МГТУ им. Баумана

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»

Дисциплина «Разработка интернет приложений»

Отчёт по лабораторной работе №3

«Функциональные возможности языка Python.»

Вариант 12

Выполнил:

Нагдимаев И. И.

ИУ5-55Б

Преподаватель:

Гапанюк Ю.Е.

Москва, 2020г

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
```

field.py

```
def field(items, *args):
    try:
        assert len(args) > 0
    except AssertionError:
        print("Нет второго ргумента")
    if len(args)==1:
        for i in range(len(goods)):
            if args[0] in goods[i] and goods[i].get(args[0])!=None:
                if i==len(goods)-1:
                    print(goods[i].get(args[0]),end="\n")
                else:
                    print(goods[i].get(args[0]),end=', ')
    else:
        for i in range(len(goods)):
            s={}
            for j in range(len(args)):
                if args[j] in goods[i] and goods[i].get(args[j]) != None:
```

```
                    s.update({args[j]: goods[i].get(args[j])})
            if i == len(goods) - 1:
                print(s,end="\n")
            else:
                print(s, end=', ')

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'blue'}
]
field(goods,'title', 'price')
field(goods,'title')
```

Результаты выполнения программы:

```
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
Ковер, Диван для отдыха

Process finished with exit code 0
```

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
```

gen_random.py

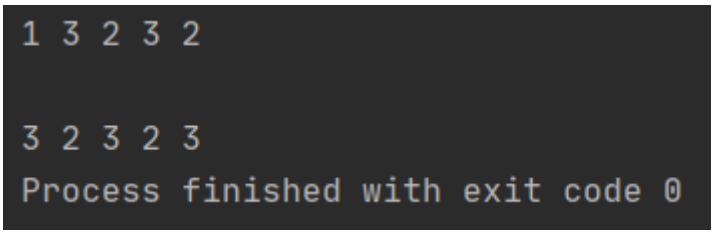
```
from random import randint


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


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


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

Результаты выполнения программы:



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
```

unique.py

```
import gen_random


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

    def __next__(self):
        while True:
            for item in self.items:
                temp_item=item
                self.index+=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, 2, 3, 3, 3, ]
    print(data1)
    iter1 = Unique(data1)
    for i in iter1:
        print(i, end=' ')
    print('\n')
    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(data2)
    iter2 = Unique(data2)
    for i in iter2:
        print(i, end=' ')
    print('\n')
    print(data2)
    iter3 = Unique(data2, ignore_case=True)
    for i in iter3:
        print(i, end=' ')
    print('\n')
    gen = gen_random.gen_random(3, 1, 3)
    iter4 = Unique(gen)
    for i in iter4:
        print(i, end=' ')


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

Результаты выполнения программы:

```
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3]
1 2 3

['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
```

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)
```

sort.py

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


if __name__ == '__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)
```

Результаты выполнения программы:

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

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
```

print_result.py

```python
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]


if __name__ == '__main__':
    test_1()
    test_2()
    test_3()
    test_4()
```

Результаты выполнения программы:

```
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.

cm_timer.py

```
from contextlib import contextmanager
from time import time, sleep


class cm_timer_1:
    def __init__(self):
        self.begin_time = time()

    def __enter__(self):
        pass

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print('time:', time() - self.begin_time)


@contextmanager
def cm_timer_2():
```

```
    begin_time=time()
    yield 1
    print('time:', time() - begin_time)


if __name__ == '__main__':

    with cm_timer_1():
        sleep(5.5)

    with cm_timer_2():
        sleep(2.5)
```

Результаты выполнения программы:

```
time: 5.514997482299805
time: 2.5034263134002686
```

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.

process_data.py

```
import json
from unique import Unique
from print_result import print_result
from cm_timer import cm_timer_1
from random import randint
from gen_random import gen_random
from field import field
import re
import sys

path = 'data_light.json'


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


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


@print_result
def f2(arg):
    return filter(lambda x: re.search(r'\bПрограммист\b', x) or
re.search(r'\bпрограммист\b', 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()
```

Результаты выполнения программы:

```
программист с опытом Python, зарплата 195464руб
Инженер-программист ККТ с опытом Python, зарплата 195900руб
инженер - программист с опытом Python, зарплата 186307руб
Инженер-программист (Клинский филиал) с опытом Python, зарплата 103455руб
Инженер-программист (Орехово-Зуевский филиал) с опытом Python, зарплата 123142руб
Ведущий программист с опытом Python, зарплата 111585руб
Программист 1С с опытом Python, зарплата 147254руб
Инженер - программист АСУ ТП с опытом Python, зарплата 126259руб
инженер-программист с опытом Python, зарплата 198658руб
Программист C++ с опытом Python, зарплата 112048руб
Программист/ Junior Developer с опытом Python, зарплата 154196руб
Программист / Senior Developer с опытом Python, зарплата 182960руб
Инженер-электронщик (программист АСУ ТП) с опытом Python, зарплата 162423руб
Старший программист с опытом Python, зарплата 174687руб
Web-программист с опытом Python, зарплата 136829руб
Веб - программист (PHP, JS) / Web разработчик с опытом Python, зарплата 187122руб
Программист/ технический специалист с опытом Python, зарплата 156209руб
программист 1С с опытом Python, зарплата 154696руб
Программист C# с опытом Python, зарплата 191714руб
Инженер-программист 1 категории с опытом Python, зарплата 112189руб
Ведущий инженер-программист с опытом Python, зарплата 110992руб
Инженер-программист САПОУ (java) с опытом Python, зарплата 160854руб
веб-программист с опытом Python, зарплата 161969руб
педагог программист с опытом Python, зарплата 140507руб
Инженер-программист ПЛИС с опытом Python, зарплата 122832руб
Инженер-программист с опытом Python, зарплата 143081руб
time: 0.04992985725402832
```