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»
Отчёт по лабораторной работе №3
“Функциональные возможности языка Python”

Выполнил:
студент группы ИУ5-32Б
Милевич Артём
Подпись и дата:

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

Москва, 2021 г.

Общее 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.

- 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.

Текст программы

```
      def field(dicts, *keys):
    assert len(keys) > 0

    if len(keys) == 1:
        for dict in dicts:
            value = dict.get(keys[0])
            if value != 0: yield value
    else:
        for dict in dicts:
            res_dict = {}
            for key in keys:
                value = dict.get(key)
                if value != 0: res_dict[key] = value
            if len(res_dict) != 0: yield res_dict

if __name__ == '__main__':
    goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]

    result_list1, result_list2 = [], []

    result_list1 = [i for i in field(goods, 'title')]
```

```
    print(result_list1)

    result_list2 = [i for i in field(goods, 'title', 'price')]
    print(result_list2)
```

Результат выполнения программы

```
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
PS C:\Microsoft VS Code\python\lab_3\code>
```

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.

gen_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Текст программы

```
import random

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

if __name__ == '__main__':
    print (list(gen_random(5, 1, 3)))
```

Результат выполнения программы

```
[1, 2, 1, 3, 1]
PS C:\Microsoft VS Code\python\lab_3\code>
```

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

```
from .gen_random import gen_random

class Unique(object):
    def __init__(self, items, ignore_case = False, **kwargs):
        self.last = set()
        self.items = items
        self.registr = ignore_case
        self.kwargs = kwargs

    def __next__(self):
        it =  iter(self.items)
        while True:
            try:
                current = next(it)
            except StopIteration:
                raise
            else:
                if self.registr == True and isinstance(current, str):
                    temp = current[:]
                    if temp.lower() not in self.last:
                        self.last.add(temp.lower())
                        return current
                elif current not in self.last:
                    self.last.add(current)
                    return current

    def __iter__(self):
        return self

if __name__ == '__main__':
    list1 = ['a', "B", 'c', 'd', 'c',"A", 'b', "C", 'c', 'b']

    print(list(Unique(gen_random(50, 1, 4))))

    print(list(Unique(list1)))

    print(list(Unique(list1, ignore_case = True)))

    # Исходный список
    print(list1)
```

Результат выполнения программы

```
[1, 4, 2, 3]
['a', 'B', 'c', 'd', 'A', 'b', 'C']
['a', 'B', 'c', 'd']
['a', 'B', 'c', 'd', 'c', 'A', 'b', 'C', 'c', 'b']
PS C:\Microsoft VS Code\python\lab_3\code>
```

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

```
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  n: n if n>0 else -
n, reverse = True)
    print(result_with_lambda)
```

Результат выполнения программы

```
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
PS C:\Microsoft VS Code\python\lab_3\code>
```

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

```
 # Здесь должна быть реализация декоратора
def print_result(func):
    def decorated_func(*args, **kwargs):
```

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

@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 программы

```
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS C:\Microsoft VS Code\python\lab_3\code>
```

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).

Текст программы

```python
from contextlib import contextmanager
import time

class cm_timer_1():

    def __init__(self, before_mes, after_mes):
        self.before_mes = before_mes
        self.after_mes = after_mes
        self.start_time = None

    def __enter__(self):
        print('Основа - класс')
        print(self.before_mes)
        self.start_time = time.time()
        return 'Важная информация'

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print(f'Время работы менеджера: {time.time() - self.start_time} сек')
```

```
            print(self.after_mes)

@contextmanager
def cm_timer_2(before_mes, after_mes):
    print('Основа - библиотека contextlib')
    print(before_mes)
    start_time = time.time()
    yield 333
    print(f'Время работы менеджера: {time.time() - start_time} сек')
    print(after_mes)

if __name__ == '__main__':
    before_mes = 'Контекстный менеджер начал свою работу'
    after_mes = 'Контекстный менеджер закончил свою работу\n'

    with cm_timer_1(before_mes, after_mes):
        time.sleep(2)
    with cm_timer_2(before_mes, after_mes):
        time.sleep(3)
```

Результат выполнения программы

```
Основа - класс
Контекстный менеджер начал свою работу
Время работы менеджера: 2.014329195022583 сек
Контекстный менеджер закончил свою работу

Основа - библиотека contextlib
Контекстный менеджер начал свою работу
Время работы менеджера: 3.003297805786133 сек
Контекстный менеджер закончил свою работу

PS C:\Microsoft VS Code\python\lab_3\code>
```

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
import sys
from lab_python_fp import cm_timer_1, Unique, field, print_result, gen_random

try:
    path = 'C:\Microsoft VS Code\python\lab_3\code\data_light.json'
except IndexError:
    raise ValueError("Не указан путь к файлу.")
else:
    with open(path, encoding='utf-8') as f:
        data = json.load(f)

@print_result
def f1(lst):
    return sorted(list(Unique(field(lst, 'job-
name'), ignore_case=True)), key=str.lower)

@print_result
def f2(lst):
    return list(filter(lambda s: str.startswith(str.lower(s), 'программист'), lst))

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

@print_result
def f4(lst):
    return dict(zip(lst, list('зарплата {} руб.'.format(num) for num in gen_random(
len(lst), 1000000, 2000000))))

if __name__ == '__main__':
```

```python
    before_mes = 'Контекстный менеджер начал свою работу'
    after_mes = 'Контекстный менеджер закончил свою работу\n'
    with cm_timer_1(before_mes, after_mes):
        f4(f3(f2(f1(data))))
```

Результат выполнения программы

```
Энергетик
Энергетик литейного производства
энтомолог
Юрисконсульт
юрисконсульт 2 категории
Юрисконсульт. Контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский - разговорный)
Юрист волонтер
Юристконсульт
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 1610744 руб.
Программист / Senior Developer с опытом Python = зарплата 1402755 руб.
Программист 1С с опытом Python = зарплата 1142082 руб.
Программист C# с опытом Python = зарплата 1724650 руб.
Программист C++ с опытом Python = зарплата 1324731 руб.
Программист C++/C#/Java с опытом Python = зарплата 1589347 руб.
Программист/ Junior Developer с опытом Python = зарплата 1392018 руб.
Программист/ технический специалист с опытом Python = зарплата 1596699 руб.
Программистр-разработчик  информационных систем с опытом Python = зарплата 1546942 руб.
Время работы менеджера: 0.6187624931335449 сек
Контекстный менеджер закончил свою работу
```