Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Радиотехника»
Кафедра «Информатика и вычислительная техника»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3

«Функциональные возможности языка Python»

Выполнил:
студент группы РТ5-31Б:
Михеева В.А.
Подпись и дата:

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

Москва, 2023 г.

Постановка задачи:

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.:

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.

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.

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.

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.

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.

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).

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:

cm_timer.py

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time}")
```

```
@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"time: {execution_time}")
```

field.py

```
def field(items, *args):
    assert len(args) > 0
    for i in items:
        if(len(args) == 1):
            yield i.get(args[0])
        else:
            dict = {}
            for j in args:
                if(i.get(j)!=None): dict[j]=i.get(j)
            yield dict

def zapusk(data, str):
    a=[]
    for i in field(data, str):
        a.append(i)
    return a
def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    for i in field(goods, 'title', 'price'):
        print(i, end='\n')
```

gen_random.py

```
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки

from random import randint

def gen_random(num_count, begin, end):
    gener = (f', зарплата {randint(begin, end)} руб.' for i in
range(num_count))
    return gener
def main():
    gener=gen_random(5, 1, 3)
    for i in gener:
```

```
        print(i, end=' ')
```

print_result.py

```
def print_result(func):
    def wrapper(*args):
        print(func.__name__)
        a=func(*args)
        if(isinstance(a, list)):
            for i in a:
                if(isinstance(i, tuple)):
                    print(*i)
                else:
                    print(i)
        elif(isinstance(a, dict)):
            for i in a.keys():
                print(f'{i} = {a[i]}')


        else: print(a)
        return a
    return wrapper



def main():
    @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]
    test_1()
    test_2()
    test_3()
    test_4()
```

process_data.py

```python
import json
import sys
import print_result
import field
import cm_timer
import sort
import unique
import gen_random
# 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='utf-8') 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.print_result
def f1():
    return sort.Sort(unique.zapusk(field.zapusk(data, 'job-
name'), True))

@print_result.print_result
def f2(arg):
    return list(filter(lambda it: 'программист' in it, arg))
@print_result.print_result
def f3(arg):
    return list(map(lambda i: i+' с опытом Python', arg))
@print_result.print_result
def f4(arg):
    return list(zip(arg, gen_random.gen_random(len(arg),
100000, 200000)))


if __name__ == '__main__':
```

```
    with cm_timer.cm_timer_1():
        f4(f3(f2(f1())))
```

sort.py

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

def Sort(mass):
    return sorted(mass)
def main():
    result = sorted(data, key=abs, reverse=True)
    print(result)
    result_with_lambda = sorted(data, key=lambda i:abs(i), reverse=True)
    print(result_with_lambda)
```

unique.py

```
import gen_random
class Unique:
    def __init__(self, items, **kwargs):
        self.items = iter(items)
        self.ignore_case = kwargs.get('ignore_case', False)
        self.set = set()

    def __iter__(self):
        return self

    def __next__(self):
        item = next(self.items)
        if isinstance(item, str) and self.ignore_case:
            key = item.lower()
        else:
            key = item
        if key not in self.set:
            self.set.add(key)
            return key
def zapusk(data, bool):
    unique_iter = Unique(data, ignore_case=bool)
    a=[]
    for item in unique_iter:
        output = item
        if (output != None): a.append(output)
    data.clear()
    return a

def main():
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data = gen_random.gen_random(10, 1, 5)
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

```
unique_iter = Unique(data, ignore_case=False)
for item in unique_iter:
    output = item
    if (output != None): print(output, end=' ')
```

Выполнение программы

f4

1с программист с опытом Python , зарплата 108624 руб.

web-программист с опытом Python , зарплата 117280 руб.

веб - программист (php, js) / web разработчик с опытом Python , зарплата 179540 руб.

веб-программист с опытом Python , зарплата 193238 руб.

ведущий инженер-программист с опытом Python , зарплата 122786 руб.

ведущий программист с опытом Python , зарплата 177241 руб.

инженер - программист с опытом Python , зарплата 196816 руб.

инженер - программист асу тп с опытом Python , зарплата 177353 руб.

инженер-программист с опытом Python , зарплата 162757 руб.

инженер-программист (клинский филиал) с опытом Python , зарплата 180435 руб.

инженер-программист (орехово-зуевский филиал) с опытом Python , зарплата 192522 руб.

инженер-программист 1 категории с опытом Python , зарплата 180790 руб.

инженер-программист ккт с опытом Python , зарплата 104241 руб.

инженер-программист плис с опытом Python , зарплата 111517 руб.

инженер-программист сапоу (java) с опытом Python , зарплата 103971 руб.

инженер-электронщик (программист асу тп) с опытом Python , зарплата 188342 руб.

педагог программист с опытом Python , зарплата 174513 руб.

помощник веб-программиста с опытом Python , зарплата 158345 руб.

программист с опытом Python , зарплата 199615 руб.

программист / senior developer с опытом Python , зарплата 103287 руб.

программист 1с с опытом Python , зарплата 135926 руб.

программист c# с опытом Python , зарплата 189101 руб.

программист c++ с опытом Python , зарплата 115633 руб.

программист c++/c#/java с опытом Python , зарплата 170362 руб.

программист/ junior developer с опытом Python , зарплата 148097 руб.

программист/ технический специалист с опытом Python , зарплата 178449 руб.

программистр-разработчик информационных систем с опытом Python , зарплата 122785 руб.

системный программист (c, linux) с опытом Python , зарплата 120309 руб.

старший программист с опытом Python , зарплата 110985 руб.

time: 0.0291028022766113 28