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 №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-31Б

Шибанова Варвара
Подпись и дата:

Проверил:
преподаватель каф. ИУ5

Подпись и дата:

Москва, 2024 г

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

Задание:

Задание лабораторной работы состоит из решения нескольких задач.
Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. Решение каждой задачи должно раполагаться 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 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.
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
```

Текст программы

```python
def field(items, *args):
    assert len(args) > 0, "Необходимо передать хотя бы одно поле."
    for item in items:
        if len(args) == 1:
            value = item.get(args[0])
            if value:
                yield value
        else:
            result = {}
            for key in args:
                value = item.get(key)
                if value:
                    result[key] = value
            if result:
                yield result


goods = [{'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}]

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
print(list(field(goods, 'title', 'price', 'color')))
```

Вывод результатов:

```
ТЕРМИНАЛ
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py> python field.py
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}]
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py>
```

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 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

Текст программы:

```
from random import randint

def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield randint(begin, end)
print(list(gen_random(7, 1, 20)))
```

Вывод результатов:

```
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py> python gen_random.py
[1, 11, 17, 9, 5, 3, 10]
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py>
```

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
```

Текст программы:

```
import random

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

    def __next__(self):
        for item in self.items:
            comparison_item = item.lower() if self.ignore_case and
isinstance(item, str) else item
            if comparison_item not in self.seen:
                self.seen.add(comparison_item)
                return item
        raise StopIteration
    def __iter__(self):
        return self

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

print('\n')

def gen_random(count, start, end):
    for _ in range(count):
        yield random.randint(start, end)

data = gen_random(10, 1, 3)
for unique_item in Unique(data):
    print(unique_item, end=" ")

print('\n')

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

```python
for unique_item in Unique(data):
    print(unique_item, end=" ")

print('\n')

for unique_item in Unique(data, ignore_case=True):
    print(unique_item, end=" ")
```

Вывод результатов:

```
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py> python unique.py
1 2

3 2 1

a A b B

a b
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py>
```

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.

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:

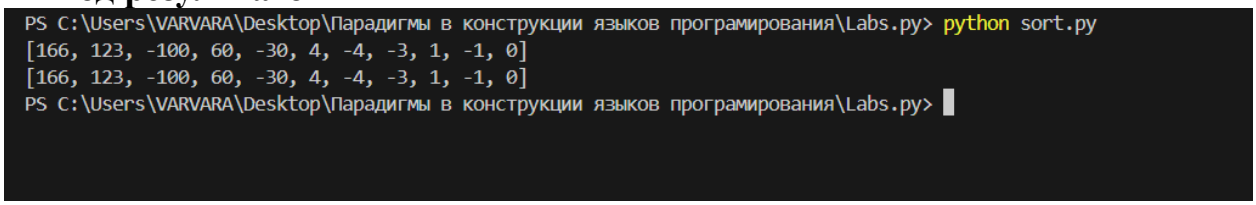
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

```
if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

Текст программы

```python
data = [4, -30, 60, 166, -100, 123, 1, -3, -1, -4, 0]

if __name__ == '__main__':
    print(sorted(data, key=abs, reverse=True))
    print(sorted(data, key=lambda x: abs(x), reverse=True))
```

Вывод результатов



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.

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
```

Текст программы

```python
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        func_name = func.__name__
        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]
```

```python
if __name__ == '__main__':
    test_1()
    test_2()
    test_3()
    test_4()
```

Вывод результатов

```
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py> python print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS C:\Users\VARVARA\Desktop\Парадигмы в конструкции языков програмирования\Labs.py>
```

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.

Текст программы

```
@contextmanager
def cm_timer_2():
    start_time = time.time()
    try:
        yield
    finally:
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"time: {elapsed_time:.2f}")


with cm_timer_2():
    time.sleep(5.2)
```

```
import json
import random
from cm_timer import cm_timer_1
from print_result import print_result


path = 'data_light.json'
with open(path, encoding="utf-8") as f:
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(set(job['job-name'].strip() for job in arg), key=str.casefold)


@print_result
def f2(arg):
    return list(filter(lambda job: job.lower().startswith('программист'), arg))


@print_result
def f3(arg):
    return list(map(lambda job: f"{job} с опытом Python", arg))


@print_result
```

```
def f4(arg):
    salaries = [random.randint(100000, 200000) for _ in arg]
    return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg,
salaries)]


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Вывод результатов

```
ООО Михаэль
Оператор 1С
Оператор Call центра
Оператор call-центра
оператор Call-центра
Оператор call-центра (удаленно)
оператор автоматизированной складской системы
оператор АЗС
Оператор АЗС (РО Башкирия)
Оператор в офис
Оператор газовой котельной
Оператор гребнечесального оборудования
Оператор Колл центра
оператор котельной
Оператор котельной 4 разряда-4 разряда
Оператор котельной установки
```