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

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

Курс «Базовые компоненты интернет-технологий»

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

Выполнил:
студент группы ИУ5-31Б
Комиссаров Михаил Вячеславович

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич

Подпись и дата:

Подпись и дата:

Москва, 2021 г.

Задача 1

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

```
# 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
    rez = list()
    if len(args) == 1:
        for item in items:
            if item.keys().__contains__(args[0]):
                rez.append(item[args[0]])
    else:
        for item in items:
            d = dict()
            for key in args:
                if item.keys().__contains__(key):
                    d[key] = item[key]
            rez.append(d)
    return rez



def main():
    print(field(goods, 'title'))


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

```
['Ковер', 'Диван для отдыха']

Process finished with exit code 0
```

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

Process finished with exit code 0
```

Задача 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.

```
import random


def gen_random(num_count, begin, end):
    return (random.randint(begin, end) for i in range(num_count))


def main():
    print(*gen_random(10, 2, 7))


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

```
3 5 5 4 6 6 7 6 3 5

Process finished with exit code 0
```

Задача 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.

```
class Unique(object):
    def __init__(self, items, ignore_case=False):
        self.items = list(items)
        self.ignore_case = ignore_case

    def __next__(self):
        if len(self.items) == 0:
            raise StopIteration
        rez = self.items[0]
        if self.ignore_case:
            self.items = list(filter(lambda x: str(x).lower() != str.lower(rez),
self.items))
        else:
            self.items = list(filter(lambda x: x != rez, self.items))
        return rez

    def __iter__(self):
        return self


def main():
```

```
    print(*Unique([1, 1, 1, 2, 2, 3]))
    print(*Unique(['b', 'B', 'a', 'A', 'c', 'C']))
    print(*Unique(['b', 'B', 'a', 'A', 'c', 'C'], ignore_case=True))


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

```
1 2 3
b B a A c C
b a c

Process finished with exit code 0
```

Задача 4

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, который 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.

```
from operator import itemgetter


def sort1(data):
    return [data[i] for i, item in sorted(enumerate([abs(item) for item in
data]), key=itemgetter(1), reverse=True)]


def sort2(data, reverse=False):
    return sorted(data, key=lambda x: abs(x), reverse=True)


def main():
    print(sort1([10, -1, 9, -17, 8, 20, -21]))
    print(sort2([10, -1, 9, -17, 8, 20, -21]))


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

```
[-21, 20, -17, 10, 9, 8, -1]
[-21, 20, -17, 10, 9, 8, -1]

Process finished with exit code 0
```

Задача 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 выводить в столбик через знак равенства.

```
def print_result(funct):
    def wrapper(*args, **kwargs):
        print(funct.__name__)
        res = funct(*args, **kwargs)
        if isinstance(res, list):
            for i in res:
                print(i)
        elif isinstance(res, dict):
            for i in res:
                print(i, '=', res[i])
        else:
            print(res)
        return funct(*args, **kwargs)

    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 main():
    test_1()
    test_2()
    test_3()
    test_4()


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

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

Задача 6

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).

```python
from contextlib import contextmanager
import time

class cm_timer_1:

    def __init__(self):
        self._start_time = None

    def start(self):
        self._start_time = time.perf_counter()

    def stop(self):
        elapsed_time = time.perf_counter() - self._start_time
        self._start_time = None
        print("time: ", elapsed_time)

    def __enter__(self):
        self.start()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.stop()

@contextmanager
def cm_timer_2():
    try:
        start_time = time.perf_counter()
        yield start_time
    finally:
        elapsed_time = time.perf_counter() - start_time
        print("time: ", elapsed_time)


def main():
```

```
    with cm_timer_1():
        time.sleep(3)
    with cm_timer_2():
        time.sleep(3)


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

```
time:  3.0092076000000003
time:  3.0108085

Process finished with exit code 0
```

Задача 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.

```python
import json
from cm_timer import cm_timer_1
import unique
from field import field
from print_result import print_result
from gen_random import gen_random

path = r'C:\Users\masah\Desktop\lab3\data_light.json'

with open(path, 'r', encoding='utf-8') as f:
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(unique.Unique(field(arg, "job-name"), ignore_case=True),
key=lambda x: x.lower())


@print_result
def f2(arg):
    return list(filter(lambda x: x[:11].lower() == 'программист', arg))


@print_result
def f3(arg):
    return list(map(lambda x: '{} с опытом Python'.format(x), arg))


@print_result
def f4(arg):
    sal = [i for i in gen_random(len(arg), 100000, 200000)]
    return ['{}, зарплата {} руб.'.format(name, sal) for name, sal in zip(arg,
sal)]


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

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
```

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
```

```
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С 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
Программистр-разработчик  информационных систем с опытом Python
```

```
f4
Программист с опытом Python, зарплата 103206 руб.
Программист / Senior Developer с опытом Python, зарплата 110038 руб.
Программист 1С с опытом Python, зарплата 167319 руб.
Программист C# с опытом Python, зарплата 183438 руб.
Программист C++ с опытом Python, зарплата 160720 руб.
Программист C++/C#/Java с опытом Python, зарплата 141987 руб.
Программист/ Junior Developer с опытом Python, зарплата 126423 руб.
Программист/ технический специалист с опытом Python, зарплата 156239 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 184631 руб
```