

**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 №4-5

Выполнил:

студент группы ИУ5-32Б
Кудрявцев Андрей

Подпись и дата:

Проверила:

преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Москва, 2023 г.

Постановка задачи

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. 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, который 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 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. 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 NotImplemented

@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) field.py

4 usages ⚠ 2

```

1  def field(items, *args):
2      assert len(args) > 0
3
4      for item in items:
5          if len(args) == 1:
6              yield item.get(args[0])
7          else:
8              filtered_item = {}
9              for arg in args:
10                 if arg in item:
11                     filtered_item[arg] = item[arg]
12             if len(filtered_item) > 0:
13                 yield filtered_item
14
15  if __name__ == '__main__':
16      goods = [
17          {'title': 'Ковер', 'price': 2000, 'color': 'green'},
18          {'title': 'Диван для отдыха', 'color': 'black'}
19      ]
20
21      for value in field(goods, *args: 'title'):
22          print(value)
23      print('-----')
24      for value in field(goods, *args: 'title', 'price'):
25          print(value)
26      print('-----')
27      for value in field(goods, *args: 'title', 'color'):
28          print(value)
29

```

2) gen_random.py

```
1  """
2  # Пример: gen_random(5, 1, 3) должен выдать 5 случайных чисел
3  # в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
4  # Hint: типовая реализация занимает 2 строки
5  def gen_random(num_count, begin, end):
6      pass
7      # Необходимо реализовать генератор
8  """
9  import random
10  2 usages
11  def gen_random(num_count, begin, end):
12      for _ in range(num_count):
13          yield random.randint(begin, end)
14
15
16  if __name__ == '__main__':
17      a = gen_random(num_count=5, begin=1, end=3)
18      print(a)
19      print(next(a))
20      print(next(a))
21      print(next(a))
22      print(next(a))
23      print(next(a))
24
```


3) unique.py

```
1 class Unique(object):
2     def __init__(self, items, **kwargs):
3         ignore_case = bool(kwargs.get('ignore_case'))
4         self.items = []
5         prev_items = set()
6         for i in items:
7             if ignore_case and i.lower() not in prev_items:
8                 self.items.append(i)
9                 prev_items.add(i.lower())
10            elif not ignore_case and i not in prev_items:
11                self.items.append(i)
12                prev_items.add(i)
13        self.index = 0
14
15    def __next__(self):
16        if self.index < len(self.items):
17            res = self.items[self.index]
18            self.index += 1
19            return res
20        raise StopIteration
21
22    def __iter__(self):
23        self.index = 0
24        return self
25
26
27 > if __name__ == '__main__':
28     data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
29     data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
30
31     a = Unique(data1, ignore_case=False)
32     print(next(a))
33     print(next(a))
34     print('-----')
35     b = Unique(data2, ignore_case=False)
36     print(next(b))
37     print(next(b))
38     print(next(b))
39     print(next(b))
40     print('-----')
41     c = Unique(data2, ignore_case=True)
42     print(next(c))
43     print(next(c))
44
```

4) sort.py

```
1 data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
2
3 > if __name__ == '__main__':
4     result = sorted(data, key=abs, reverse=True)
5     print(result)
6
7     result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
8     print(result_with_lambda)
9
```

5) print_result.py

```
9 usages
1 def print_result(func):
2     def wrapper(*args, **kwargs):
3         result = func(*args, **kwargs)
4         print(func.__name__)
5
6         if type(result) == list:
7             print(*result, sep='\n')
8         elif type(result) == dict:
9             for key, value in result.items():
10                 print(f'{key} = {value}')
11         else:
12             print(result)
13         return result
14     return wrapper
15
16 1 usage
17 @print_result
18 def test_1():
19     return 1
20
21 1 usage
22 @print_result
23 def test_2():
24     return 'iu5'
25
26 1 usage
27 @print_result
28 def test_3():
29     return {'a': 1, 'b': 2}
30
31 1 usage
32 @print_result
33 def test_4():
34     return [1, 2]
35
36 > if __name__ == '__main__':
37     print('!!!!!!!')
38     test_1()
39     test_2()
40     test_3()
41     test_4()
42
```

6) cm_timer.py

```
1 import time
2 from contextlib import contextmanager
3
4 class cm_timer_1:
5     def __enter__(self):
6         self.start_time = time.time()
7
8     def __exit__(self, exc_type, exc_val, exc_tb):
9         elapsed_time = time.time() - self.start_time
10        print("time: ", elapsed_time)
11
12 @contextmanager
13 def cm_timer_2():
14     start_time = time.time()
15     yield
16     elapsed_time = time.time() - start_time
17     print("time: ", elapsed_time)
18
19 if __name__ == '__main__':
20     with cm_timer_1():
21         time.sleep(5.5)
22
23     with cm_timer_2() as start_time:
24         time.sleep(5.5)
```

7) process_data.py

```
1 import json
2
3 import field
4 import gen_random
5 import unique
6 from print_result import print_result
7 from cm_timer import cm_timer_1
8
9 path = 'data_light.json'
10 with open(path, encoding='utf-8') as f:
11     data = json.load(f)
12
13 @print_result
14 def f1(arg):
15     return sorted(unique.Unique(field.field(data, *args: 'job-name'), ignore_case=True))
16
17 @print_result
18 def f2(arg):
19     return list(filter(lambda x: x.lower().startswith("программист"), arg))
20
21 @print_result
22 def f3(arg):
23     return list(map(lambda x: (x + " с опытом Python"), arg))
24
25 @print_result
26 def f4(arg):
27     salaries = gen_random.gen_random(len(arg), begin: 100_000, end: 200_000)
28     result = []
29     for job, salary in zip(arg, salaries):
30         result.append(f"{job}, зарплата {salary} py6")
31     return result
32
33 if __name__ == '__main__':
34     with cm_timer_1():
35         f4(f3(f2(f1(data))))
```

Результат выполнения программы

1) field.py

```
Ковер
Диван для отдыха
-----
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}
-----
{'title': 'Ковер', 'color': 'green'}
{'title': 'Диван для отдыха', 'color': 'black'}
```

2) gen_random.py

```
<generator object gen_random at 0x00000021E2738F7B0>
3
1
1
3
2
```

3) unique.py

```
1
2
-----
a
A
b
B
-----
a
b
```

4) sort.py

```
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

5) print_result.py

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

6) cm_timer.py

```
time: 5.507128953933716  
time: 5.506460905075073
```

7) process_data.py

```
f1  
1С программист  
2-ой механик  
3-ий механик  
4-ый механик  
4-ый электромеханик  
ASIC специалист  
JavaScript разработчик  
RTL специалист  
Web-программист  
[химик-эксперт  
web-разработчик  
Автожестящик  
Автоинструктор  
Автомаляр  
Автомойщик  
Автор студенческих работ по различным дисциплинам  
Автослесарь - моторист  
Автоэлектрик  
Агент  
Агент банка
```

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

Программистр-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

Программистр-разработчик информационных систем с опытом Python

f4

Программист с опытом Python, зарплата 166281 руб

Программист / Senior Developer с опытом Python, зарплата 123237 руб

Программист 1C с опытом Python, зарплата 179061 руб

Программист C# с опытом Python, зарплата 141307 руб

Программист C++ с опытом Python, зарплата 183970 руб

Программист C++/C#/Java с опытом Python, зарплата 168615 руб

Программист/ Junior Developer с опытом Python, зарплата 174765 руб

Программист/ технический специалист с опытом Python, зарплата 159809 руб

Программистр-разработчик информационных систем с опытом Python, зарплата 142646 руб

time: 0.04358935356140137