Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)»

Факультет «Информатика и системы управления»

Курс «Базовые компоненты интернет-технологий»

Лабораторная работа №3 «Функциональные возможности языка Python»

Выполнил:

студент группы ИУ5-33Б Николай Горкунов

подпись: , дата:

Проверил:

преподаватель Юрий Гапанюк

подпись: , дата:

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 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 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.

Исходный код:

```
def field(items, *args):
    size = len(args)
    assert size > 0
    ret = {}
    for item in items:
        skip = True
        for arg in args:
            if item.get(arg) is not None:
                ret[arg] = item[arg]
                skip = False
        if not skip:
            if size == 1:
                yield ret[args[0]]
            else:
                yield ret
        ret.clear()


def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color':
'black'},
        {'name': 'Skip me', 'addition': 'please'},
        {'title': 'Стол', 'color': 'brown'},
        {'title': 'Табуретка', 'serialNum': 1000, 'price':
int(1e9), 'color': '', 'name': '', 'addition': 'из будущего'}
    ]
    oneFieldGen = field(goods, 'title')
    print('=====  one field  =====')
    for i in oneFieldGen:
        print(i)
    twoFieldsGen = field(goods, 'title', 'price')
    print('===== two fields  =====')
```

```
    for i in twoFieldsGen:
        print(i)
    manyFieldsGen = field(goods, 'title', 'serialNum', 'name',
'addition')
    print('===== many fields =====')
    for i in manyFieldsGen:
        print(i)

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/field.py
=====  one field  =====
Ковер
Диван для отдыха
Стол
Табуретка
===== two fields  =====
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
{'title': 'Стол'}
{'title': 'Табуретка', 'price': 1000000000}
===== many fields =====
{'title': 'Ковер'}
{'title': 'Диван для отдыха'}
{'name': 'Skip me', 'addition': 'please'}
{'title': 'Стол'}
{'title': 'Табуретка', 'serialNum': 1000, 'name': '', 'addition': 'из будущего'}
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```

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.

Исходный код:

```
import random

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

def main():
    rndGen = gen_random(5, 1, 3)
    for i in rndGen:
        print(i, end = " ")
    print()

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/gen_random.py
2 1 3 1 3
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```

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.

Исходный код:

```
import random
import copy
from gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = False
        if 'ignore_case' in kwargs:
            self.ignore_case = kwargs['ignore_case']
        self.used_elements = set()
        self.iterator = iter(items)

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            try:
                current = next(self.iterator)
            except StopIteration:
                raise StopIteration
            cur_cpy = copy.copy(current)
            if self.ignore_case:
                cur_cpy = cur_cpy.casefold()
            if cur_cpy not in self.used_elements:
                self.used_elements.add(cur_cpy)
                return current

def main():
    # будет последовательно возвращать только 1 и 2
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

```
    for i in Unique(data):
        print(i, end = " ")
    print()
    # будет последовательно возвращать только 1, 2 и 3
    data = gen_random(10, 1, 3)
    for i in Unique(data):
        print(i, end = " ")
    print()
    # будет последовательно возвращать только a, A, b, B
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for i in Unique(data):
        print(i, end = " ")
    print()
    # будет последовательно возвращать только a, b
    for i in Unique(data, ignore_case = True):
        print(i, end = " ")
    print()

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/unique.py
1 2
2 1 3
a A b B
a b
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```

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.

Исходный код:

```
import functools

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

def main():
    result = [i[1] for i in sorted([(abs(j), j) for j in data],
reverse = True)]
    print(result)
    result_with_lambda = list(map(lambda x: x, sorted(data, key =
functools.cmp_to_key(lambda a, b: (abs(a) - abs(b))), reverse =
True)))
    print(result_with_lambda)

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/sort.py
[123, 100, -100, 30, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```

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.

Исходный код:

```
def print_result(some_func):
    def decorated(*args, **kwargs):
        ret = some_func(*args, **kwargs)
        print(some_func.__name__)
        if isinstance(ret, dict):
            for i in ret:
                print('{} = {}'.format(i, ret[i]))
        elif isinstance(ret, list):
            for i in ret:
                print(i)
        else:
            print(ret)
        return ret
    return decorated

@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]

def main():
```

```
    test_1()
    test_2()
    test_3()
    test_4()

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```

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).
Исходный код:

```
from contextlib import contextmanager
import time

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

    def __enter__(self):
        self.t = time.time()
        return

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print('time:', time.time() - self.t)

@contextmanager
def cm_timer_2():
    t = time.time()
    yield
    print('time:', time.time() - t)

def main():
    try:
        with cm_timer_1() as cm_object:
            cm_object
    except:
        pass

    with cm_timer_2() as cm_object:
        try:
            cm_object
        except:
```

```
            pass

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/cm_timer.py
time: 7.152557373046875e-07
time: 7.152557373046875e-07
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```

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.

Исходный код:

```
import json
import sys
from cm_timer import cm_timer_1
from print_result import print_result
import random

path = 'lab_python_fp/data.json'

with open(path) as f:
    data = json.load(f)

@print_result
def f1(arg):
    return list(map(lambda x: x[1], dict(map(lambda y:
(y.casefold(), y), list(map(lambda x: x.get('job-name'),
arg))))).items()))

@print_result
```

```python
def f2(arg):
    return list(filter(lambda x: x.casefold().find('программист')
== 0, arg))

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

@print_result
def f4(arg):
    return list(zip(arg, list(map(lambda w: 'зарплата {}
руб.'.format(random.randint(100000, 200000)), arg))))

def main():
    with cm_timer_1():
        f4(f3(f2(f1(data))))

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

Пример выполнения:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$ python3.8 lab_python_fp/process_data.py
f1
Администратор на телефоне
Медицинская сестра
Охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
Разнорабочий
Электро-газосварщик
Водитель Gett/Гетт и Yandex/Яндекс такси на личном автомобиле
Монолитные работы
Организатор - тренер
Помощник руководителя
Автоэлектрик
Врач ультразвуковой диагностики в детскую поликлинику
Менеджер по продажам ИТ услуг (B2B)
Менеджер по персоналу
Аналитик
Воспитатель группы продленного дня
Инженер по качеству
Инженер по качеству 2 категории (класса)
Водитель автомобиля
Пекарь
Переводчик
Терапевт
Врач-анестезиолог-реаниматолог
Инженер-конструктор в наружной рекламе
Монтажник-сборщик рекламных конструкций
Оператор фрезерно-гравировального станка
Зоотехник
```

… и так далее …

```
варщик зефира
варщик мармеладных изделий
Оператор склада
Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
Заведующий музеем в д.Копорье
Документовед
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
Менеджер (в промышленности)
f2
программист
Программист C++/C#/Java
программист 1С
Программистр-разработчик  информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
f3
программист с опытом Python
Программист C++/C#/Java с опытом Python
программист 1С с опытом Python
Программистр-разработчик  информационных систем с опытом Python
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', 'зарплата 168169 руб.')
('Программист C++/C#/Java с опытом Python', 'зарплата 101704 руб.')
('программист 1С с опытом Python', 'зарплата 113519 руб.')
('Программистр-разработчик  информационных систем с опытом Python', 'зарплата 157227 руб.')
('Программист C++ с опытом Python', 'зарплата 111683 руб.')
('Программист/ Junior Developer с опытом Python', 'зарплата 100658 руб.')
('Программист / Senior Developer с опытом Python', 'зарплата 117742 руб.')
('Программист/ технический специалист с опытом Python', 'зарплата 106151 руб.')
('Программист C# с опытом Python', 'зарплата 158442 руб.')
time: 0.02424168586730957
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022$
```