

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

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

Курс «Парадигмы и конструкции языков программирования»

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

Выполнил:

студент группы ИУ5-31Б
Иванов Кирилл

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Подпись и дата:

Постановка задачи

Задание:

Задача 1 (файл `field.py`)

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 sm_timer.py)

Необходимо написать контекстные менеджеры sm_timer_1 и sm_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 NotImplementedError
```

```
@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

1 задание

```
import time

# на основе класса

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

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

# Пример использования:

from time import sleep

with cm_timer_1():
    sleep(5.5)

# библиотека contextlib

from contextlib import contextmanager
import time

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

# Пример использования:

from time import sleep

with cm_timer_2():
    sleep(5.5)
```

2 задание

```
def field(items, *args):
    assert len(args) > 0

    for item in items:
        if len(args) == 1:
            # Если передано только одно поле, выдаем его значение
            field_name = args[0]
```

```

        value = item.get(field_name)
        if value is not None:
            yield value
    else:
        # Если передано несколько полей, формируем словарь с заполненными
        # значениями
        filtered_item = {field: item.get(field) for field in args if
            item.get(field) is not None}
        if filtered_item:
            yield filtered_item

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

# Выдаем значения полей 'title'
for value in field(goods, 'title'):
    print(value)

# Выдаем словари с заполненными значениями полей 'title' и 'price'
for item in field(goods, 'title', 'price'):
    print(item)

```

3 задание

```

import random

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

# Пример использования:
random_numbers = gen_random(5, 1, 3)

for number in random_numbers:
    print(number)

```

4 задание

```

def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)

        print(func.__name__)

        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f'{key} = {value}')
        else:
            print(result)

        return result

    return wrapper

# Пример использования 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()

```

5 задание

```

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
# с lambda
if __name__ == '__main__':
    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
# без lambda
if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True)
    print(result)

```

6 задание

```

import random

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

    def __next__(self):
        while True:
            item = next(self.items)
            key = item.lower() if self.ignore_case and isinstance(item, str)
            else item
            if key not in self.seen:
                self.seen.add(key)
                return item

    def __iter__(self):
        return self

# Пример использования:

```

```
# Список с дубликатами
data_list = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
unique_iterator = Unique(data_list)

# Выводит только уникальные элементы
for item in unique_iterator:
    print(item)

# Генератор с дубликатами
def gen_random(count, start, end):
    for _ in range(count):
        yield random.randint(start, end)

data_gen = gen_random(10, 1, 3)
unique_iterator_gen = Unique(data_gen)

# Выводит только уникальные элементы
for item in unique_iterator_gen:
    print(item)

# Список строк с дубликатами (учитывая регистр)
data_strings = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
unique_iterator_case_sensitive = Unique(data_strings)

# Выводит только уникальные элементы с учетом регистра
for item in unique_iterator_case_sensitive:
    print(item)

# Список строк с дубликатами (игнорируя регистр)
data_strings_ignore_case = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
unique_iterator_ignore_case = Unique(data_strings_ignore_case,
ignore_case=True)

# Выводит только уникальные элементы, игнорируя регистр
for item in unique_iterator_ignore_case:
    print(item)
```