

**Московский государственный технический
Университет им. Н.Э. Баумана**

**Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»**

**Курс «Базовые компоненты интернет-технологий»
Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python»**

Выполнила:
студентка группы ИУ5-31Б
Гапеева О.Р.

Проверил:
Гапанюк Е.Ю.

2022 г.

Задание

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 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
```

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 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(10, 1, 3)
```

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 выводиться 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

field.py

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green', 'amount': 256},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black', 'amount': 102},
    {'title': 'Стол маленький', 'price': 2700, 'color': 'white', 'amount': 53},
    {'title': 'Ваза для цветов', 'price': 1590, 'color': 'blue', 'amount': 96},
]

def field(items, *args):
    try:
        assert len(args) > 0
        r = [{} for i in range(len(items))]
        for i in range(len(items)):
            for key in items[i]:
                if key in args:
                    r[i][key] = items[i][key]
        return r
    except:
        return "Not list of dicts as argument passed"
```

gen_random.py

```
from random import randint

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

if __name__ == '__main__':
    d = gen_random(2, 100, 500)
    for i in d:
        print(i)
```

unique.py

```
# Итератор для удаления дубликатов
class Unique(object):

    def __init__(self, items, **kwargs):
```



```

self.r = []

for key, value in kwargs.items():
    if key == 'ignore_case' and value == True:
        items = [i.lower() for i in items]

    for i in items:
        if i not in self.r:
            self.r.append(i)
    pass

def __next__(self):
    try:
        x = self.r[self.begin]
        self.begin += 1
        return x
    except:
        raise StopIteration

def __iter__(self):
    self.begin = 0
    return self

if __name__ == '__main__':
    a = [1,4,87,3,5,7,2,4,6,4,3,6,3,4,2]
    b = ['A', 'a', 'B', 'b']
    for i in Unique(b):
        print(i)

```

sort.py

```

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

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

    result_with_lambda = sorted(data, key=lambda a: -abs(a))
    print(result_with_lambda)

```

print_result.py

```
def print_result(f):
    def wrapper(a):
        print(f.__name__)
        res = f(a)
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for k,v in res.items():
                print(k, '=', v)
        return res
    return wrapper

@print_result
def test123(a):
    return a

if __name__ == '__main__':
    test123({'a':1, 'b':2, 'c':3})
    newName = test123
    newName([5,7,2,9,1,4,3])
```

cm_timer.py

```
import time
from contextlib import contextmanager

class cm_timer_1():
    def __enter__(self):
        self.start_time = time.time()
    def __exit__(self, type, value, traceback):
        print(time.time() - self.start_time)

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield True
    print(time.time()-start_time)

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(1)
    with cm_timer_2():
        time.sleep(2)
```

process_data.py

```
import json
from unique import Unique
from field import field
from operator import concat
from gen_random import gen_random
from print_result import print_result
from cm_timer import cm_timer_1

@print_result
def f1(a):
    return Unique([i['job-name'] for i in field(data, 'job-name')], ignore_case=True)

@print_result
def f2(a):
    return filter(lambda a: a.startswith('программист'), a)
#Метод str.startswith() возвращает True, если строка str начинается указанным
#префиксом prefix, в противном случае возвращает False.

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

@print_result
def f4(a):
    c = zip(a, gen_random(len(a), 100000, 200000))
    return c

with open('data_light.json', "r", encoding="utf-8") as f:
    data = json.loads(f.read())
    with cm_timer_1():
        (f4(f3(f2(f1(data)))))
```

Экранные формы с примерами выполнения программы

field.py

```
rs/olgap/Documents/BKIT/pythonProject/field.py
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стол малень-  
кий', 'price': 2700}, {'title': 'Ваза для цветов', 'price': 1590}]
```

gen_random.py

```
(env) PS C:\Users\olgap\Documents\BKIT> & c:/Users/olgap/Documents/BKIT/dz/env/Scripts/python.exe c:/Use  
rs/olgap/Documents/BKIT/pythonProject/gen_random.py  
497  
427  
_
```

unique.py

```
(env) PS C:\Users\olgap\Documents\BKIT> & c:/Users/olgap/Documents/BKIT/dz/env/Scripts/python.exe c:/Use  
rs/olgap/Documents/BKIT/pythonProject/unique.py  
A  
a  
B  
b  
_
```

sort.py

```
(env) PS C:\Users\olgap\Documents\BKIT> & c:/Users/olgap/Documents/BKIT/dz/env/Scripts/python.exe c:/Use  
rs/olgap/Documents/BKIT/pythonProject/sort.py  
[123, 100, -100, -30, 4, -4, 1, -1, 0]  
[123, 100, -100, -30, 4, -4, 1, -1, 0] _
```

print_result.py

```
(env) PS C:\Users\olgap\Documents\BKIT> & c:/Users/olgap/Documents/BKIT/dz/env/Scripts/python.exe c:/Use  
rs/olgap/Documents/BKIT/pythonProject/print_result.py  
test_1  
1  
test_2  
iu5  
test_3  
a = 1  
b = 2  
test_4  
1  
2  
_
```

cm_timer.py

```
(env) PS C:\Users\olgap\Documents\BKIT> & c:/Users/olgap/Documents/BKIT/dz/env/Scripts/python.exe c:/Use  
rs/olgap/Documents/BKIT/pythonProject/cm_timer.py  
1.005204200744629  
2.0087532997131348
```

process_data.py

f1

nothing to show

f2

nothing to show

f3

программист с опытом Python

программист c++/c#/java с опытом Python

программист 1с с опытом Python

программист-разработчик информационных систем с опытом Python

программист c++ с опытом Python

программист/ junior developer с опытом Python

программист / senior developer с опытом Python

программист/ технический специалист с опытом Python

программист c# с опытом Python

f4

программист с опытом Python 152215

программист c++/c#/java с опытом Python 194640

программист 1с с опытом Python 183779

программист-разработчик информационных систем с опытом Python 141235

программист c++ с опытом Python 197042

программист/ junior developer с опытом Python 186759

программист / senior developer с опытом Python 173316

программист/ технический специалист с опытом Python 121849

программист c# с опытом Python 150699

0.031792402267456055