Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ЛАБОРАТОРНАЯ РАБОТА

№ 3

Функциональные возможности языка Python

Группа ИУ5-35Б

Студент _____ /_____/_____Александров А.В_____/

(Подпись, дата) (И.О.Фамилия)

Преподаватель _____/_____/_____/

(Подпись, дата) (И.О.Фамилия)

2024

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 располагаться 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.
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.

```
#1 ЗАДАНИЕ
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 # проверка на истинность, иначе AssertionError

    if len(args) == 1:
        for i in items:
            result = i[args[0]]
            if result != None:
                yield result
    else:
        for item in items:
            result = {i:item[i] for i in args if item[i] != None} #создаем словарь, если значение не None
            yield result

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

```
{'title': 'Ковер', 'price': 2000} {'title': 'Диван для отдыха', 'price': 5300}
```

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

```
#2 ЗАДАНИЕ
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
def gen_random(num_count, begin, end):
    import random
    for i in range(num_count):
        yield random.randint(begin, end)
for num in gen_random(5, 1, 3):
    print(num)
```

```
3
3
3
3
1
```

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.

```
#3 ЗАДАНИЕ
# 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
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
```

```
        self.items = items
        self.ignore = kwargs.get("ignore_case", False)
        self.Set = set() #для уникальных значений
    def __next__(self):
        for item in self.items:
            to_low = item.lower() if self.ignore and type(item) == str else item
            if to_low not in self.Set:
                self.Set.add(to_low)
                return item
        raise StopIteration #для генерации исключений
    def __iter__(self):
        return self

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
unique_data1 = Unique(data1)
print(list(unique_data1))

data2 = (x for x in gen_random(10, 1, 3))
unique_data2 = Unique(data2)
print(list(unique_data2))

data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
unique_data3 = Unique(data3)
print(list(unique_data3))

unique_data4 = Unique(data3, ignore_case=True)
print(list(unique_data4))
```

```
[1, 2]
[3, 1]
['a', 'A', 'b', 'B']
['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:
С использованием lambda-функции.
Без использования lambda-функции.

```
#4 ЗАДАНИЕ
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
if __name__ == '__main__':
    result = sorted(data, key = abs, reverse = True)
    print(result)

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

```
print(result_with_lambda)
```

```
[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 выводить в столбик через знак равенства.

```
#5 ЗАДАНИЕ
def print_result(func):
    def wrapper(*args, **kwargs):#при добавлении декоратора сначала вызывается wrapper
        result = func(*args, **kwargs)#вызов функции после декоратора
        print(func.__name__)
        if type(result) == list:
            for item in result:
                print(item)
        elif type(result) == dict:
            for k, v in result.items():
                print(k, " = ", v)
        else:
            print(result)
        return result #возвращаем результат
    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()
```

```
!!!!!!!!
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).

```
#6 ЗАДАНИЕ
#1 cm_timer_1 на примере класса
import time
from contextlib import contextmanager
class cm_timer_1:
    def __enter__(self): #дейтсвия до основного кода
        self.start_time = time.time() #сохраняем время начала
        return self
    def __exit__(self, exc_type, exc_value, traceback): #после выполнения with
        #exc_type - исключение внутри with
        #exc_value - сообщение об ошибке
        #traceback показывает, где произошла ошибка
        print(time.time() - self.start_time, "- время выполнения cm_timer_1")
if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(5.5)

@contextmanager
def cm_timer_2():
    start_time = time.time()
```

```
    try:
        yield #до кода выполняется, после передает управление with
    finally: #после with возвращаемся в finally
        print(time.time() - start_time, "- время выполнения cm_timer_2")
if __name__ == "__main__":
    with cm_timer_2():
        time.sleep(5.5)
```

```
5.500940799713135 - время выполнения cm_timer_1
5.501137971878052 - время выполнения cm_timer_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.

```
#7 ЗАДАНИЕ
import json
import sys

path = r"D:\Users\79169\Загрузки\data_light.json"
with open(path, 'r', encoding='utf-8') as f:
    data = json.load(f)

@print_result
```

```
def f1(arg):
    return sorted(Unique(field(arg, 'job-name')))

@print_result
def f2(arg):
    return list(filter(lambda x: True if x.lower().startswith('программист') else False, arg))

@print_result
def f3(arg):
    return list(map(lambda x: x + ' с опытом Python', arg))

@print_result
def f4(arg):
    salary = gen_random(len(arg), 100000, 200000)
    return [i + ', зарплата {} руб.'.format(j) for i, j in zip(arg, salary)]

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

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
Web-разработчик
[химик-эксперт
web-разработчик
```

```
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
программист
программист 1С
```

```
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
программист с опытом Python
программист 1С с опытом Python
-
f4
Программист с опытом Python, зарплата 122127 руб.
Программист / Senior Developer с опытом Python, зарплата 177943 руб.
Программист 1С с опытом Python, зарплата 121358 руб.
Программист C# с опытом Python, зарплата 114170 руб.
Программист C++ с опытом Python, зарплата 139865 руб.
Программист C++/C#/Java с опытом Python, зарплата 114893 руб.
Программист/ Junior Developer с опытом Python, зарплата 194561 руб.
Программист/ технический специалист с опытом Python, зарплата 174825 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 146194 руб.
программист с опытом Python, зарплата 179438 руб.
программист 1С с опытом Python, зарплата 116904 руб.
0.010000944137573242 - время выполнения cm_timer_1
```