|  |  |  |
| --- | --- | --- |
| Защищено:    Шаповалов А.А.      "\_\_"\_\_\_\_\_\_\_\_\_\_\_2022 г. |  | Демонстрация:  Шаповалов А.А.      "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

**Отчет по лабораторной работе № 3-4 по курсу**

**Базовые компоненты интернет-технологий**

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

< # >

 (количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-52Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Шаповалов Архип Аркадьевич | "28"\_\_января \_\_2022  г. |

Москва, МГТУ   -  2022

**1. Цель лабораторной работы:**

Изучение возможностей функционального программирования в языке Python.

**2. Индивидуальное задание номер 1**

**Формулировка:**

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

### 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)

* 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

### 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.

* 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

### 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](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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 программы:**

### **field.py**

from \_\_future\_\_ import annotations  
  
  
def field(items: list[dict] | tuple[dict], \*args: str):  
 assert len(args) > 0, \  
 "\n[ArgumentError] -> No keys specified to print!"  
  
 # assert len(set(map(lambda d: tuple(d.keys()), (d for d in items)))) == 1, \  
 # "\n[ArgumentError] -> Goods list have dicts with different keys"  
  
 assert tuple(arg for arg in args if arg in items[0].keys()) == args, \  
 "\n[ArgumentError] -> Selected keys are not in goods list!"  
  
 if len(args) == 1:  
 for product in items:  
 yield product[args[0]]  
 else:  
 for product in items:  
 yield {arg: product[arg] for arg in args}  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
 ]  
  
 print("Testing...")  
  
 # Testing  
 assert tuple(i for i in field(goods, 'title')) == ('Ковер', 'Диван для отдыха'), \  
 "\n[TestError][#1] -> -Failed-"  
 assert tuple(i for i in field(goods, 'title', 'price')) == ({'title': 'Ковер', 'price': 2000},  
 {'title': 'Диван для отдыха', 'price': 5300}), \  
 "\n[TestError][#2] -> -Failed-"  
  
 print("All tests passed!\n")  
  
 print("Run with args ('title, color'):")  
 # Run  
 for callback in field(goods, 'title', "color"):  
 print(callback)

### **gen\_random.py**

from random import randint  
  
  
def gen\_random(num\_count: int, begin: int, end: int):  
 assert begin < end, \  
 "\n[ArgumentError] -> BEGIN > END"  
 for \_ in range(num\_count):  
 yield randint(begin, end)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Run with args ('10, 5, 30'):")  
 # Run  
 for callback in gen\_random(10, 5, 30):  
 print(callback)

### **unique.py**

from types import GeneratorType  
  
  
class Unique(object):  
 def \_\_init\_\_(self, data: tuple or list or GeneratorType, ignore\_case: bool = False):  
 # С учетом, что указано, что это массив, такое определение типа корректно  
 if isinstance(data, GeneratorType):  
 data = list(i for i in data)  
  
 self.\_\_cache = []  
 self.\_\_ignore\_case = ignore\_case  
 self.\_\_len = len(data)  
  
 if type(data[0]) is str and ignore\_case:  
 self.\_\_data = [char.lower() for char in data]  
 return  
  
 self.\_\_data = data  
  
 def \_\_next\_\_(self):  
 if self.\_\_pos >= self.\_\_len:  
 raise StopIteration  
  
 while self.\_\_data[self.\_\_pos] in self.\_\_cache:  
 self.\_\_pos += 1  
 if self.\_\_pos >= self.\_\_len:  
 raise StopIteration  
  
 self.\_\_cache.append(self.\_\_data[self.\_\_pos])  
 self.\_\_pos += 1  
  
 return self.\_\_data[self.\_\_pos - 1]  
  
 def \_\_iter\_\_(self):  
 self.\_\_pos = 0  
 return self  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Testing...")  
  
 # Testing  
 assert tuple(i for i in Unique([1, 1, 1, 1, 1, 2, 2, 2, 2, 2])) == (1, 2), \  
 "\n[TestError][#1] -> -Failed-"  
 assert tuple(i for i in Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'])) == ('a', 'A', 'b', 'B'), \  
 "\n[TestError][#2] -> -Failed-"  
 assert tuple(i for i in Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'],  
 ignore\_case=True)) == ('a', 'b'), \  
 "\n[TestError][#3] -> -Failed-"  
  
 print("All tests passed!\n")  
  
 print("Run with args ('title, color'):")  
 # Run  
 for callback in Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'],  
 ignore\_case=True):  
 print(callback)

### **sort.py**

from \_\_future\_\_ import annotations  
  
  
def sort(array: list | tuple, is\_lambda=False):  
 if is\_lambda:  
 return tuple(sorted(array, key=lambda i: -abs(i))) # Одна строка с анонимной функцией  
 return tuple(sorted(array, key=abs, reverse=True)) # Одна строка без анонимной функции  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("Testing...")  
  
