

**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 ЛР№3-4

Выполнил:
студент группы ИУ5-34Б
Кожевников М. А.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2023 г.

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
```

4 usages

```
def field(dicts, *args):
    assert len(args) > 0
    ans = []
    for cur_dict in dicts:
        if len(args) == 1:
            for key in args:
                if cur_dict.get(key) is not None:
                    ans.append(cur_dict.get(key))
        else:
            cur_ans = dict()
            for key in args:
                if cur_dict.get(key) is not None:
                    cur_ans[key] = cur_dict[key]
            ans.append(cur_ans)
    return ans

if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    print(field(goods, *args: 'title'))
    print(field(goods, *args: 'title', 'price'))
```

F:\Python\python.exe F:\Уник\прога\proga_sem3\Lab3-4\field.py

['Ковер', 'Диван для отдыха']

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

Process finished with exit code 0

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

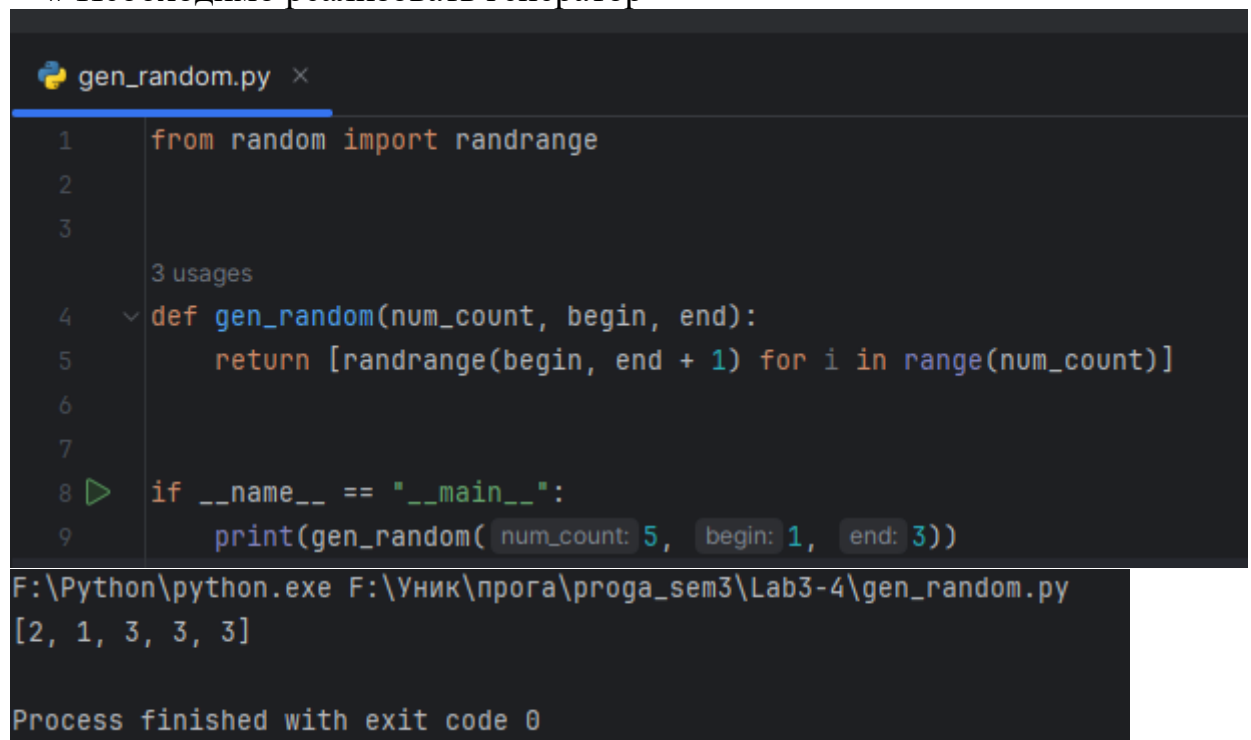
в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Hint: типовая реализация занимает 2 строки

def gen_random(num_count, begin, end):

pass

Необходимо реализовать генератор



