха
```

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.

Текст программы

```
import random


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


def main():
    for i in gen_random(5, 1, 3):
        print(i)


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

Экранная форма с примером выполнения программы:

```
3
3
2
1
3
```

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.

Текст программы

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.alphabet = set()
        if isinstance(items, list):
            self.items = iter(items)
        else:
            self.items = items
        b = kwargs.get('ignore_case')
        if b:
            self.ignore_case = bool(b)
        else:
            self.ignore_case = False

    def __next__(self):
        while True:
            nowElem = next(self.items)
            if self.ignore_case & isinstance(nowElem, str):
                nowElemLow = nowElem.lower()
                if nowElemLow not in self.alphabet:
                    self.alphabet.add(nowElemLow)
                    return nowElem
            else:
                if nowElem not in self.alphabet:
```

```
                    self.alphabet.add(nowElem)
                    return nowElem

    def __iter__(self):
        return self


def main():
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    iter1 = Unique(data, ignore_case=True)
    for i in iter1:
        print(i)


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

Экранная форма с примером выполнения программы:

```
a
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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы

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

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

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

Экранная форма с примером выполнения программы:

```
[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.

Текст программы

```
def print_result(func):
    def wrapper(*args, **kwargs):
        print(str(func).split(" ")[1])
        result = func(*args, **kwargs)
        if isinstance(result, list):
            for i in result:
                print(i)
        elif isinstance(result, dict):
            for key, val in result.items():
                print(str(key) + " = " + str(val))
        else:
            print(result)
        return result
    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
```

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


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

Экранная форма с примером выполнения программы:

```
. , .
!!!!!!!!
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.

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).

Текст программы

```python
from contextlib import contextmanager
import time


class cm_timer_1:
    def __init__(self):
        self.start_time = 0

    def __enter__(self):
        self.start_time = time.time()
```

```
    def __exit__(self, exc_type, exc_val, exc_tb):
        print("time: " + str(time.time() - self.start_time))


@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print("time: " + str(time.time() - start_time))


if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(1.5)

    with cm_timer_2():
        time.sleep(2.5)
```

Экранная форма с примером выполнения программы:

```
time: 1.5046021938323975
time: 2.5156619548797607
```

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

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

path = r"C:\My\3 курс\5 семестр\РИП\Лабы\lab3\data_light.json"

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


@print_result
def f1(arg):
    return sorted(Unique(field(arg, 'job-name'), ignore_case=True))


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


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


@print_result
def f4(arg):
    for job, sal in zip(range(len(arg)), gen_random(len(arg), 100_000,
200_000)):
        arg[job] += ", зарплата " + str(sal) + " руб."
    return arg
```

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

Экранная форма с примером выполнения программы:

```
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер станционного телевизионного оборудования
электросварщик
энтомолог
юрисконсульт 2 категории
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик  информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 186366 руб.
Программист / Senior Developer с опытом Python, зарплата 138691 руб.
Программист 1С с опытом Python, зарплата 136661 руб.
Программист C# с опытом Python, зарплата 152899 руб.
Программист C++ с опытом Python, зарплата 147308 руб.
Программист C++/C#/Java с опытом Python, зарплата 116878 руб.
Программист/ Junior Developer с опытом Python, зарплата 104047 руб.
Программист/ технический специалист с опытом Python, зарплата 165441 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 149569 руб.
time: 0.036927223205566406
```