Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторным работам №3-4
« Функциональные возможности языка Python »

Выполнил:
студент группы ИУ5-32Б
Кузьмин А.Ю.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

дата: 16.12.2022

Москва, 2022 г.

Полученное 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.

Задание 1

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.

```python
from collections.abc import Generator


def field(lst, *args):
    # assert len(lst) > 0
    if len(args) == 0:
        return
    if len(args) == 1:
        key = args[0]
        for dct in lst:
            if key in args:
                yield dct[key]
    else:
        for dct in lst:
            res = {}
            for key in args:
                if key in dct:
```

```
                    res[key] = dct[key]
            yield res


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

    a = field(goods, 'title')
    print(isinstance(a, Generator))
    print(list(a))  # должен выдавать 'Ковер', 'Диван для отдыха'
    print(list(a))
    print("=============")

    b = field(goods, 'title', 'price')
    print(isinstance(a, Generator))
    print(list(b))
    # должен выдавать {'title': 'Ковер', 'price': 2000},
    # {'title': 'Диван для отдыха', 'price': 5300}
    print(list(b))
    print("=============")

    c = field(goods)
    print(isinstance(a, Generator))
    print(list(c))
# Пример выполнения кода
[Running] python -u "c:\BKIT\lab3\lab_python_fp\field.py"
True
['??????', '?????? ??? ???????']
[]
=============
True
[{'title': '??????', 'price': 2000}, {'title': '?????? ??? ???????',
'price': 5300}]
[]
=============
True
[]

[Done] exited with code=0 in 0.159 seconds
```

Задание 2

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`

```python
import random


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


if __name__ == "__main__":
    args = (5, 1, 3)
    print(f"Пример результата вызова функции gen_random{args}")
    print(*gen_random(*args))
# Пример выполнения кода
[Running] python -u "c:\BKIT\lab3\lab_python_fp\tempCodeRunnerFile.py"
??????? ??????????? ??????? ???????? gen_random(5, 1, 3)
1 1 1 3 2

[Done] exited with code=0 in 0.191 seconds
```

Задание 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.

Пример:

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

`Unique(data)` будет последовательно возвращать только 1 и 2.

```
data = gen_random(10, 1, 3)
```

`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.

```python
from gen_random import gen_random


class Unique:
    def __init__(self, items, **kwargs):
        self.it = iter(items)
        self.ignore_case = kwargs["ignore_case"] if "ignore_case" in kwargs\
            else False
        self.prev = None
        self.is_first = True

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            current = next(self.it)
            if self.is_first or current != self.prev and not self.ignore_case\
                or (self.ignore_case and isinstance(current, str)
                    and isinstance(self.prev, str)
                    and current.lower() != self.prev.lower()):
                break
        self.prev = current
```

```
        self.is_first = False
        return current


if __name__ == "__main__":
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(list(Unique(data)))

    data = gen_random(10, 1, 3)
    print(list(Unique(data)))

    data = ["a", "A", "b", "B", "a", "A", "b", "B"]
    print(list(Unique(data)))

    print(list(Unique(data, ignore_case=True)))
# Пример выполнения кода
[Running] python -u "c:\BKIT\lab3\lab_python_fp\tempCodeRunnerFile.py"
[1, 2]
[3, 1, 3, 2, 3, 2]
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
['a', 'b', 'a', 'b']

[Done] exited with code=0 in 0.159 seconds
```

Задание 4

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.

```
def sort_with_lambda(lst):
    return sorted(lst, key=lambda x: abs(x), reverse=True)


def sort_not_use_lambda(lst):
    return sorted(lst, key=abs, reverse=True)


if __name__ == "__main__":
    data = [1, -4, 7, -3, 61]
    print(f"Пример результата вызова функции sort_with_lambda{data}")
    print(sort_with_lambda(data))
```

```
    print(f"\nПример результата вызова функции sort_not_use_lambda{data}")
    print(sort_not_use_lambda(data))
    print(f"\nИсходный массив {data}")
    print(data)
# Пример выполнения кода
[Running] python -u "c:\BKIT\lab3\lab_python_fp\tempCodeRunnerFile.py"
******* ************ ******* ******** sort_with_lambda[1, -4, 7, -3, 61]
[61, 7, -4, -3, 1]

******* ************ ******* ******** sort_not_use_lambda[1, -4, 7, -3, 61]
[61, 7, -4, -3, 1]

********** ******* [1, -4, 7, -3, 61]
[1, -4, 7, -3, 61]

[Done] exited with code=0 in 0.147 seconds
```

Задание 5

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.

```python
def print_result(func):
    def wrap(*args, **kwargs):
        res = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(res, list):
            print(*res, sep="\n")
        elif isinstance(res, dict):
            for key, val in res.items():
                print(f"{key} = {val}")

        return res

    return wrap


@print_result
def return_dict():
    return {"one": 1, "two": 2}


@print_result
```

```python
def return_list():
    return [1, 2, 3, 4, 5]


if __name__ == "__main__":
    return_dict()
    print("===========")
    return_list()
# Пример выполнения кода
[Running] python -u "c:\BKIT\lab3\lab_python_fp\print_result.py"
return_dict
one = 1
two = 2
===========
return_list
1
2
3
4
5

[Done] exited with code=0 in 0.182 seconds
```

Задание 6

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).

```python
import time
from contextlib import contextmanager


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

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

    def __exit__(self, *args):
        print(time.time() - self.start)
```

```python
@contextmanager
def cm_timer_2():
    start = 0
    try:
        start = time.time()
        yield None
    finally:
        print(time.time() - start)


if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(2)
        print("exit cm_timer_1")
    print("==================")
    with cm_timer_2():
        time.sleep(1)
        print("exit cm_timer_2")
# Пример выполнения программы
[Running] python -u "c:\BKIT\lab3\lab_python_fp\cm_timer.py"
exit cm_timer_1
2.000730276107788
==================
exit cm_timer_2
1.0006248950958252

[Done] exited with code=0 in 3.204 seconds
```

Задание 7

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.

```python
from cProfile import label
from field import field
import json
import sys
from print_result import print_result
from cm_timer import cm_timer_1
from gen_random import gen_random
from unique import Unique
from sorts import sort_not_use_lambda
# Сделаем другие необходимые импорты

path = "../data_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

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

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
# В реализации функции f4 может быть до 3 строк

@print_result
```

```python
def f1(arg):
    return Unique(sorted(arg, key=lambda x: x.lower()), ignore_case=True)


@print_result
def f2(arg):
    return filter(lambda x: x[:11] == "программист", arg)


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


@print_result
def f4(arg):
    return map(lambda x: f"{x[0]}, зарплата {x[1]} руб.",
               zip(field(data, "job-name"),
                   gen_random(len(data), 100000, 200000)))



if __name__ == '__main__':
    with cm_timer_1():
        print(*f1(f2(f3(f4(data)))), sep = "\n")
```