```
gen_random.py x
1  from random import randrange
2
3
4  3 usages
5  def gen_random(num_count, begin, end):
6      return [randrange(begin, end + 1) for i in range(num_count)]
7
8  if __name__ == "__main__":
9      print(gen_random(num_count=5, begin=1, end=3))

F:\Python\python.exe F:\Уник\прога\proga_sem3\Lab3-4\gen_random.py
[2, 1, 3, 3, 3]

Process finished with exit code 0
```

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
```

```

3 usages
class Unique(object):
    def __init__(self, items, **kwargs):
        self.cur = 0
        self.ignore_case = kwargs.get("ignore_case", False)
        self.data = []
        if self.ignore_case:
            for el in items:
                if el not in self.data and type(el) is not str:
                    self.data.append(el)
                elif type(el) is str and el.lower() not in self.data:
                    self.data.append(el.lower())
            else:
                for el in items:
                    if el not in self.data:
                        self.data.append(el)

        def __next__(self):
            if self.cur < len(self.data):
                self.cur += 1
                return self.data[self.cur - 1]
            raise StopIteration

        def __iter__(self):
            return self

if __name__ == '__main__':
    arr = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    # arr = [1,2,2,3,3,1]
    it = Unique(arr, ignore_case=True)
    for el in it:
        print(el)

```

F:\Python\python.exe F:\Уник\прога\proga_sem3\Lab3-4\unique.py

1
2
3

Process finished with exit code 0

F:\Python\python.exe F:\Уник\прога\proga_sem3\Lab3-4\unique.py

a
b

Process finished with exit code 0

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:

С использованием `lambda`-функции.

Без использования `lambda`-функции.

Шаблон реализации:

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

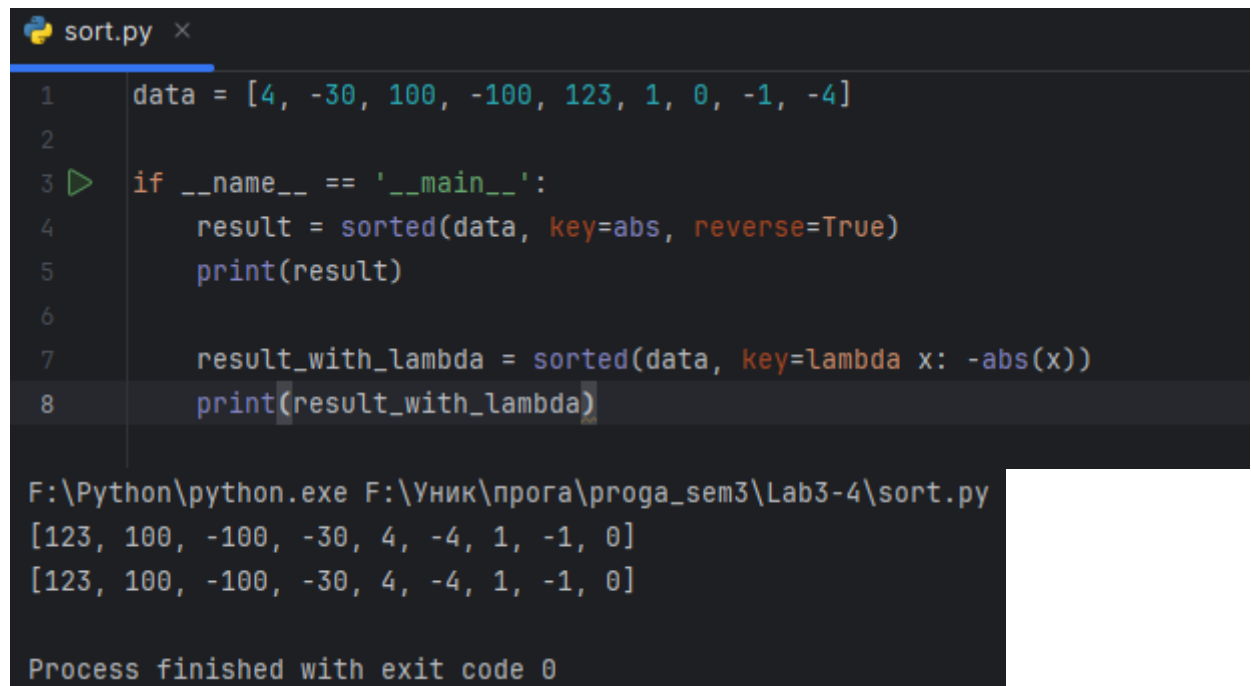
```
if __name__ == '__main__':
```

```
    result = ...
```

```
    print(result)
```

```
    result_with_lambda = ...
```

```
    print(result_with_lambda)
```



```
sort.py x
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))
8     print(result_with_lambda)

F:\Python\python.exe F:\Уник\пора\proga_sem3\Lab3-4\sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 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.

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

