

**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
Котова Анастасия

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

main.py

```

from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.sort import sort_array
from lab_python_fp.print_result import print_result_tests
from lab_python_fp.cm_timer import cm_timer_1, cm_timer_2
import time

def main():
    print("Example 1")
    goods = [
        {'title': 'Kover', 'price': 2000, 'color': 'green'},
        {'title': 'Divan dlia otidiha', 'color': 'black'}
    ]
    print(*field(goods, 'title'))
    print(*field(goods, 'title', 'price'))

    print("Example 2")
    gen_random(5, 1, 3)

    print("Example 3")
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(Unique(data))
    data = gen_random(10, 1, 3)
    print(Unique(data))
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(Unique(data))
    print(Unique(data, ignore_case = True))

    print("Example 4")
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
    sort_array(data, 1)

```

```

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
sort_array(data, 2)

print("Example 5")
print_result_tests()

print("Example 6")
with cm_timer_2():
    time.sleep(1.5)
with cm_timer_2():
    time.sleep(5.5)

if __name__ == "__main__":
    main()

```

field.py

```

def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        temp = []
        for i in items:
            temp_key = i.get(args[0], "None")
            if temp_key != "None":
                temp.append(temp_key)
        return temp
    else:
        k = []
        for i in items:
            temp = {}
            for j in args:
                temp_key = i.get(j, "None")
                if temp_key != "None":
                    temp.update({j: temp_key})
            k.append(temp)
        return k

```

gen_random.py

```

import random
def gen_random(num_count, begin, end):
    assert num_count > 0
    temp = []
    for i in range(num_count):
        temp.append(random.randrange(begin, end + 1))

```



```

print(*temp, sep = ", ")
return temp

```

unique.py

```

class Unique(object):
    def __init__(self, items, **kwargs):
        self.__r = []
        if kwargs == {}:
            try:
                self.__r = sorted(set([i for i in items]))
            finally:
                return

        for key, value in kwargs.items():
            if key == "ignore_case" and value == True:
                try:
                    self.__r = sorted(set([i.lower() for i in items]))
                finally:
                    break

    def unique(self):
        return self.__r

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

    def __str__(self):
        return str(self.__r)

    def __iter__(self):
        return self

```

sort.py

```

def sort_array(data, temp):
    if temp == 1:
        result = sorted(data, key = abs)[::-1]
        print(result)
    elif temp == 2:
        result_with_lambda = sorted(data, key = lambda x: abs(x))[::-1]
        print(result_with_lambda)
    else:
        print("ERROR")

```

print_result.py

```
def print_result(fun):
    def wrapper():
        print(fun.__name__)
        if isinstance(fun(), list):
            print(*fun(), sep = "\n")
        elif isinstance(fun(), dict):
            temp_fun = fun()
            for i in temp_fun:
                print(i, temp_fun.get(i), sep = " = ")
        else:
            print(fun())
    return wrapper

@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 print_result_tests():
    test_1()
    test_2()
    test_3()
    test_4()
```

cm_timer.py

```
from contextlib import contextmanager
from time import time
class cm_timer_1:
    def __enter__(self):
        self.__time_begin = time()
    def __exit__(self, type, value, traceback):
        print(time() - self.__time_begin)

@contextmanager
def cm_timer_2():
    time_begin = time()
```

```
yield
print(time() - time_begin)
```

process_data.py

```
import json
from field import field
from gen_random import gen_random
from unique import Unique
from sort import sort_array
from print_result import print_result
from cm_timer import cm_timer_1, cm_timer_2

from operator import concat

def f1(data):
    return Unique(field(data, 'job-name'), ignore_case = True).unique() # list

def f2(temp):
    return filter(lambda a: a.startswith('программист'), temp)

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

def f4(temp):
    return zip(temp, gen_random(len(temp), 100000, 200000))

if __name__ == '__main__':
    with open('lab_python_fp\data_light.json', encoding = "UTF-8-sig") as f:
        data = json.load(f)
        with cm_timer_1():
            for i in f4(f3(f2(f1(data)))):
                print(i)
```

Экранные формы с примерами выполнения программы

Example 1

Kover Divan dlia otdiha

```
{'title': 'Kover', 'price': 2000} {'title': 'Divan dlia otdiha'}
```

Example 2

1, 2, 2, 2, 2

Example 3

[1, 2]

1, 3, 1, 2, 1, 3, 2, 2, 3, 2

[1, 2, 3]

['A', 'B', 'a', 'b']

['a', 'b']

Example 4

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

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

Example 5

test_1

1

test_2

iu5

test_3

a = 1

b = 2

test_4

1

2

Example 6

1.500257968902588

5.501081943511963

Example 7

101405, 148062, 186362, 188011, 118187, 174824, 171108, 105773, 168375

('программист с опытом Python', 101405)

('программист / senior developer с опытом Python', 148062)

('программист 1с с опытом Python', 186362)

('программист c# с опытом Python', 188011)

('программист c++ с опытом Python', 118187)

('программист c++/c#/java с опытом Python', 174824)

('программист/ junior developer с опытом Python', 171108)

('программист/ технический специалист с опытом Python', 105773)

('программист-разработчик информационных систем с опытом Python', 168375)

0.005984783172607422