**Код:**

def field(items, \*args):

assert len(args) > 0, "Необходимо указать хотя бы один ключ"

if len(args) == 1:

key = args[0]

for item in items:

value = item.get(key)

if value is not None:

yield value

else:

for item in items:

result = {key: item[key] for key in args if item.get(key) is not None}

if result:

yield result

# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

goods = [

{'title': 'Ковер', 'price': 2000, 'color': 'green'},

{'title': 'Диван для отдыха', 'color': 'black'}

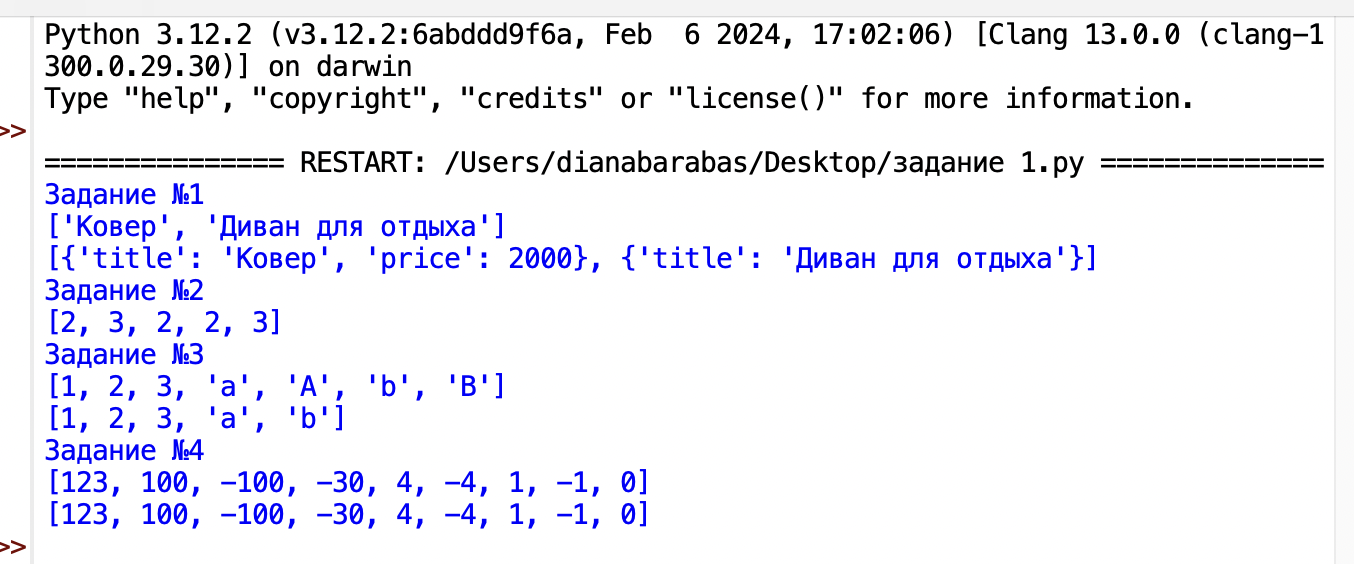
]

print("Задание №1")

print(list(field(goods, 'title'))) # Вывод: ['Ковер', 'Диван для отдыха']

print(list(field(goods, 'title', 'price'))) # Вывод: [{'title': 'Ковер', 'pri

**Вывод:**



**Код:**

import random

def gen\_random(num\_count, begin, end):

for \_ in range(num\_count):

yield random.randint(begin, end)

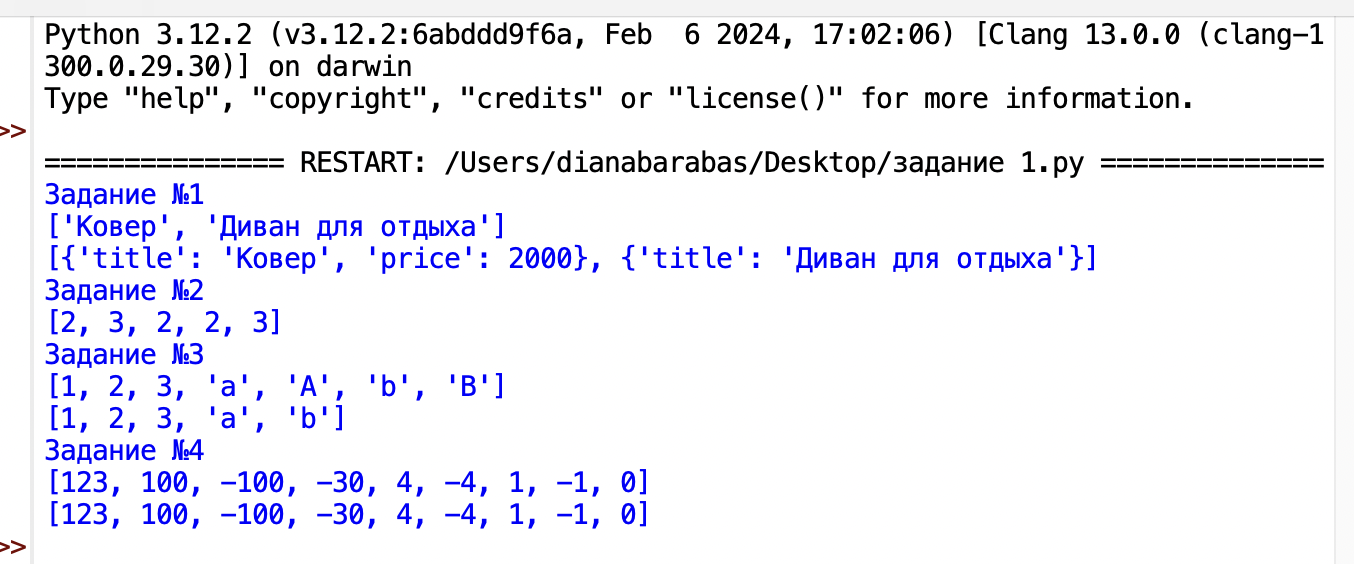
# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

