Московский государственный технический

университет им. Н.Э. Баумана

Факультет Радиотехнический
Кафедра РТ5

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3–4

«Функциональные возможности языка Python»

Выполнил:

студент группы РТ5-31Б:

Бабасанова Н. С.

Руководитель:

преподаватель каф. ИУ5

Гапанюк Ю. Е.

Москва, 2023г.

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 раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача 1:

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.

Текст программы:

```
goods = [
  {'title': 'Ковер', 'price': 2000, 'color': 'green'},
  {'title': 'Диван для отдыха', 'color': 'black'}
]
def field(items,*args):
  assert len(args) > 0
  for item in items:
    if len(args)==1:
      yield get_data (item, args[0])
    else:
      yield {key:get_data(item, key) for key in args}


def get_data(item, key):
  return item.get(key)

print([x for x in field(goods, 'title')])
print([x for x in field(goods, 'title', 'price')])
```

Результаты вывода:

```
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': None}]
```

Задача 2:

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.

Текст программы:

```
import random

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

print([x for x in gen_random(5,1,3)])
```

Результаты вывода:

```
[3, 3, 3, 3, 1]
```

Задача 3:

- 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:

```
import random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.iter_items = iter(items) if isinstance(items, list) else items
        self.ignore_case = kwargs.get('ignore_case', False)
        self.duplicates = []

    def __next__(self):
        while True:
            try:
                cur = next(self.iter_items)

                if self.ignore_case:
                    cur_check = cur.lower()
                else:
                    cur_check = cur

                if cur_check not in self.duplicates:
                    self.duplicates.append(cur_check)
                    return cur_check
```

```
            except Exception:
                raise StopIteration

    def __iter__(self):
        return self


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


numbers = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
letters = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
data = gen_random(10, 1, 3)

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

Результаты вывода:

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

Задача 4:

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.

Текст программы:

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

if __name__ == '__main__':
    # Без лямбда-функции
    print(sorted(data, key = abs, reverse=True))
    # С лямбда-функцией
    print(sorted(data, key=lambda x: abs(x), reverse=True))
```

Результаты вывода:

```
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
```

Задача 5:

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:

```
import functools
def print_result_helper(func, res):
    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)


def print_result(func):
    @functools.wraps(func)
    def decorated_func(*args, **kwargs):
        res = func(*args, **kwargs)
        print_result_helper(func, 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__':
    test_1()
    test_2()
    test_3()
    test_4()
```

Результаты вывода:

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Задача 6:

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:

```
from time import sleep
import datetime
import contextlib


@contextlib.contextmanager
def cm_timer_1():
    t1 = datetime.datetime.now()
    yield
    t2 = datetime.datetime.now()
    res = t2 - t1
    res1 = str(res.seconds) + '.' + str(res.microseconds)
    print('Execution time {0}'.format(res1))


class timer:
    def __enter__(self):
        self.t1 = datetime.datetime.now()

    def __exit__(self, exp_type, exp_value, traceback):
        t2 = datetime.datetime.now()
        res = t2 - self.t1
        print('Execution time {0}'.format(res))


print('cm_timer_1 start\n...\n')
with cm_timer_1():
    sleep(5.5)
print('\ncm_timer_2 start\n...\n')
with timer():
    sleep(5.5)
```

Результаты вывода:

```
cm_timer_1 start
...

Execution time 5.501783

cm_timer_2 start
...

Execution time 0:00:05.501492
```

Задача 7:

- 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.cm_timer import timer
from lab_python_fp.print_result import print_result
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique as unique

path = sys.argv[1]
# print(path)

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

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

# print(data)

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`
# Важно!
# Функции с 1 по 3 дожны быть реализованы в одну строку
# В реализации функции 4 может быть до 3 строк
# При этом строки должны быть не длиннее 80 символов


#@print_result
def f1(arg):
    return sorted([st for st in unique(field(arg, 'job-name'), ignore_case=True)], key=lambda x:
x.upper())


# @print_result
def f2(arg):
    return list(filter(lambda x: x.upper().startswith('ПРОГРАММИСТ'), arg))


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


@print_result
def f4(arg):
    salary = gen_random(len(arg), 100000, 200000)
    return [i + ', зарплата ' + str(j) + ' руб.' for i, j in zip(arg, salary)]

with timer():
    f4(f3(f2(f1(data))))
```

Результаты вывода:

```
cm_timer_1 start
...

Execution time 5.505814

cm_timer_2 start
...

Execution time 0:00:05.503019
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': None}]
[3, 3, 3, 2, 2]
[1, 2]
['a', 'A', 'b', 'B']
[1, 3, 2]
['A', 'a', 'B', 'b']
программист с опытом Python, зарплата 117099 руб.
программист / senior developer с опытом Python, зарплата 159142 руб.
программист 1с с опытом Python, зарплата 154144 руб.
программист c# с опытом Python, зарплата 187068 руб.
программист c++ с опытом Python, зарплата 183118 руб.
программист c++/c#/java с опытом Python, зарплата 121792 руб.
программист/ junior developer с опытом Python, зарплата 160599 руб.
программист/ технический специалист с опытом Python, зарплата 175222 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 146178 руб.
Execution time 0:00:00.046701
```