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: Мирсонов В. А.
Группа: РТ5-51Б

Дата выполнения: «12» октября 2020 г.

Подпись:_____

Проверил: Гапанюк Ю. Е.

Дата проверки: «12» октября 2020 г.

Подпись:_____

Москва, 2020 г.

Содержание

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

Общее 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)

Описание задачи

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 количество 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
```

Текст программы

```
def field(items, *args):
    assert len(args) > 0
    #Вывод значений одного ключа
    if len(args) == 1:
        for d in items:
            if args[0] in d.keys() and not d[args[0]] is None:
```

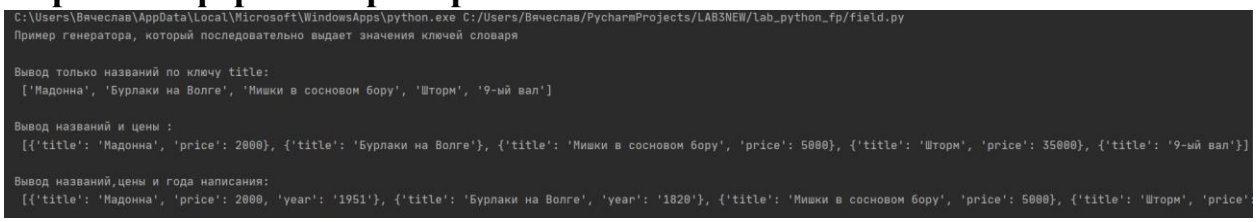
```
            yield d[args[0]]
    else:
        for d in items:
            elem = {}
            for arg in args:
                if arg in d.keys() and not d[arg] is None:
                    elem[arg] = d[arg]
            if len(elem) != 0:
                yield elem


def main():
    paintings = [
        {'title': 'Мадонна', 'price': 2000, 'year': '1951'},
        {'title': 'Бурлаки на Волге', 'year': '1820'},
        {'title': 'Мишки в сосновом бору', 'price': 5000},
        {'title': 'Шторм', 'price': 35000, 'year': '1859'},
        {'title': '9-ый вал', 'year': '1910'}
    ]

    print("Пример генератора, который последовательно выдает значения ключей словаря")
    print("\nВывод только названий по ключу title:\n", str(list(field(paintings, 'title'))))
    print("\nВывод названий и цены :\n", str(list(field(paintings, 'title', 'price'))))
    print("\nВывод названий,цены и года написания:\n", str(list(field(paintings, 'title', 'price', 'year'))))

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

Экранные формы с примерами выполнения



Задача 2 (файл gen_random.py)

Описание задачи

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 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
```

Текст программы

```
import random


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


def main():
    print("Пример генератора 10 случайных чисел от -10 до 10")
    print(str(list(gen_random(10, -10, 10)))[1:-1])

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

Экранные формы с примерами выполнения

```
C:\Users\Вячеслав\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/Вячеслав/PycharmProjects/LAB3NEW/lab_python_fp/gen_random.py
Пример генератора 10 случайных чисел от -10 до 10
5, 4, -5, -10, -5, 8, -5, 1, 5, 9

Process finished with exit code 0
```

Задача 3 (файл unique.py)

Описание задачи

- 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
```

Текст программы

```
from lab_python_fp.gen_random import gen_random


class Unique(object):
    def __init__(self,items,**kwargs):
        self.data = []
        strings = []
        for i in items:
            if isinstance(i, str):
                if 'ignore_case' in kwargs.keys() and kwargs['ignore_case'] is True:
                    if i.lower() not in strings:
                        strings.append(i.lower())
                        self.data.append(i)
                else:
                    if i not in self.data:
                        self.data.append(i)
            else:
                if i not in self.data:
                    self.data.append(i)


    def __next__(self):
        if not self.data:
            raise StopIteration
        return self.data.pop(0)

    def __iter__(self):
        return self


def main():
    data1 = [1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 2, 1]
    data2 = gen_random(5, 0, 2)
    data3 = ['a', 'b', 'c', 'A', 'B', 'C']

    print("Пример итератора уникальных данных")
    print(str(list(Unique(data1))))
    print(str(list(Unique(data2))))
    print(str(list(Unique(data3))))
    print(str(list(Unique(data3, ignore_case=True))))  # без учета регистра

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

Экранные формы с примерами выполнения

```
C:\Users\Вячеслав\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/Вячеслав/PycharmProjects/LAB3NEW/lab_python_fp/unique.py
Пример итератора уникальных данных
[1, 2, 3]
[2, 0, 1]
['a', 'b', 'c', 'A', 'B', 'C']
['a', 'b', 'c']

Process finished with exit code 0
```

