



ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления» (ИУ-5)

## ЛАБОРАТОРНАЯ РАБОТА

№ 3-4

Функциональные возможности языка Python

Группа ИУ5-35Б

Студент Мешков 16.12.2024 /Д.Е. Мушкарин /

(Подпись, дата)

(И.О.Фамилия)

Преподаватель \_\_\_\_\_ /Ю. Е. Гапанюк/

(Подпись, дата)

(И.О.Фамилия)

2024

**Общее задание:**

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:

**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.

## Код:

### field.py

# Пример:

```
goods = [  
    {'title': 'Ковер', 'price': 2000},  
    {'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  
    for i in items:  
        if (len(args)>1):  
            field_dict = {}  
            for j in args:
```

```

        res = i.get(j)
        if (res is not None):
            field_dict[j] = res
    if field_dict:
        yield field_dict
    else:
        res = i.get(args[0])
        if res:
            yield res

```

```

if __name__ == "__main__":
    for i in field(goods,"title","price"):
        print(i)

```

### Вывод:

```

[user@nobara-pc] - [~/Documents/py_sem3/lab_python_fp] - [Вт дек 17, 20:21]
$ <git:(main*)> /usr/bin/python /home/user/Documents/py_sem3/lab_python_fp/field.py
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}

```

## 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.

### Пример:

`gen_random(5, 1, 3)` должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

### Код:

#### gen\_random.py

```

from random import randint

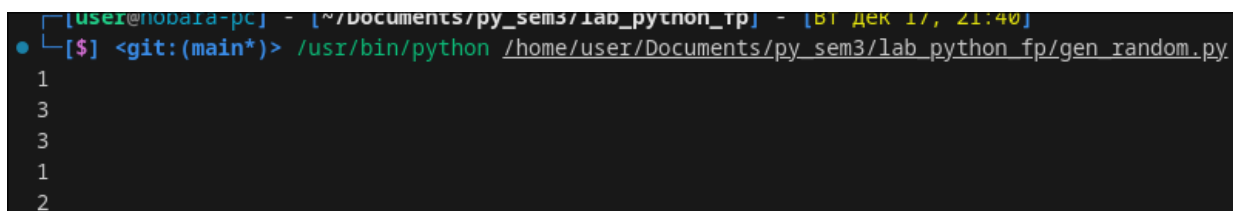
# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

```

# Hint: типовая реализация занимает 2 строки  
def gen\_random(num\_count, begin, end):

```
    for _ in range(num_count):  
        yield randint(begin, end)  
if __name__ == "__main__":  
    for i in gen_random(5, 1, 3):  
        print(i)
```

## Вывод:



```
[user@nova1a-pc] - [~/Documents/py_sem3/lab_python_fp] - [ВТ дек 17, 21:40]  
• [git:(main*)] /usr/bin/python /home/user/Documents/py_sem3/lab_python_fp/gen_random.py  
1  
3  
3  
1  
2
```

## 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.

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.

# Код:

## unique.py

```
# Итератор для удаления дубликатов
from gen_random import gen_random
```

```
class Unique(object):
    def __init__(self, items, **kwargs):
        ignore_case = kwargs.get("ignore_case", False)
        self.current = 0
        if ignore_case:
            temp = set()
            self._items = []
            for i in items:
                if isinstance(i, str):
                    if i.lower() not in temp:
                        temp.add(i.lower())
                        self._items.append(i)
                else:
                    if i not in temp:
                        temp.add(i)
                        self._items.append(i)
            else:
                self._items = list({i for i in items})

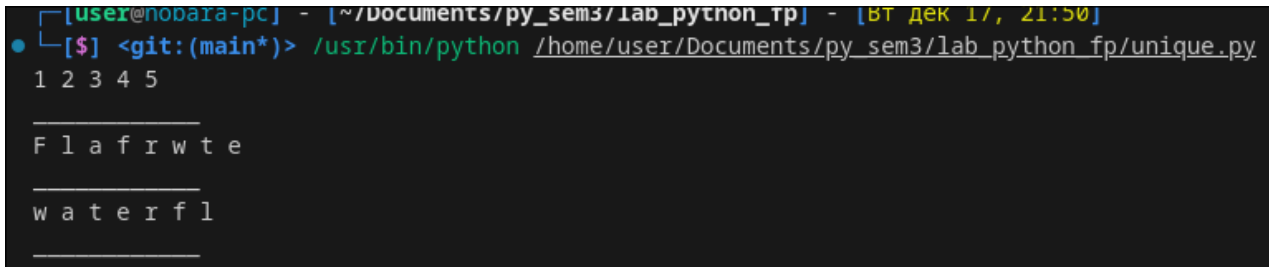
        def __iter__(self):
            return self

        def __next__(self):
            if self.current >= len(self._items):
                raise StopIteration
            item = self._items[self.current]
            self.current += 1
            return item

if __name__ == "__main__":
    data = [i for i in "waterfallF"] * 5
    for i in Unique(gen_random(100, 1, 5)):
        print(i, end=" ")
    print("\n_____")
    for i in Unique(data):
        print(i, end=" ")
    print("\n_____")
    for i in Unique(data, ignore_case = True):
```

```
print(i,end=" ")
print("\n_____")
```