print("Задание №2")

print(list(gen\_random(5, 1, 3))) # Пример вывода: [2, 2, 3, 2, 1]

**Вывод:**



**Задание 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.

**Код:**

class Unique:

def \_\_init\_\_(self, items, \*\*kwargs):

self.ignore\_case = kwargs.get('ignore\_case', False)

self.seen = set()

self.items = iter(items)

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

while True:

item = next(self.items)

check\_item = item.lower() if self.ignore\_case and isinstance(item, str) else item

if check\_item not in self.seen:

self.seen.add(check\_item)

return item

# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

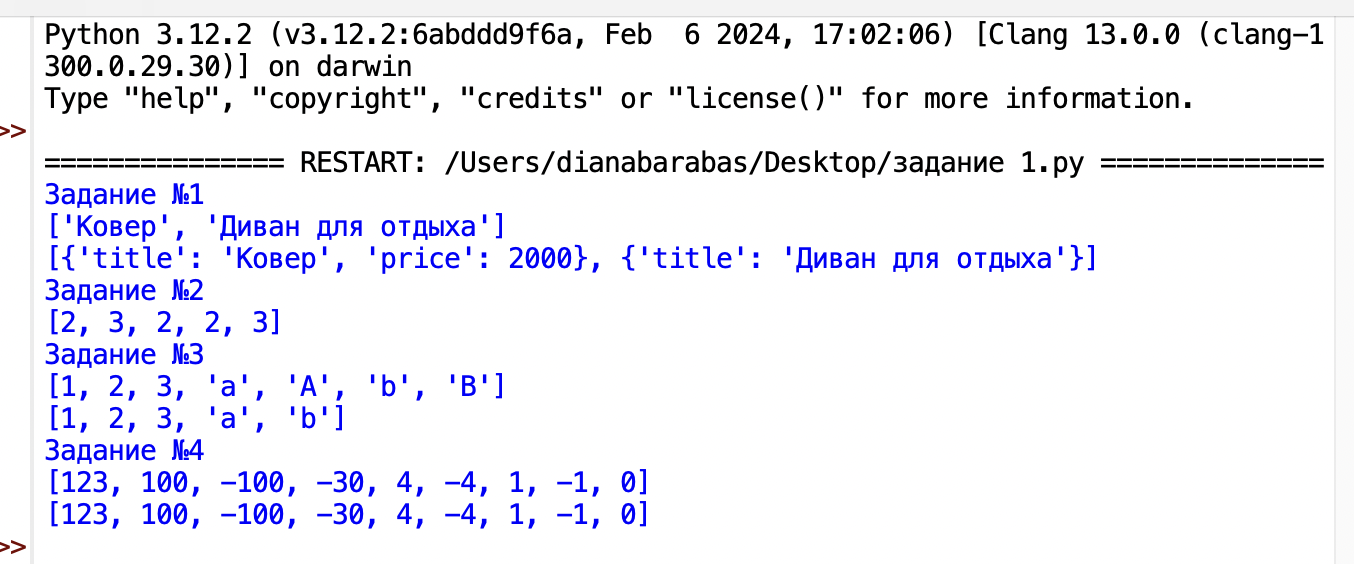
data = [1, 1, 1, 2, 2, 3, 'a', 'A', 'b', 'B']

print("Задание №3")

print(list(Unique(data))) # Вывод: [1, 2, 3, 'a', 'A', 'b', 'B']

print(list(Unique(data, ignore\_case=True))) # Вывод: [1, 2, 3, 'a', 'b']

**Вывод:**



**Задание 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:**

print("Задание №4")

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

if \_\_name\_\_ == '\_\_main\_\_':

# Без использования lambda-функции

result = sorted(data, key=abs, reverse=True)

print(result)

# С использованием lambda-функции

result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

print(result\_with\_lambda)

**Вывод:**

