

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

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

Курс «Разработка интернет приложений»

Отчет по лабораторной работе №3
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-52Б
Бабин Артём

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2021 г.

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.

Задача 1 (файл `field.py`)

Необходимо реализовать генератор `field`. Генератор `field` последовательно выдает значения ключей словаря.

- 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.

Текст программы

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            result = item.get(args[0])
            if result is not None:
                yield result
    else:
        for item in items:
            result = {key: item.get(key) for key in args if item.get(key)
is not None}
            if result != {}:
                yield result
```

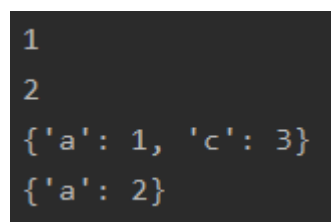
```

if __name__ == '__main__':
    lst = [
        {'a': 1, 'b': 2, 'c': 3},
        {'a': 2, 'c': None, 'd': -1},
        {'a': None, 'c': None, 'd': 5}
    ]

    for i in field(lst, 'a'):
        print(i)
    for i in field(lst, 'a', 'c'):
        print(i)

```

Экранная форма с примером выполнения программы:



```

1
2
{'a': 1, 'c': 3}
{'a': 2}

```

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.

Текст программы

```

import random

def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield random.randint(begin, end)

if __name__ == '__main__':
    for i in gen_random(5, 0, 2):
        print(i)

```

Экранная форма с примером выполнения программы:

```
2
2
1
0
2
```

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.

Текст программы

```
class Unique:
    def __init__(self, data, **kwargs):
        self.data = data
        self.type = type(data)
        self.index = 0
        self.unique_values = set()
        self.ignore_case = kwargs['ignore_case'] if
kwargs.get('ignore_case') is not None else False

    def __iter__(self):
        return self

    def check_elem(self, elem):
        if self.ignore_case:
```

```

        elem = elem.lower()
    if elem not in self.unique_values:
        self.unique_values.add(elem)
    return True
return False

def __next__(self):
    if self.type == list:
        while self.index < len(self.data):
            elem = self.data[self.index]
            self.index += 1
            if self.check_elem(elem):
                return elem
        else:
            for elem in self.data:
                if self.check_elem(elem):
                    return elem
    raise StopIteration

if __name__ == '__main__':
    data1 = [1, 1, 2, 2, 3, 4, 1, 0]
    data2 = ['a', 'b', 'A', 'a', 'B', 'b']
    data3 = (i for i in data2)
    for i in Unique(data1):
        print(i)
    print()
    for i in Unique(data2):
        print(i)
    print()
    for i in Unique(data3, ignore_case=True):
        print(i)

```

Экранная форма с примером выполнения программы:

```

1
2
3
4
0

a
b
A
B

a
b

```

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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы

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

Экранная форма с примером выполнения программы:

```
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 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 выводиться в столбик через знак равенства.

Текст программы

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if type(result) == list:
            for elem in result:
                print(elem)
        elif type(result) == dict:
            for (key, val) in result.items():
                print(f"{key} = {val}")
        else:
            print(result)
        return result
    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]

if __name__ == '__main__':
    print('!!!!!!!!')
    test_1()
    test_2()
```

```
test_3()
test_4()
```

Экранная форма с примером 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.

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).

Текст программы

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return 1

    def __exit__(self, exp_type, exp_value, traceback):
        print('Execution time: {} s'.format(time.time() -
self.start_time))
```



```

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield 1
    print('Execution time: {} s'.format(time.time() - start_time))

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

    with cm_timer_2():
        time.sleep(2)

```

Экранная форма с примером выполнения программы:

```

Execution time: 2.011338949203491 s
Execution time: 2.004998207092285 s

```

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

Текст программы

```
import json
```

```
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.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1
```

```
path = "../data_light.json"
with open(path, encoding='utf-8') as f:
    data = json.load(f)
```

```
@print_result
def f1(arg):
    return sorted([unique_prof for unique_prof in Unique([prof for prof
in field(arg, 'job-name')], ignore_case=True)])
```

```
@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith('Программист'), arg))
```

```
@print_result
```

```

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

@print_result
def f4(arg):
    return [f"{prof}, зарплата {salary} руб." for prof, salary in
zip(arg, gen_random(len(arg), 100_000, 200_000))]

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))

```

Экранная форма с примером выполнения программы:

```

f2
Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1C с опытом Python
Программист C# 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
Программист-разработчик информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 107048 руб.
Программист / Senior Developer с опытом Python, зарплата 196477 руб.
Программист 1C с опытом Python, зарплата 188075 руб.
Программист C# с опытом Python, зарплата 128623 руб.
Программист C++ с опытом Python, зарплата 154633 руб.
Программист C++/C#/Java с опытом Python, зарплата 109568 руб.
Программист/ Junior Developer с опытом Python, зарплата 180790 руб.
Программист/ технический специалист с опытом Python, зарплата 190169 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 183061 руб.
Execution time: 0.0955359935760498 s

```