

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 №1**

Выполнил:
студент группы ИУ5-31Б
Егошин Дмитрий
Павлович

Подпись: _____

Дата: _____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

Подпись: _____

Дата: _____

Москва, 2021 г.

Лабораторная работа №3

Описание задания

Задание лабораторной работы состоит из решения нескольких задач. Файлы, содержащие решения отдельных задач, должны располагаться в пакете `lab_python_fr`. Решение каждой задачи должно располагаться в отдельном файле. При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Описание задачи 1

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.

Файл `field.py`

```
def field(items, *args):  
    assert len(args) > 0  
    if len(args) == 1:  
        for dict in items:  
            try:  
                if dict[args[0]] is not None:  
                    yield dict[args[0]]  
            except KeyError:  
                pass  
    else:  
        for dict in items:  
            res = {}  
            for key in args:
```

```

        try:
            if dict[key] is not None:
                res.update({key: dict[key]})
        except KeyError:
            pass
    if not len(res) == 0:
        yield res

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

x = field(goods, 'title')
for i in range(len(goods)):
    if i != len(goods) - 1:
        print(next(x), end=', ')
    else:
        print(next(x))

x = field(goods, 'title', 'price')
for i in range(len(goods)):
    if i != len(goods) - 1:
        print(next(x), end=', ')
    else:
        print(next(x))

```

Результат выполнения программы

```

"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович
Ковер, Диван для отдыха
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

Process finished with exit code 0

```

Описание задачи 2

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

Файл `gen_random.py`

```

from random import randint
def gen_random(num, begin, end):
    for x in range(num):
        yield randint(begin, end)

if __name__ == '__main__':

```

```
for i in gen_random(10, 0, 100):  
    print(i, end=' ')  
print()
```

Результат выполнения программы

```
"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович/  
82 45 7 71 19 97 86 41 17 52  
  
Process finished with exit code 0
```

Описание задачи 3

- 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.

Файл unique.py

```
from gen_random import gen_random

class Unique(object):
    def __init__(self, items, ignore_case=False, **kwargs):
        self.seen = set()
        self.items = items
        self.ic = ignore_case
        self.kwargs = kwargs

    def __next__(self):
        it = iter(self.items)
        while True:
            try:
                current = next(it)
            except StopIteration:
                raise
            else:
                if self.ic == True and isinstance(current, str):
                    temp = current[:]
                    if temp.lower() not in self.seen:
                        self.seen.add(temp.lower())
                        return current
                elif current not in self.seen:
                    self.seen.add(current)
                    return current

    def __iter__(self):
        return self

if __name__ == '__main__':
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print([i for i in Unique(data)])

    data = gen_random(10, 1, 3)
    print([i for i in Unique(data)])

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print([i for i in Unique(data)])
    print([i for i in Unique(data, ignore_case=True)])
```

Результат выполнения программы

```
"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович
[1, 2]
[1, 3, 2]
['a', 'A', 'b', 'B']
['a', 'b']
```

Описание задачи 4

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.

Файл sort.py

```
import lab_python_fp.unique
import lab_python_fp.field

def sort(data):
    ndata = []
    ndata = lab_python_fp.field.field(data, 'job-name')
    ndata = list(lab_python_fp.unique.Unique(ndata, True))
    ndata = sorted(ndata, reverse=True)
    for i in ndata:
        print(i, end=' ', ' ')
    print()
    return list(ndata)

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
result = sorted(data, reverse=True)
print(result)
resultLambda = lambda x: sorted(x, reverse=True)
print(resultLambda(data))
```

Результат выполнения программы

```
"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович
[123, 100, 30, 4, 1, 0, -1, -4, -30, -100]
[123, 100, 30, 4, 1, 0, -1, -4, -30, -100]

Process finished with exit code 0
```

Описание задачи 5

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.

