Отчет по лабораторной работе№3 по курсу

"Разработка Интернет-Приложений"

«Функциональные возможности языка Python»

Выполнил:

Студент группы

ИУ5-55Б

Бахман А.А.

Москва, МГТУ – 2021

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. 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 = False, Абв и АБВ - разные строки
        #           ignore_case = True, 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 (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.

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:
cm_timer.py

```
from time import sleep, time
from contextlib import contextmanager
class cm_timer_1():
    def __init__(self):
        self.timer = time()

    def __enter__(self):
        pass
    def __exit__(self, exp_type, exp_value, traceback):
        self.timer = time() - self.timer
        print("time: {0:0.1f}".format(self.timer))

@contextmanager
def cm_timer_2():
    t = time()
    yield
    t = time() - t
    print("time: {0:0.1f}".format(t))

if __name__ == "__main__":
    with cm_timer_1():
        sleep(3.3)

    with cm_timer_2():
        sleep(3.3)
```

Результат работы:

Результат работы:

```
thon/лаб3/cm_timer.py"
time: 3.3
time: 3.3
```

field.py

```
def field(items, *args):
    if(len(args) == 0):
        print("Ошибка: введите аргументы")
        return
    elif(len(args) == 1):
        return [dicts[args[0]] for dicts in items if args[0] in dicts]
    else:
        retdict = list()
    for dicts in items:
        retdict.append({key:dicts[key] for key in dicts if (key in args) and
(dicts[key])})
    return retdict
def main():
    goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 50000},
    {'title': 'Кресло', 'price': 20000, 'color': 'grey', 'description':
'comfort'},
    {'title' : 'Линолеум', 'description': '20 meters, smooth'}
]
    print(field(goods, 'title', 'price'))
    print(field(goods, 'title'))

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

Результат работы:

```
thon/лаб3/field.py"
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 50000}, {'title': 'Кресло', 'price': 20000},
{'title': 'Линолеум'}]
['Ковер', 'Диван для отдыха', 'Кресло', 'Линолеум']
```

get_random.py

```
from random import randrange

def get_random(number, min, max):
    return [randrange(min, max+1) for i in range(number)]

if __name__ == "__main__":
    print(get_random(10, 1, 5))
```

Результат работы:

```
thon/лаб3/get_random.py"
[1, 3, 3, 5, 1, 3, 1, 1, 3, 5]
PS C:\Users\Xiaomi\Desktop\Study\5 sem\Python\лаб3>
```

print_result.py

```
def print_result(func):
```

```
    def decorated_func(*args, **kwargs):
        print(func.__name__)
        if isinstance(func(*args, **kwargs), dict):
            for key in func(*args,**kwargs):
                print(key, '=', func(*args, **kwargs)[key])
        elif isinstance(func(*args, **kwargs), list):
            for item in func(*args, **kwargs):
                print(item)
        else:
            print(func(*args, **kwargs))
        return func(*args, **kwargs)
    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__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат работы:

```
thon/лаб3/print_result.py"
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

process_data.py

```
import json
from cm_timer import cm_timer_1, cm_timer_2
from field import field
from unique import Unique
from print_result import print_result
from get_random import get_random

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

@print_result
def f2(data):
    return list(filter(lambda d: 'программист' in d[:11], data))

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

@print_result
def f4(data):
    return list(zip(data, list(map(lambda x: "Зарплата " + x + "
руб",map(str,(get_random(len(data), 100000, 200000)))))))

if __name__ == "__main__":
    with open('data_light.json', 'r', encoding='utf8') as jft:
        data = json.load(jft)
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Результат работы:

```
специалист по работе с клиентами
специалист по работе с крупными клиентами
специалист по работе с персоналом
специалист по реабилитации инвалидов
специалист по ремонту полиграфического оборудования
специалист по связям с общественностью волонтер
специалист по сертификатам и документообороту
специалист по сертификации переоборудованных автотранспортах средств
специалист по снабжению
специалист по социальной работе
специалист по социальной работе 1 категории (класса)
специалист по тендерам
специалист по учебно-методической работе
специалист по фольклору отдела социально-культурной деятельности
специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
специалист производственной системы
специалист расчетно-претензионного отдела
специалист тендерного отдела
специалист технической поддержки
специалист, ведущий
специалист-землеустроитель
стажировка у арбитражного управляющего / помощник арбитражного управляющего
```

```
программист/ junior developer
программист/ технический специалист
программистр-разработчик  информационных систем
f3
программист / senior developer с опытом Python
программист 1с 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 163386 руб')
('программист / senior developer с опытом Python', 'Зарплата 139330 руб')
('программист 1с с опытом Python', 'Зарплата 137626 руб')
('программист c# с опытом Python', 'Зарплата 129634 руб')
('программист c++ с опытом Python', 'Зарплата 173923 руб')
('программист c++/c#/java с опытом Python', 'Зарплата 114343 руб')
('программист/ junior developer с опытом Python', 'Зарплата 111994 руб')
('программист/ технический специалист с опытом Python', 'Зарплата 175200 руб')
('программистр-разработчик  информационных систем с опытом Python', 'Зарплата 154787 руб')
time: 1.1
```

Данные подгружаются из .json файла

sort.py

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

if __name__ == "__main__":
    result = sorted(data, key=abs, reverse=True)
    print(result)

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

Результат работы:

```
thon/лаб3/sort.py"
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
```

unique.py

```python
from get_random import get_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.unicList = list()
        self.data = items
        self.index = 0
        if 'ignore_case' in kwargs.keys() and kwargs['ignore_case'] == True:
            self.ignore_case = True
        else:
            self.ignore_case = False

    def __next__(self):
        while True:
            if self.index < len(self.data):
                if self.ignore_case == False:
                    current = self.data[self.index]
                    self.index += 1
                    if not current in self.unicList:
                        self.unicList.append(current)
                        return current
                else:
                    current = self.data[self.index]
                    self.index += 1
                    if isinstance(current, str):
                        current = current.lower()
                    if not current in self.unicList:
                        self.unicList.append(current)
                        return current
            else:
                raise StopIteration

    def __iter__(self):
        return self

if __name__ == "__main__":
    data = ['a','A','b', 'B', 'c', 'C', 'C', 'A', 'D']
    random_nums = get_random(10, 1, 7)
    print(data)
    for i in Unique(data, ignore_case=True):
        print(i, end='\t')
    print('\n')

    print(random_nums)
    for i in Unique(random_nums):
```

```
        print(i, end='\t')
```

Результат работы:

```
thon/лаб3/unique.py"
['a', 'A', 'b', 'B', 'c', 'C', 'C', 'A', 'D']
a       b       c       d

[4, 3, 5, 2, 7, 1, 5, 5, 1, 6]
4       3       5       2       7       1       6
```