|  |  |  |
| --- | --- | --- |
| Защищено:  Гапанюк Ю.Е.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2023** г. |  | Демонстрация:  Тихонова Д.Д.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2023** г. |

**Отчет по лабораторной работе № 2 по курсу**

**Парадигмы и конструкции языков программирования**

#### Тема работы: " Функциональные возможности языка Python. "

14

(количество листов)

Вариант № **24**

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-54Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Тихонова Д.Д. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2023**  г. |

Москва, МГТУ - **2023**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

[1. Описание задания 3](#_Toc152969029)

# Описание задания

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:

orders = [

        {'item': 'Лампа', 'quantity': 2, 'price': 300},

        {'item': 'Стол', 'quantity': 1, 'price': 1500},

        {'item': 'Книга', 'quantity': 3, 'price': 20}

    ]

field(orders, 'item') должен выдавать 'Лампа', 'Стол', 'Книга'

field(orders, 'item', 'price') должен выдавать {'item': 'Лампа', 'price': 300}, {'item': 'Стол'}

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.

Шаблон для реализации генератора:

 # Извлекаем и выводим количество каждого товара в заказах

    for quantity in field(orders, 'quantity'):

        print(f"Количество товара: {quantity}")

    # Результат: 2, 1, 3

    # Извлекаем и выводим информацию о каждом товаре в заказах (наименование и цена)

    for product\_info in field(orders, 'item', 'price'):

        print(f"Товар: {product\_info['item']}, Цена: {product\_info['price']} руб.")

    # Результат: {'item': 'Лампа', 'price': 300},

    #           {'item': 'Стол', 'price': 1500},

    #           {'item': 'Книга', 'price': 20}

def field(items, \*args):

assert len(args) > 0

# Необходимо реализовать генератор

**Листинг:**

def field(orders, \*args):

    assert len(args) > 0

    for order in orders:

        # Если передан только один аргумент, выдаем значения полей товаров

        if len(args) == 1:

            product\_name = args[0]

            quantity = order.get(product\_name)

            if quantity is not None:

                yield quantity

        else:

            # Если передано несколько аргументов, выдаем словари с информацией о товарах

            result\_dict = {}

            has\_quantity = False

            for product\_name in args:

                quantity = order.get(product\_name)

                if quantity is not None:

                    result\_dict[product\_name] = quantity

                    has\_quantity = True

            if has\_quantity:

                yield result\_dict

if \_\_name\_\_ == '\_\_main\_\_':

    # Пример использования для списка заказов:

    orders = [

        {'item': 'Лампа', 'quantity': 2, 'price': 300},

        {'item': 'Стол', 'quantity': 1, 'price': 1500},

        {'item': 'Книга', 'quantity': 3, 'price': 20}

    ]

    # Извлекаем и выводим количество каждого товара в заказах

    for quantity in field(orders, 'quantity'):

        print(f"Количество товара: {quantity}")

    # Результат: 2, 1, 3

    # Извлекаем и выводим информацию о каждом товаре в заказах (наименование и цена)

    for product\_info in field(orders, 'item', 'price'):

        print(f"Товар: {product\_info['item']}, Цена: {product\_info['price']} руб.")

    # Результат: {'item': 'Лампа', 'price': 300},

    #           {'item': 'Стол', 'price': 1500},

    #           {'item': 'Книга', 'price': 20}

**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

**Листинг:**

import random

def gen\_random(num\_count, begin, end):

    for \_ in range(num\_count):

        yield random.randint(begin, end)

if \_\_name\_\_ == '\_\_main\_\_':

    # Пример использования:

    random\_numbers = list(gen\_random(5, 1, 3))

    print(random\_numbers)

**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, Aбв и АБВ - разные строки

# ignore\_case = False, Aбв 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

**Листинг:**

from lab2.lab\_python\_fp.gen\_random import gen\_random

class Unique(object):

    def \_\_init\_\_(self, items, \*\*kwargs):

        self.items = iter(items)

        self.unique\_set = set()

        self.ignore\_case = kwargs.get('ignore\_case', False)

    def \_\_next\_\_(self):

        while True:

            item = next(self.items)

            key = item.lower() if self.ignore\_case and isinstance(item, str) else item

            if key not in self.unique\_set:

                self.unique\_set.add(key)

                return item

    def \_\_iter\_\_(self):

        return self

if \_\_name\_\_ == '\_\_main\_\_':

    # Пример использования:

    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

    unique\_iter1 = Unique(data1)

    print("data1: "+str(list(unique\_iter1)))  # Вывод: [1, 2]

    data2 = gen\_random(10, 1, 3)

    unique\_iter2 = Unique(data2)

    print("data1: "+str(list(unique\_iter2)))  # Вывод: Случайные числа без дубликатов

    data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    unique\_iter3 = Unique(data3)

    print("data3: "+str(list(unique\_iter3))) # Вывод: ['a', 'A', 'b', 'B']

    unique\_iter4 = Unique(data3, ignore\_case=True)

    print("data4: "+str(list(unique\_iter4)))  # Вывод: ['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. 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)

**Листинг:**

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

if \_\_name\_\_ == '\_\_main\_\_':

    # Способ 1: С использованием lambda-функции

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

    print(result)

    # Способ 2: Без использования lambda-функции

    result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

    print(result\_with\_lambda)

**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

Листинг:  
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()

**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).

**Листинг:**

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}")

@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}")

if \_\_name\_\_ == '\_\_main\_\_':

    with cm\_timer\_1():

        time.sleep(5.5)

    with cm\_timer\_2():

        time.sleep(5.5)

**Задача 7. (файл process\_data.py)**

Применим их на реальном примере инструменты из предыдущих задач.

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))))

**Листинг:**

import json

import sys

from cm\_timer import cm\_timer\_1

from print\_result import print\_result

from gen\_random import gen\_random

from field import field

from unique import Unique

path = sys.argv[1] if len(sys.argv) > 1 else "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'), ignore\_case=True))

@print\_result

def f2(arg):

    return list(filter(lambda x: x.lower().startswith("программист"), arg))

@print\_result

def f3(arg):

    return list(map(lambda x: x + ", с опытом Python", arg))

@print\_result

def f4(arg):

    salaries = list(gen\_random(len(arg), 100000, 200000))

    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))))