Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

Лабораторная работа №3

По курсу «Разработка интернет приложений»

Подготовил:

Студент группы ИУ5-55Б.

Турчин Д.С.

04.10.2020

Проверил:

Преподаватель кафедры ИУ5
Гапанюк Ю.Е.

Москва, 2021 г.

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

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 (файл field.py)

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

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.

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. Пример:

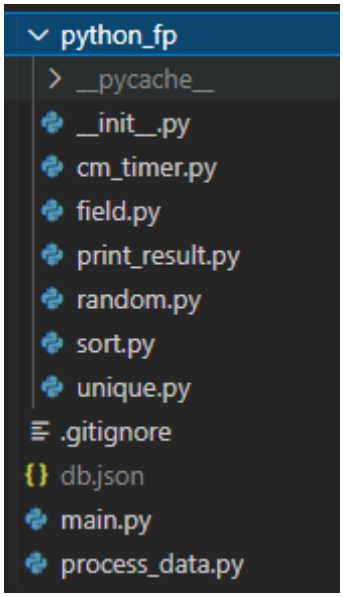
```
with cm_timer_1():
    sleep(5.5)
```

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).

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 db.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.



field.py

lab-3 > python_fp > field.py > field

```python
def field(dist_list, *args):
    assert len(args) > 0
    yield [x for x in [
        {arg:dist_item[arg] for arg in
          args if arg in dist_item and dist_item[arg]} for dist_item in dist_list if bool(dist_item)
      ] if bool(x)]

```

gen_random.py

```python
from random import randint

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

unique.py

lab-3 > python_fp > unique.py > Unique

```python
class Unique(object):
    __ignore_case = False
    __items = []
    __iter = iter(__items)

    def __init__(self, items, **kwargs):
        self.__ignore_case = False
        self.__items = []
        self.__iter = iter(self.__items)

        if 'ignore_case' in kwargs:
            self.__ignore_case = bool(kwargs['ignore_case'])

        for item in items:
            if type(item) == type('') and self.__ignore_case:
                if not any(str(x).lower() == item.lower() for x in self.__items):
                    self.__items.append(item)
            else:
                if not item in self.__items:
                    self.__items.append(item)

    def __next__(self):
        return next(self.__iter)

    def __iter__(self):
        return self.__iter

    def ignore_case(self):
        return self.__ignore_case
```

sort.py

lab-3 > python_fp > sort.py > sort

```python
def sort(arr):
    return sorted(arr, reverse=True, key = abs)

def sort_lambda(arr):
    return sorted(arr, reverse=True, key=lambda x: abs(x))
```

print_result.py

```python
def print_result(func):
  def decorator(*args):
    print(func.__name__)
    result = func(*args)
    if (type(result) == list):
      print(*result, sep='\n')
    elif (type(result) == dict):
      for key in result.keys():
        print(key, '=', result[key])
    else:
      print(result)
    return result

  return decorator
```

cm_timer.py

lab-3 > python_fp > cm_timer.py > time

```python
from time import time
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.__start = time()
    def __enter__(self):
        return self
    def __exit__(self, type, value, traceback):
        print('time: ', round(time() - self.__start, 2))

@contextmanager
def cm_timer_2():
    start = time()
    yield
    time_end = time()
    print('time: ', round(time() - start, 2))
```

process_data.py

lab-3 > process_data.py > {} json

```python
import json
import sys

from python_fp.print_result import print_result
from python_fp.cm_timer import cm_timer_1
from python_fp.sort import sort
from python_fp.random import gen_random
from python_fp.unique import Unique
from python_fp.field import field

path = r"./db.json"

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

@print_result
def f1(arg):
    return sorted(
        Unique(
            (x['job-name'] for x in list(field(arg, 'job-name'))[0]),
            ignore_case=True
        )
    )

@print_result
def f2(arg):
    return filter(lambda x: x.lower().startswith('программист'), arg)

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

@print_result
def f4(arg):
    salary = list(gen_random(len(arg), 100000, 200000))
    work = list(zip(arg, salary))
    return list(map(lambda x: x[0] + ', зарплата ' + str(x[1]) + ' руб.', work))


def main():
    print('\tproccess_data.py')
    with cm_timer_1():
        print(f4(f3(f2(f1(data)))))

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

Main.py – точка входа в программу с демонстрацией работы различных модулей данной лабораторной работы.

main.py

lab-3 > main.py > main > print_decorator_example2

```python
from python_fp.field import field
from python_fp.random import gen_random
from python_fp.unique import Unique
from python_fp.sort import sort, sort_lambda
from python_fp.print_result import print_result
from python_fp.cm_timer import cm_timer_1, cm_timer_2

from time import sleep

def main():
    print('\tfield')
    example = [
        {'title': 'Кровать'},
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': 'Торшер', 'price': 100, 'color': 'black'},
        {'title': 'бумага. Поле price - None', 'price': None, 'color': 'black'},
        {'titleNone': 'None. Оба поля для выборки - None', 'price': None, 'color': None},
    ]
    for item in field(example, 'title', 'price'):
        print(item)
    print('\n')

    print('\trandom')
    for item in gen_random(6, 10, 15):
        print(item)
    print('\n')

    print('\tuniq')
    uniq_example = [1, 1, 1,4,3,4,3,3,3,3,3, 1, 1, 2, 2, 2, 2, 2]
    uniq_example_lower_strings = ["hello", "Hello", "hellO", "hi", "Hi"]
    uniq = Unique(uniq_example)
    print(f"test 1. ignore_case: {uniq.ignore_case()}")
    for item in uniq:
        print(item)

    uniq_str = Unique(uniq_example_lower_strings, ignore_case = True)
    print(f"\ntest 2. ignore_case: {uniq_str.ignore_case()}")
    for item in uniq_str:
        print(item)
    print('\n')

    print('\tsort')
    sort_example = [4, -30, 100, -100, 123, 1, 0, -1, -4]
    print(sort(sort_example))
    print(sort_lambda(sort_example))
```

```
48      print('\n\tprint_decorator_example')
49      @print_result
50      def print_decorator_example1():
51          return 1
52      @print_result
53      def print_decorator_example2():
54          return 'iu5'
55      @print_result
56      def print_decorator_example4():
57          return {'a': 1, 'b': 2}
58      @print_result
59      def print_decorator_example3():
60          return [1, 2]
61      print_decorator_example1()
62      print_decorator_example2()
63      print_decorator_example3()
64      print_decorator_example4()
65
66      print('\ncm_timer')
67      with cm_timer_1():
68          sleep(1)
69      with cm_timer_2():
70          sleep(1.5)
71
72      print('\nprocess_data')
73
74  if __name__ == "__main__":
75      main()
```

Экранные формы с результатом.

```
        field
[{'title': 'Кровать'}, {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Торшер', 'price': 100}, {'title': 'бумага. Поле price - None'}]


        random
15
12
14
14
13
11


        uniq
test 1. ignore_case: False
1
4
3
2

test 2. ignore_case: True
hello
hi


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

        print_decorator_example
print_decorator_example1
1
print_decorator_example2
iu5
print_decorator_example3
1
2
print_decorator_example4
a = 1
b = 2

cm_timer
time:  1.0
time:  1.51

process_data
PS C:\Users\Denactive\source\repos\Denactive\WebApplicationDevs>
```