Файл print_result.py

```
import lab_python_fp.sort

def printresult(func):
    def f(data):
        print('Имя функции {}'.format(func.__name__))
        print('Результат функции:')
        if type(func(data)) == list:
            for i in func(data):
                print(i)
        elif type(func(data)) == dict:
            for kw, arg in func(data).items():
                print('{} = {}'.format(kw, arg))

            else:
                print(func(data))
        return func(data)
    return f

@printresult
def test_1(data):
    return 1

@printresult
def test_2(data):
    return 'iu5'

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

@printresult
def test_4(data):
    return [1, 2]

data = 123
test_1(data)
test_2(data)
test_3(data)
test_4(data)
```

Результат выполнения программы

```
"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович
Имя функции test_1
Результат функции:
1
Имя функции test_2
Результат функции:
iu5
Имя функции test_3
Результат функции:
a = 1
b = 2
Имя функции test_4
Результат функции:
1
2
Process finished with exit code 0
```

Описание задачи 6

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`).

Файл `cm_timer.py`

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

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

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

@contextmanager
def cm_timer_2():
    start_time = time()
    yield
    print('time: ', time() - start_time)
```

```
with cm_timer_1():  
    sleep(1)  
with cm_timer_2():  
    sleep(1)
```

Результат выполнения программы

```
"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович/  
time: 1.000807762145996  
time: 1.0004796981811523  
  
Process finished with exit code 0
```

Описание задачи 7

- 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.

Файл process_data.py

```
import lab_python_fp.sort
import lab_python_fp.unique
import lab_python_fp.timer
import lab_python_fp.printresult
import lab_python_fp.gen_random
import lab_python_fp.field
import json

path = r'C:\Users\Дмитрий
Павлович\Desktop\lab3\lab_python_fp\data_light.json'
with open(path, encoding='utf-8') as f:
    data = json.load(f)

@lab_python_fp.printresult.printresult
def f1(arg):
    return lab_python_fp.sort.sort(arg)

@lab_python_fp.printresult.printresult
def f2(arg):
    return list(filter(filtration, arg))

@lab_python_fp.printresult.printresult
def f3(arg):
    return list(map(lambda item: item + ' с опытом Python', arg))

@lab_python_fp.printresult.printresult
def f4(arg):
    salary = [i for i in lab_python_fp.gen_random.gen_random(len(arg),
100000, 200000)]
    return ['{} зарплата {} руб.'.format(job, salary) for job, salary in
zip(arg, salary)]

def filtration(arg):
    if arg[0:11] == 'Программист':
        return True
    else:
        return False

with lab_python_fp.timer.cm_timer_1():
    f4(f3(f2(f1(data))))
```

Результат выполнения программы

Начало выполнения:

```
"C:\Users\Дмитрий Павлович\AppData\Local\Programs\Python\Python38-32\python.exe" "C:/Users/Дмитрий Павлович/
Имя функции f1
Результат функции:
юриисконсульт 2 категории, энтомолог, электросварщик, электромонтер станционного телевизионного оборудования,
юриисконсульт 2 категории, энтомолог, электросварщик, электромонтер станционного телевизионного оборудования,
юриисконсульт 2 категории
энтомолог
электросварщик
электромонтер станционного телевизионного оборудования
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
эколог
шлифовщик механического цеха
шлифовщик 5 разряда
шинномонтаж
```

Конец выполнения:

```
Результат функции:
Программист-разработчик информационных систем с опытом Python, зарплата 100683 руб.
Программист/ технический специалист с опытом Python, зарплата 141617 руб.
Программист/ Junior Developer с опытом Python, зарплата 100089 руб.
Программист C++/C#/Java с опытом Python, зарплата 143523 руб.
Программист C++ с опытом Python, зарплата 135631 руб.
Программист C# с опытом Python, зарплата 169574 руб.
Программист 1C с опытом Python, зарплата 147445 руб.
Программист / Senior Developer с опытом Python, зарплата 150544 руб.
Программист с опытом Python, зарплата 195086 руб.
time: 0.06102609634399414

Process finished with exit code 0
```