## Вывод:



```
[user@nobarra-pc] - [~/Documents/py_sem3/lab_python_tp] - [ВТ дек 17, 21:50]
• [ ] <git:(main*)> /usr/bin/python /home/user/Documents/py_sem3/lab_python_fp/unique.py
1 2 3 4 5

_____
F l a f r w t e

_____
w a t e r f l

_____
```

## Задача 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.

## Код:

### sort.py

```
from cm_tymer import cm_timer_2
```

```
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 x: abs(x),
reverse=True)
print(result_with_lambda)
```

## Вывод:

```
● └─[$] <git:(main*)> /usr/bin/python /home/user/Documents/py_sem3/lab_python_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 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 выводиться в столбик через знак равенства.

## Код:

### `print_result.py`

```
def print_result(function):
def wrapper(*args, **kwargs):
    result = function(*args, **kwargs)
    print(function.__name__)
    if isinstance(result, list):
        for i in result:
            print(i)
    elif isinstance(result, dict):
        for i in result:
            print(f"{i}={result[i]}")
    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()
```

## Вывод:

```
└─[$] <git:(main*)> /usr/bin/python /home/user/Documents/py_sem3/lab_python_fp/print_result.py
!!!!!!!
test_1
1
test_2
iu5
test_3
a=1
b=2
test_4
1
2
```

## Задача 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`).

## Код:

### `cm_timer.py`

```
import time
from contextlib import contextmanager

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        end_time = time.time()
        elapsed_time = end_time - self.start_time
        print(f"time: {elapsed_time:.10f}")

@contextmanager
def cm_timer_2():
    start_time = time.time()
    try:
        yield
    finally:
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"time: {elapsed_time:.10f}")

if __name__ == '__main__':
    from time import sleep

    with cm_timer_2():
        sleep(3.5)

    with cm_timer_1():
        sleep(3.5)
```

## Вывод:

```
└─[$] <git:(main*)> /usr/bin/python /home/user/Documents/py_sem3/lab_python_fp/cm_timer.py
time: 3.5001482964
time: 3.5001060963
```

## 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.

## Код:

**process\_data.py**

```
import json
```

```
import sys
from print_result import print_result
from cm_typer import cm_timer_1
from time import sleep
from unique import Unique
from gen_random import gen_random
# Сделаем другие необходимые импорты

path = "lab2/data/data_light.json"

# 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,
заменяя `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):
    return [i for i in Unique(list(j["job-name"] for j in arg), ignore_case =
True)]

@print_result
def f2(arg):
    return [i for i in filter(lambda x: x.lstrip()[1:11].lower() ==
"программист", arg)]

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

```

@print_result
def f4(arg):
    pairs = zip(arg,gen_random(len(arg),100000,200000))
    return list(i+" " + str(j) for i,j in pairs)
if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))

```

## Вывод:

\* результат f1 включает ровно 1000 строк.

```

994 эколог
995 электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной
996 электромонтер по испытаниям и измерениям 4-6 разряд
997 электромонтер станционного телевизионного оборудования
998 электросварщик
999 энтомолог
1000 юриконсульт 2 категории
фельдшер фельдшерско - акушерского пункта
физик-эксперт
формовщик
фрезеровщик
фтизиатрия
химик
художник-постановщик
швея - мотористка
шиномонтаж
шлифовщик 5 разряда
шлифовщик механического цеха
эколог
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер станционного телевизионного оборудования
электросварщик
энтомолог
юрисконсульт 2 категории
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 170166
Программист / Senior Developer с опытом Python 129665
Программист 1C с опытом Python 157633
Программист C# с опытом Python 166865
Программист C++ с опытом Python 135818
Программист C++/C#/Java с опытом Python 143928
Программист/ Junior Developer с опытом Python 135131
Программист/ технический специалист с опытом Python 119910
Программист-разработчик информационных систем с опытом Python 187939
time: 0.0097558498

```