Московский государственный технический университет им. Н.Э. Баумана

Факультет ИУ
Кафедра ИУ5

Курс «Основы информатики»
Отчет лабораторной работе №3-4

Выполнил студент группы ИУ5-33Б:
Бакушев И.О.
Подпись и дата:

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

Москва, 2024

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 количествово 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.

Текст программы

```
def field(items, *args):
    assert len(args) > 0
    for item in items:
        if len(args) == 1:
            value = item.get(args[0])
            if value is not None:
                yield value
        else:
            result = {}
            all_none = True
            for arg in args:
                value = item.get(arg)
                if value is not None:
                    result[arg] = value
                    all_none = False
            if not all_none:
                yield result


goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'title': None, 'price': 5300, 'color': None}
]

print("field(goods, 'title'):")
for item in field(goods, 'title'):
    print(item)

print("¥nfield(goods, 'title', 'price'):")
for item in field(goods, 'title', 'price'):
    print(item)

print("¥nfield(goods, 'color'):")
for item in field(goods, 'color'):
    print(item)
```

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.

Текст программы

```
import random


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


print("gen_random(5, 1, 3):")
for item in gen_random(5, 1, 3):
    print(item)
```

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.

Текст программы

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

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

```
    def __iter__(self):
        return self


import random

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
print("Unique(data):")
for item in Unique(data):
    print(item)

data = (random.randint(1, 3) for _ in range(10))
print("\nUnique(gen_random(10, 1, 3)):")
for item in Unique(data):
    print(item)

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
print("\nUnique(data):")
for item in Unique(data):
    print(item)

print("\nUnique(data, ignore_case=True):")
for item in Unique(data, ignore_case=True):
    print(item)
```

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.
С использованием lambda-функции.
Без использования lambda-функции.

Текст программы

```
from builtins import sorted

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


def sortArray(data):
    for i in range( len(data)-1):
        for j in range( len(data)-i-1):
            if abs(data[j]) < abs(data[j+1]):
                data[j], data[j+1] = data[j+1], data[j]
    return data

sort = lambda data: sorted(data, key=abs, reverse=True)
```

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 выводить в столбик через знак равенства.

Текст программы

```python
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

@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__':
    test_1()
    test_2()
```

```
    test_3()
    test_4()
```

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).

Текст программы

```
import time
from contextlib import contextmanager


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

    def __exit__(self, exc_type, exc_val, exc_tb):
        end = time.time()
        print(f"time: {end - self.start}")


@contextmanager
def cm_timer_2():
    start = time.time()
    try:
        yield
    finally:
        end = time.time()
        print(f"time: {end - start}")


from time import sleep

with cm_timer_1():
    sleep(5.5)

with cm_timer_2():
    sleep(2.2)
```

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.

Текст программы

```
import json
from unique import Unique
from print_result import print_result
import cm_timer
from gen_random import gen_random
import random


@print_result
def f1(arg):
    return sorted(list(Unique(map(lambda x: x['job-name'].lower(), arg), ignore_case=True)))


@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith('п р о г р а м м и с т'), arg))


@print_result
def f3(arg):
    return list(map(lambda x: x + ' с о п ы т о м Python', arg))
```

```python
@print_result
def f4(arg):
    if not arg:
        return []
    salaries = list(gen_random(len(arg), 100000, 200000))
    result = [f"{specialty}, з а р п л а т а {salary} р у б" for specialty, salary in
zip(arg, salaries)]
    return result
```

Выполнение программы

```
--- unique.py tests ---

Unique(data):
1
2

Unique(gen_random(10, 1, 3)):
2
1
3

Unique(data):
a
A
b
B

Unique(data, ignore_case=True):
a
b
```

```
--- sort.py tests ---

sorted(data, key=abs, reverse=True): [123, 100, -100, -30, 4, -4, 1, -1, 0]
sorted(data, key=lambda x: abs(x), reverse=True): [123, 100, -100, -30, 4, -4, 1, -1, 0]

--- print_result.py tests ---
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

--- cm_timer.py tests ---
cm_timer_1:
time: 1.5014033317565918

cm_timer_2:
time: 0.5004920959472656
```

```
--- process_data.py tests ---
f1
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
агент банка
агент нпф
```

```
 f2
 программист
 программист / senior developer
 программист 1с
 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 1с 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 128378 руб
программист / senior developer с опытом Python, зарплата 165337 руб
программист 1с с опытом Python, зарплата 145816 руб
программист c# с опытом Python, зарплата 110300 руб
программист c++ с опытом Python, зарплата 188613 руб
программист c++/c#/java с опытом Python, зарплата 145530 руб
программист/ junior developer с опытом Python, зарплата 112562 руб
программист/ технический специалист с опытом Python, зарплата 179973 руб
программистр-разработчик  информационных систем с опытом Python, зарплата 153296 руб
time: 0.05349278450012207
```