 # Testing  
 assert sort((4, -30, 30, 100, -100, 123, 1, 0, -1, -4)) == (123, 100, -100, -30, 30, 4, -4, 1, -1, 0), \  
 "\n[TestError][#1] -> -Failed-"  
 assert sort((4, -30, 30, 100, -100, 123, 1, 0, -1, -4),  
 is\_lambda=True) == (123, 100, -100, -30, 30, 4, -4, 1, -1, 0), \  
 "\n[TestError][#2] -> -Failed-"  
  
 print("All tests passed!\n")  
  
 print("Run with array (4, -30, 30, 100, -100, 123, 1, 0, -1, -4):")  
 print(sort((4, -30, 30, 100, -100, 123, 1, 0, -1, -4)))

### **print\_result.py**

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 print(func.\_\_name\_\_)  
  
 result = func(\*args, \*kwargs)  
  
 r\_type = type(result)  
 if r\_type is not dict and r\_type is not list and r\_type is not tuple:  
 print(result)  
 return result  
  
 if r\_type is list or r\_type is tuple:  
 print(\*result, sep='\n')  
 return result  
  
 for key in result:  
 print(f'{key} = {result[key]}')  
 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**

from time import time as now  
from time import sleep  
from contextlib import contextmanager  
  
  
class cm\_timer\_1():  
 def \_\_enter\_\_(self):  
 self.start = now()  
  
 def \_\_exit\_\_(self, \*args, \*\*kwargs):  
 print(f"Time: {now() - self.start}")  
  
  
@contextmanager  
def cm\_timer\_2():  
 start = now()  
 yield  
 print(f"Time: {now() - start}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 sleep(2.4)  
 with cm\_timer\_2():  
 sleep(2.5)

### **process\_data.py**

from sys import argv  
from json import loads as jloads  
from field import field  
from gen\_random import gen\_random  
from unique import Unique  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
  
path = 'data\_light.json'  
  
with open(path, encoding="utf-8") as f:  
 data = jloads(f.read())  
  
  
def f1(arg):  
 return sorted(Unique([pos for pos in field(arg, "job-name")], ignore\_case=True))  
  
  
def f2(arg):  
 return filter(lambda s: s.startswith("программист"), arg)  
  
  
def f3(arg):  
 return list(map(lambda s: s + " c опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 \_zip = zip(arg, gen\_random(len(arg), 100000, 200000))  
 \_str = [f"<{a:^70}> {b:^8} руб." for a, b in \_zip]  
 return \_str  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Результат работы программы:**

**field.py**

Testing...

All tests passed!

Run with args ('title, color'):

{'title': 'Ковер', 'color': 'green'}

{'title': 'Диван для отдыха', 'color': 'black'}

Process finished with exit code 0

### **gen\_random.py**

Run with args ('10, 5, 30'):

15

17

9

17

6

26

9

19

20

26

### **unique.py**

Testing...

All tests passed!

Run with args ('title, color'):

a

b

### **sort.py**

Testing...

All tests passed!

Run with array (4, -30, 30, 100, -100, 123, 1, 0, -1, -4):

(123, 100, -100, -30, 30, 4, -4, 1, -1, 0)

### **print\_result.py**

!!!!!!!!

test\_1

1

test\_2

iu5

test\_3

a = 1

b = 2

test\_4

1

2

**cm\_timer.py**

Time: 2.405076026916504

Time: 2.503603935241699

### **process\_data.py**

f4

< программист c опытом Python > 181201 руб.

< программист / senior developer c опытом Python > 103331 руб.

< программист 1с c опытом Python > 187769 руб.

< программист c# c опытом Python > 138700 руб.

< программист с++ c опытом Python > 138821 руб.

< программист с++/с#/java c опытом Python > 180617 руб.

< программист/ junior developer c опытом Python > 192569 руб.

< программист/ технический специалист c опытом Python > 160784 руб.

< программистр-разработчик информационных систем c опытом Python > 100961 руб.

Time: 0.03844094276428223