```

1  # Здесь должна быть реализация декоратора
2
3  9 usages
4  def print_result(func):
5      def wrapper(*args, **kwargs):
6          print(func.__name__)
7          res = func(*args, **kwargs)
8          if type(res) is list:
9              for el in res:
10                 print(el)
11             elif type(res) is dict:
12                 for key in res:
13                     print("{} = {}".format(*args, key, res[key]))
14             else:
15                 print(res)
16             return res
17         return wrapper
18
19  1 usage
20  @print_result
21  def test_1():
22      return 1
23
24  1 usage
25  @print_result
26  def test_2():
27      return 'iu5'
28
29  1 usage
30  @print_result
31  def test_3():
32      return {'a': 1, 'b': 2}
33
34  1 usage
35  @print_result
36  def test_4():
37      return [1, 2]
38
39  if __name__ == '__main__':
40      print('!!!!!!!')
41      test_1()
42      test_2()
43      test_3()
44      test_4()
45

```

F:\Python\python.exe F:\Уник\прогр\proga_sem3\Lab3-4\sort.py

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

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

Process finished with exit code 0

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).

```
cm_timer.py x
1  from time import time, sleep
2  from contextlib import contextmanager
3
4
5  3 usages
6  class cm_timer_1:
7      def __enter__(self):
8          self.start_time = time()
9
10     def __exit__(self, exc_type, exc_val, exc_tb):
11         self.res_time = time() - self.start_time
12         print("time: {:.3}".format(self.res_time))
13
14  1 usage
15  @contextmanager
16  def cm_timer_2():
17      start_time = time()
18      try:
19          yield
20      finally:
21          delta_time = time() - start_time
22          print("time: {:.3}".format(delta_time))
23
24  if __name__ == '__main__':
25      with cm_timer_1():
26          sleep(2.1)
27
28      with cm_timer_2():
29          sleep(2.1)
```

```
F:\Python\python.exe F:\Уник\прога\proga_sem3\Lab3-4\cm_timer.py
time: 2.1
time: 2.1
```

```
Process finished with exit code 0
```

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))))
```

```

1 from print_result import print_result
2 from unique import Unique
3 from field import field
4 from cm_timer import cm_timer_1
5 from gen_random import gen_random
6 import json
7 import sys
8
9 path = "data_light.json"
10
11 # Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария
12
13 with open(path, encoding="utf-8") as f:
14     data = json.load(f)
15
16 # Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`
17 # Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
18 # В реализации функции f4 может быть до 3 строк
19
20 1 usage
21 @print_result
22 def f1(arg):
23     return sorted([el for el in Unique(field(arg, *args: 'job-name'), ignore_case=True)])
24
25 1 usage
26 @print_result
27 def f2(arg):
28     return list(filter(lambda s: s.startswith("программист"), arg))
29
30 1 usage
31 @print_result
32 def f3(arg):
33     return list(map(lambda x: x + " с опытом Python", arg))
34
35 1 usage
36 @print_result
37 def f4(arg):
38     return [{"{:}, запущена {:} pyб.".format(*args: man, sal) for man, sal in zip(arg, gen_random(len(arg), begin: 100_000, end: 200_000))}]
39
40 if __name__ == '__main__':
41     with cm_timer_1():
42         f4(f3(f2(f1(data))))

```

Результаты работы программы приведены в отдельном файле Задача7.txt