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 «Радиотехнический»

КАФЕДРА ИУ-5 «Системы обработки информации и управления»

Отчет по лабораторной работе № 3 по курсу

Разработка интернет-приложений

Тема работы: "Функциональные возможности языка Python."

Выполнил: Лисин. А. В.
Группа: РТ5-51Б

Дата выполнения: «27» ноября 2020 г.

Подпись:_____

Проверил: Гапанюк Ю. Е.

Дата проверки: «27» ноября 2020 г.

Подпись:_____

Москва, 2020 г.

Содержание

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

Описание задания:

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. 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.

Шаблон для реализации генератора:

```
# Пример:
# goods = [
#    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#    {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}

def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
```

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`

Шаблон для реализации генератора:

```
# 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
def gen_random(num_count, begin, end):
    pass
    # Необходимо реализовать генератор
```

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(1, 3, 10)
```

`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.

Шаблон для реализации класса-итератора:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # Нужно реализовать конструктор
        # В качестве ключевого аргумента, конструктор должен принимать bool-
параметр ignore_case,
        # в зависимости от значения которого будут считаться одинаковыми строки
в разном регистре
        # Например: ignore_case = True, Абв и АБВ - разные строки
        #           ignore_case = False, Абв и АБВ - одинаковые строки, одна из
которых удалится
        # По-умолчанию ignore_case = False
        pass
```

```
    def __next__(self):
        # Нужно реализовать __next__
        pass

    def __iter__(self):
        return self
```

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.

Шаблон реализации:

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

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

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.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@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__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат выполнения:

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

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.

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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

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

with open(path) 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
def f1(arg):
    raise NotImplemented


@print_result
def f2(arg):
    raise NotImplemented


@print_result
def f3(arg):
    raise NotImplemented


@print_result
def f4(arg):
```

```
    raise NotImplemented


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Ход работы:

Текст field.py:

```
goods = [
   {'title': 'Ковер', 'price': '2000', 'color': 'green'},
   {'title': 'Диван для отдыха', 'price': '5300', 'color': 'black'},
   {'title': 'Кровать', 'price': '10500', 'color': 'white'}
]


def field(items, *args):
   assert len(args) > 0, 'Не переданы аргументы полей словаря'
   count = 0
   for i in items:
      count += 1
      if len(args) == 1:
         print('\'', end='')
         print(i.get(*args), end='')
         if count < len(items):
            print('\'', end=', ')
         else:
            print('\'')
      else:
         k = 0
         print('{\'', end='')
         while k < len(args):
            temp_args = args[k]
            k += 1
            print(temp_args, end='')
            print('\': \'', end='')
            print(i.get(temp_args), end='')
            if k < len(args):
               print('\', \'', end='')
            else:
               print('\'', end='')
         if count < len(items):
            print('}, ', end='')
         else:
            print('}')


def main():
   # field(goods)
   field(goods, 'title')
   field(goods, 'title', 'price')
   field(goods, 'title', 'price', 'color')


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

Текст gen_random.py:

```
import random
```

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


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


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

Текст unique.py:

```
from lab_python_fp.gen_random import gen_random


class Unique(object):
    def __init__(self, items, **kwargs):
        self.used_elements = set()
        self.items = items
        self.counter = 0
        if len(kwargs) != 0:
            self.ignore_case = kwargs
        else:
            self.ignore_case = False

    def __next__(self):
        while True:                                 # как это работает?
            for item in self.items:
                temp_item = item
                self.counter += 1
                if (temp_item not in self.used_elements) \
                        and not(self.ignore_case and temp_item.swapcase() in self.used_elements):
                    self.used_elements.add(temp_item)
                    return temp_item
            else:
                raise StopIteration

    def __iter__(self):
        return self

def main():
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(data1)
    itr1 = Unique(data1)
    for i1 in itr1:
        print(i1, end=' ')
    print('\n', end='')
    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(data2)
    itr2 = Unique(data2)
    for i2 in itr2:
        print(i2, end=' ')
    print('\n', end='')
```

```
    print(data2)
    itr3 = Unique(data2, ignor_case=True)
    for i3 in itr3:
        print(i3, end=' ')
    print('\n', end='')
    data3 = gen_random(5, 1, 3)
    itr4 = Unique(data3)
    for i4 in itr4:
        print(i4, end=' ')


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

Текст sort.py:

```
def sort(x):
    return abs(x)


def main():
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

    result = sorted(data, key=sort, reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)


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

Текст print_result.py:

```
def print_result(func_to_decorate):

    def decorated_func():
        print(func_to_decorate.__name__)
        result = func_to_decorate()
        if type(result) is list:
            for i in result:
                print(i)
        elif type(result) is dict:
            for i in result:
                print(i, result.get(i), sep=' = ')
        else:
            print(result)

    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]


def main():
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()


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

Текст cm_timer.py:

```
import time
from contextlib import contextmanager


class cm_timer_1:

    def __init__(self):
        self.begin_time = time.time()

    def __enter__(self):
        pass

    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_type is not None:
            print(exc_type, exc_val, exc_tb)
        else:
            print('time: ', time.time() - self.begin_time)


@contextmanager
def cm_timer_2():
    begin_time = time.time()
    yield 1
    print('time: ', time.time() - begin_time)


def main():
    with cm_timer_1():
        time.sleep(5.5)

    with cm_timer_2():
        time.sleep(2.5)


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

Текст process_data.py:

```
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.print_result import print_result
from lab_python_fp.unique import Unique
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
import re
import json
import sys


path = 'data_light.json'


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


@print_result
def f1(arg):
    return Unique(field(arg, 'job-name'), ignore_case=True)


@print_result
def f2(arg):
    return filter(lambda x: re.search('Программист', x) or
re.search('программист', x), arg)


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


@print_result
def f4(arg):
    price = gen_random(len(arg), 100000, 200000)
    res = list(zip(arg, (list(map(lambda x: ', зарплата ' + x + ' руб',
''.join(str(list(price)))[1:-1].split(', '))))))
    return [''.join(i) for i in res]


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


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

Результат работы:

Экранные формы:

field.py:

```
C:\Lab_Python\Lab_Python\lab03\venv\Scripts\python.exe C:/Lab_Python/Lab_Python/lab03/lab_python_fp/field.py
'Ковер', 'Диван для отдыха', 'Кровать'
{'title': 'Ковер', 'price': '2000'}, {'title': 'Диван для отдыха', 'price': '5300'}, {'title': 'Кровать', 'price': '10500'}
{'title': 'Ковер', 'price': '2000', 'color': 'green'}, {'title': 'Диван для отдыха', 'price': '5300', 'color': 'black'}, {'title': 'Кровать', 'price': '10500', 'color': 'white'}

Process finished with exit code 0
```

gen_random.py:

```
C:\Lab_Python\Lab_Python\lab03\venv\
2 2 1 1 3
Process finished with exit code 0
```

unique.py:

```
C:\Lab_Python\Lab_Python\lab03\venv\Script
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
a A b B
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
a b
1 2
Process finished with exit code 0
```

sort.py:

```
C:\Lab_Python\Lab_Python\lab03\venv\Scr
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

print_result.py:

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

cm_timer.py:

```
C:\Lab_Python\Lab_Python\lab03\venv\
time:  5.514646291732788
time:  2.5129945278167725

Process finished with exit code 0
```

process_data.py:

```
C:\Lab_Python\Lab_Python\lab03\venv\Scripts\python.exe C:/Lab_Python/Lab_Python/lab03/process_data.py
f1
<lab_python_fp.unique.Unique object at 0x0000016376EF52E0>
f2
<filter object at 0x0000016376EF5280>
f3
Системный программист (C, Linux) с опытом Python
Веб-программист с опытом Python
Программист с опытом Python
Программист C++/C#/Java с опытом Python
1С программист с опытом Python
программист с опытом Python
Инженер-программист ККТ с опытом Python
инженер - программист с опытом Python
Инженер-программист (Клинский филиал) с опытом Python
Инженер-программист (Орехово-Зуевский филиал) с опытом Python
Ведущий программист с опытом Python
Программист 1С с опытом Python
Программистр-разработчик  информационных систем с опытом Python
Инженер - программист АСУ ТП с опытом Python
инженер-программист с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Инженер-электронщик (программист АСУ ТП) с опытом Python
Старший программист с опытом Python
Web-программист с опытом Python
Веб - программист (PHP, JS) / Web разработчик с опытом Python
Программист/ технический специалист с опытом Python
программист 1С с опытом Python
Программист C# с опытом Python
Инженер-программист 1 категории с опытом Python
Ведущий инженер-программист с опытом Python
Инженер-программист САПОУ (java) с опытом Python
Помощник веб-программиста с опытом Python
веб-программист с опытом Python
педагог программист с опытом Python
Инженер-программист ПЛИС с опытом Python
Инженер-программист с опытом Python
f4
Системный программист (C, Linux) с опытом Python, зарплата 162011 руб
Веб-программист с опытом Python, зарплата 153939 руб
Программист с опытом Python, зарплата 152191 руб
Программист C++/C#/Java с опытом Python, зарплата 140073 руб
1С программист с опытом Python, зарплата 194091 руб
программист с опытом Python, зарплата 128388 руб
Инженер-программист ККТ с опытом Python, зарплата 195699 руб
инженер - программист с опытом Python, зарплата 132654 руб
Инженер-программист (Клинский филиал) с опытом Python, зарплата 148498 руб
Инженер-программист (Орехово-Зуевский филиал) с опытом Python, зарплата 114536 руб
Ведущий программист с опытом Python, зарплата 149348 руб
Программист 1С с опытом Python, зарплата 102395 руб
Программистр-разработчик  информационных систем с опытом Python, зарплата 180736 руб
Инженер - программист АСУ ТП с опытом Python, зарплата 103142 руб
инженер-программист с опытом Python, зарплата 160732 руб
Программист C++ с опытом Python, зарплата 167953 руб
Программист/ Junior Developer с опытом Python, зарплата 100544 руб
Программист / Senior Developer с опытом Python, зарплата 146152 руб
Инженер-электронщик (программист АСУ ТП) с опытом Python, зарплата 180915 руб
Старший программист с опытом Python, зарплата 144868 руб
Web-программист с опытом Python, зарплата 157507 руб
Веб - программист (PHP, JS) / Web разработчик с опытом Python, зарплата 165859 руб
Программист/ технический специалист с опытом Python, зарплата 169223 руб
программист 1С с опытом Python, зарплата 187758 руб
Программист C# с опытом Python, зарплата 143681 руб
Инженер-программист 1 категории с опытом Python, зарплата 183381 руб
```