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 лабораторным работам №4-5
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-32Б
Носков Алексей

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

Москва, 2024 г.

Постановка задачи

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'}`

- ❿ В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количествово аргументов.
- ❿ Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
- ❿ 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
```

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
```

Задача 3 (файл unique.py)

- ❿ Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
- ❿ 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.
- ❿ При реализации необходимо использовать конструкцию **kwargs.
- ❿ Итератор должен поддерживать работу как со списками, так и с генераторами.
- ❿ 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
```

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:

1. С использованием lambda-функции.
2. 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)
```

Задача 5 (файл print_result.py)

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции.

- ⑩ Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- ⑩ Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.
- ⑩ 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
```

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.

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))))
```

Текст программы
field.py

```
goods = [
   {'title': 'Ковер', 'price': 2000, 'color': 'green'},
   {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
]


def field(items, *args):
   assert len(args) > 0


   for d in items:
      if len(args) == 1:
         yield d[args[0]]
      else:
         temp = {}
         for arg in args:
            temp[arg] = d[arg]
         yield temp
```

```
        # yield { d[x] for x in args }


a = field(goods, 'title')
b = field(goods, 'title', 'price')
c = field(goods, 'title', 'price', 'color')
print(list(a))
print(list(b))
print(list(c))
```

unique.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.visited = set()
        self.items = iter(items)
        self.ignore_case = kwargs.get('ignore_case', False)
        self.buff = None

    def __next__(self):
        while True:
            if self.buff is not None:
                item = self.buff
                self.buff = None
            else:
                item = next(self.items)
            # print(next(self.items))

            if isinstance(item, str) and self.ignore_case:
                key = item.lower()
            else:
                key = item

            if key not in self.visited:
                self.visited.add(key)
                return item

    def __iter__(self):
        return self
```

print_result.py

```
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__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

cm_timer.py

```
import time
from contextlib import contextmanager

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

```
    def __exit__(self, exc_type, exc_value, traceback):
        self.end_time = time.time()
        elapsed_time = self.end_time - self.start_time
        print(f"time: {elapsed_time:.2f}")

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

# with cm_timer_1():
#     time.sleep(2)

# with cm_timer_2():
#     time.sleep(2)
```

gen_random.py

```
import random

def gen_random(num_count, begin, end):
    assert num_count > 0
    for x in range(num_count): yield random.randint(begin, end)

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

sort.py

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

if __name__ == "__main__":
    result_with_lambda = sorted(data, key = lambda x: -abs(x))
    print(result_with_lambda)

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

process_data.py

```
import json
import sys
from print_result import print_result
from cm_timer import cm_timer_1
```

```
from unique import Unique
import random
from gen_random import gen_random

path = "data_light.json"

with open(path) as f:
    data = json.load(f)

@print_result
def f1(arg):
    return list(Unique(x['job-name'] for x in data))


@print_result
def f2(arg):
    return list(filter(lambda x: x[:11].lower() == 'программист', arg))


@print_result
def f3(arg):
    return [x + 'с опытом Python' for x in arg]


@print_result
def f4(arg):
    s = f', зарплата {random.randint(100_000, 200_000)} руб.'
    return [x + f', зарплата {random.randint(100_000, 200_000)} руб.' for x in arg]


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

Вывод программы

Задание 1

```
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
```

Задание 2

```
1
2
1
3
```

3

Задание 3 (тестовый код раскомментирован для демонстрации)

1
2

a
A
b
B

a
b

Задание 4

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

Задание 5

!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Задание 6 (тестовый код раскомментирован для демонстрации)

time: 2.00
time: 2.00

Задание 7 (вывод сокращён для наглядности)

1
2
1
3
1
f1
Администратор на телефоне
Медицинская сестра
Охранник сутки-день-ночь-вахта
ВРАЧ АНЕСТЕЗИОЛОГ РЕАНИМАТОЛОГ

теплотехник
разнорабочий
Электро-газосварщик
…
Оператор склада
Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
Заведующий музеем в д.Копорье
Документовед
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
Инженер-программист
Менеджер (в промышленности)
f2
Программист
Программист C++/C#/Java
программист
Программист 1С
Программистр-разработчик информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
программист 1С
Программист C#
f3
Программистс опытом Python
Программист C++/C#/Javac опытом Python
программистс опытом Python
Программист 1Сс опытом Python
Программистр-разработчик информационных системс опытом Python
Программист C++с опытом Python
Программист/ Junior Developerс опытом Python
Программист / Senior Developerс опытом Python
Программист/ технический специалистс опытом Python
программист 1Сс опытом Python
Программист C#с опытом Python
f4
Программистс опытом Python, зарплата 112026 руб.
Программист C++/C#/Javac опытом Python, зарплата 176535 руб.
программистс опытом Python, зарплата 161130 руб.
Программист 1Сс опытом Python, зарплата 150801 руб.
Программистр-разработчик информационных системс опытом Python, зарплата 188353 руб.
Программист C++с опытом Python, зарплата 161591 руб.

Программист/ Junior Developerc опытом Python, зарплата 145343 руб.
Программист / Senior Developerc опытом Python, зарплата 159081 руб.
Программист/ технический специалистс опытом Python, зарплата 157286 руб.
программист 1Cc опытом Python, зарплата 168483 руб.
Программист C#c опытом Python, зарплата 175342 руб.
time: 0.02