Задача 4 (файл sort.py)

Описание задачи

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)
```

Текст программы

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


def main():
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
    print('Исходные данные: ', data)
    result = sorted(data, key=sort, reverse=True)
    print('Сортировка без lambda-функции: ', result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print('Сортировка при помощи lambda-функции: ', result_with_lambda)


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

Экранные формы с примерами выполнения

```
C:\Users\Вячеслав\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/Вячеслав/PycharmProjects/LAB3NEW/lab_python_fp/sort.py
Исходные данные:  [4, -30, 100, -100, 123, 1, 0, -1, -4]
Сортировка без lambda-функции:  [123, 100, -100, -30, 4, -4, 1, -1, 0]
Сортировка при помощи lambda-функции:  [123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

Задача 5 (файл print_result.py)

Описание задачи

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
```

Текст программы

```python
def print_result(func):
    def wrapper(*args):
        print(func.__name__)
        return_value = func(*args)
        if isinstance(return_value,list):
            print('\n'.join(str(value) for value in return_value))
        elif isinstance(return_value,dict):
            print('\n'.join((str(key) + ' = ' + str(return_value[key]) for key in return_value.keys())))
        else:
            print(return_value)
        return return_value
    return wrapper



@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 3]

def main():
    print("\nПример реализации декоратора")

    test_1()
    test_2()
    test_3()
    test_4()

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

Экранные формы с примерами выполнения

```
C:\Users\Вячеслав\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/Вячеслав/PycharmProjects/LAB3NEW/lab_python_fp/ptint_result.py

Пример реализации декоратора
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
3

Process finished with exit code 0
```

Задача 6 (файл cm_timer.py)

Описание задачи

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.

Текст программы

```python
import time
from time import sleep
from lab_python_fp.ptint_result import print_result



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

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('time: ', time.time() - self.time)


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


def main():
    print("Пример реализации контекстных менеджеров вывода времени работы блока кода")
    with cm_timer_1():
        time.sleep(3.5)

    with cm_timer_2():
        time.sleep(4.5)
```

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

Экранные формы с примерами выполнения

```
C:\Users\Вячеслав\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/Вячеслав/PycharmProjects/LAB3NEW/lab_python_fp/cm_timer.py
Пример реализации контекстных менеджеров вывода времени работы блока кода
time:  3.5003926753997803
cm_timer_2
<generator object cm_timer_2 at 0x0000028DD032CF20>
```

Задача 7 (файл process_data.py)

Описание задачи

- 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))))
```

Текст программы

```
from lab_python_fp.gen_random import gen_random
from lab_python_fp.field import field
from lab_python_fp.unique import Unique
from lab_python_fp.ptint_result import print_result
from lab_python_fp.cm_timer import cm_timer_1

import json
import sys

path = "C:\data_light.json"

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


@print_result
def f1(arg):
    return sorted(Unique(field(arg, 'job-name'), ignore_case=True))
    # Сортируем уникальные значения по полям наимаенование работы без учета регистра


@print_result
def f2(arg):
    return list(filter(lambda x: x.lower().startwith('программист'), arg))
    # Возвращаем отфильтрованные значения в которых значение работы начинается
    # с "программист", при этом делаем все в нижний регистр
```

```python
@print_result
def f3(arg):
    return list(map(lambda x: x + ' с опытом Python', arg))
    # Модифицируем каждый выбранный в f2 элемент массива, добавив строку
    # 'с опытом Python' при помощи функции map


@print_result
def f4(arg):
    salary = list(gen_random(len(arg), 100000, 200000))
    # Генерируем зарплаты для каждого выбранного в массив программиста
    jobs = list(zip(arg, salary))
    return list(map(lambda x: x[0] + ', зарплата ' + str(x[1]) + ' руб.', jobs))


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


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

Экранные формы с примерами выполнения

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик  информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 136690 руб.
Программист / Senior Developer с опытом Python, зарплата 137148 руб.
Программист 1С с опытом Python, зарплата 112827 руб.
Программист C# с опытом Python, зарплата 168499 руб.
Программист C++ с опытом Python, зарплата 119852 руб.
Программист C++/C#/Java с опытом Python, зарплата 157068 руб.
Программист/ Junior Developer с опытом Python, зарплата 192815 руб.
Программист/ технический специалист с опытом Python, зарплата 184106 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 177970 руб.
0.060166
```