**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 №3-4

«**Функциональные возможности языка Python.**»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Баринов А. А. |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2023 г.

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.

Текст программы

def field(items, \*args):

assert len(args) > 0

if len(args) == 1:

for item in items:

value = item.get(args[0])

if value is not None:

yield value

else:

for item in items:

filtered\_item = {key: item[key] for key in args if item.get(key) is not None}

if filtered\_item:

yield filtered\_item

# Пример использования:

goods = [

{'title': 'Ковер', 'price': 2000, 'color': 'green'},

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

]

for value in field(goods, 'title'):

print(value)

for item in field(goods, 'title', 'price'):

print(item)

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.

Текст программы

import random

def gen\_random1(num\_count, begin, end):

for x in range(num\_count):

yield random.randint(begin, end)

# Пример использования:

random\_numbers = list(gen\_random1(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.

Текст программы

from gen\_random import \*

class Unique(object):

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

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

self.seen = set()

if self.ignore\_case:

self.items = (str(item).lower() for item in items)

else:

self.items = iter(items)

def \_\_next\_\_(self):

while True:

item = next(self.items)

key = item.lower() if self.ignore\_case else item

if key not in self.seen:

self.seen.add(key)

return item

def \_\_iter\_\_(self):

return self

# Примеры использования:

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

unique1 = Unique(data1)

print(list(unique1))

data2 = gen\_random1(10, 1, 3)

unique2 = Unique(data2)

print(list(unique2))

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

unique3 = Unique(data3)

print(list(unique3))

unique4 = Unique(data3, ignore\_case=True)

print(list(unique4))

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.

Текст программы

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

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

# С использованием lambda-функции

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

print(result)

# Без использования lambda-функции

result\_without\_lambda = sorted(data, key=abs, reverse=True)

print(result\_without\_lambda)

Задача 5 (файл print\_result.py)

Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

Текст программы

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

# Пример использования 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()

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.

Текст программы

from contextlib import contextmanager

import time

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}')

# Пример использования

with cm\_timer\_1():

time.sleep(5.5)

with cm\_timer\_2():

time.sleep(5.5)

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

from field import \*

from gen\_random import \*

from unique import \*

from print\_result import \*

from sort import \*

from cm\_timer import \*

import json

with open('data\_light.json', 'r', encoding='utf-8') as dat:

data = json.load(dat)

@print\_result

def f1(arg):

return sorted(list(Unique([j['job-name'] for j in arg], 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: f"{x}, с опытом Python", arg))

@print\_result

def f4(arg):

salary = gen\_random1(len(arg), 100000, 200000)

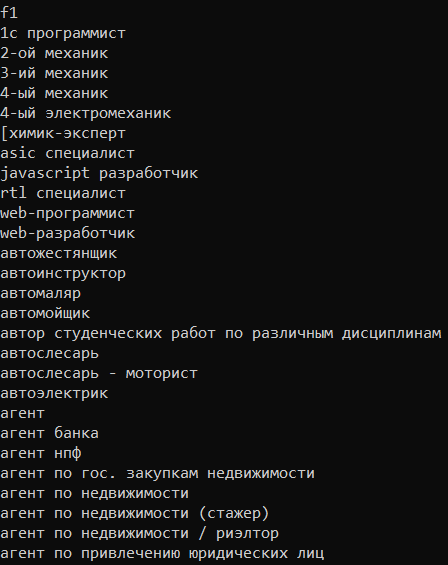
return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg, salary)]

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

with cm\_timer\_1():

f4(f3(f2(f1(data))))

Результаты выполнения



И т. д.

