

Парадигмы и конструкции языков программирования.

ОТЧЕТ **по лабораторной работе №3-4** **«Функциональные возможности языка Python»**

Задача 1 (файл field.py).

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

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.

```
def field(items, *args):
    assert len(args) > 0 # Убедимся, что переданы аргументы

    for item in items:
        if len(args) == 1:
            value = item.get(args[0])
            if value is not None:
                yield value

        else:
            result = {key: item.get(key) for key in args}
            result = {k: v for k, v in result.items() if v is not
None}

            if result:
                yield result
```

Пример использования:

```
goods = [  
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
    {'title': 'Диван для отдыха', 'price': None, 'color': 'black'}  
]  
print(list(field(goods, 'title')))  
print(list(field(goods, 'title', 'price')))
```

```
===== RESTART: /Users/veraleonteva/Desktop/ЛР3-4(1).py =====  
['Ковер', 'Диван для отдыха']  
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]  
>>> |
```

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.

```
import random
```

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

```
# Пример использования генератора  
for number in gen_random(5, 1, 3):  
    print(number)
```

```
===== RESTART: /Users/veraleonteva/Desktop/gen_random.py =====  
1  
2  
3  
1  
3  
>>>
```

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.

```
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

```
data = [1, 1, 1, 2, 2, 3, 'a', 'A', 'b', 'B']
print(list(Unique(data)))
print(list(Unique(data, ignore_case=True)))
```

```
===== RESTART: /Users/veraleonteva/Desktop/unique.py =====
[1, 2, 3, 'a', 'A', 'b', 'B']
[1, 2, 3, 'a', 'b']
>>>
```

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`.

Необходимо решить задачу двумя способами:

1. С использованием `lambda`-функции.
2. Без использования `lambda`-функции.

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

```
# 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)
```

```
===== RESTART: /Users/veraleonteva/Desktop/sort.py =====
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 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.

```

def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(f'Результат работы функции "{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

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'ebm3'

@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()

```

```

===== RESTART: /Users/veraleonteva/Desktop/print_result.py =====
=====
Результат работы функции "test_1":
1
Результат работы функции "test_2":
ebm3
Результат работы функции "test_3":
a = 1
b = 2
Результат работы функции "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.

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`).

```
import time
from contextlib import contextmanager

# Первый контекстный менеджер с использованием класса
class CmTimer1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time:.1f}")

# Второй контекстный менеджер с использованием декоратора
@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    elapsed_time = time.time() - start_time
    print(f"time: {elapsed_time:.1f}")

# Примеры использования
with CmTimer1():
    time.sleep(5.5)

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

```
===== RESTART: /Users/veraleonteva/Desktop/cm_timer.py =====
time: 5.5
time: 5.5
>>>
```

