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 3

Выполнил:
студент группы ИУ5-31Б
Шимолина Полина
Кирилловна

Подпись:_____

Дата:_____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

Подпись:_____

Дата:_____

Москва, 2021 г.

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.

field.py

```python
def field(items, *args):
    assert len(args)>0
    if len(args)==1:
        for i in items:
            try:
                if i[args[0]] is not None:
                    yield i[args[0]]
            except:
                pass
    else:
        for i in items:
            n={}
            for j in args:
                try:
                    if i[j] is not None:
                        n.update({j:i[j]})
                except:
                    pass
            if not len(n)==0:
                yield n

if __name__=='__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    print(','.join(field(goods, 'title')))
    for n in field(goods,'title','price'):
        print(n)
    pass
```

Результат:

```
Ковер,Диван для отдыха
{'title': 'Ковер', 'price': 2000}
{'title': '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.
Пример:
`gen_random(5, 1, 3)` должен выдать 5 случайных чисел в диапазоне от 1 до 3, например `2, 2, 3, 2, 1`

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

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

Результат:

```
3 1 2 1 2
```

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.

unique.py

```
from gen_random import gen_random


class Unique(object):

    def __init__(self, items, **kwargs):
        self.used_el = set()
        self.data = items
        assert len(kwargs) < 2
        if len(kwargs) == 0:
            self.ignore_case = False
        else:
            try:
                self.ignore_case = kwargs['ignore_case']
            except KeyError as k:
                print('Неверное имя параметра итератора: ожидалось
{}'.format(k))
                raise

    def __next__(self):
        iter = iter(self.data)
        while True:
            try:
                curr = next(iter)
                if self.ignore_case and isinstance(curr, str):
                    check = curr[:].lower()
                    if check not in self.used_el:
                        self.used_el.add(check)
                        return curr
                elif curr not in self.used_el:
                    self.used_el.add(curr)
                    return curr
            except StopIteration:
                raise

    def __iter__(self):
        return self


if __name__ == '__main__':
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print([i for i in Unique(data)])

    data = list(gen_random(10, 1, 3))
    print([i for i in Unique(data)])

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print([i for i in Unique(data)])
    print([i for i in Unique(data, ignore_case=True)])
```

Результат:

```
[1, 2]
[3, 2, 1]
['a', 'A', 'b', 'B']
['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. 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.

sort.py

```
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 d: abs(d), reverse = True)
    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.

print_result.py

```
def print_result(func):
    def d_f(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        #проверяем является ли результат листом
        if isinstance(result, list):
            for i in result:
                print(i)
        #проверяем является ли результат словарем
        elif isinstance(result, dict):
            for kw, arg in result.items():
                print('{} = {}'.format(kw, arg))
        else:
            print(result)
        return result
    return d_f
```

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. 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).

cm_timer.py

```
from time import time, sleep, perf_counter
from contextlib import contextmanager

#на основе класса
class cm_timer_1:
    def __init__(self):
        self.start = 0

    def __enter__(self):
        self.start = time()

    def __exit__(self, exp_type, exp_value, traceback):
        print('time: {}'.format(time() - self.start))

#с использованием библиотеки contextlib
@contextmanager
def cm_timer_2():
    start = time()
    yield ' '
    print('time: {}'.format(time() - start))

if __name__ == '__main__':
    with cm_timer_1():
        sleep(3)
    with cm_timer_2():
        sleep(3)
```

Результат:

```
time: 3.012655735015869
time: 3.0172817707061768
```

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.

process_data.py

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

path = r'C:\Users\Полина\Desktop\f o 1 d e r\учеба\2 курс\бкит\lab
3\data_light.json'

with open(path, encoding='utf-8') as f:
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(list(Unique(list(field(arg, "job-name")),
ignore_case=True)), key=lambda a: a.lower())


@print_result
def f2(arg):
```

```
    return list(filter(lambda j: j[:11].lower() == "программист", arg))


@print_result
def f3(arg):
    return list(map(lambda a: '{} с опытом Python'.format(a), list(arg)))


@print_result
def f4(arg):
    sal = [i for i in gen_random(len(arg), 100000, 200000)]
    return ['{}, зарплата: {} рублей'.format(job, sal) for job, sal in
zip(arg, sal)]


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

Результат:

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
```

```
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: 181542 рублей
Программист / Senior Developer с опытом Python, зарплата: 194077 рублей
Программист 1С с опытом Python, зарплата: 167267 рублей
Программист C# с опытом Python, зарплата: 130601 рублей
Программист C++ с опытом Python, зарплата: 162011 рублей
Программист C++/C#/Java с опытом Python, зарплата: 100470 рублей
Программист/ Junior Developer с опытом Python, зарплата: 128584 рублей
Программист/ технический специалист с опытом Python, зарплата: 114431 рублей
Программистр-разработчик  информационных систем с опытом Python, зарплата: 197239 рублей
time: 2.700223922729492
```