**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

## «Функциональные возможности языка Python.»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33 |  | преподаватель каф. ИУ5 |
| Ефременко Даниил |  | Канев Антон. |
| Подпись и дата: |  | Подпись и дата: |

**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 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 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.

**Текст программы**

def field(items, \*args):  
 assert len(args) > 0  
 if len(args) == 1:  
 for dictionary in items:  
 note = dictionary.get(args[0])  
 if note is not None:  
 yield note  
 else:  
 for d in items:  
 dictionary = dict()  
 for arg in args:  
 note = d.get(arg)  
 if note is not None:  
 dictionary[arg] = note  
 if len(dictionary) != 0:  
 yield dictionary  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 goods = [  
 {'title': 'Ковер', 'price': 5000, 'color': 'red'},  
 {'title': 'Диван для отдыха', 'price': 10000, 'color': 'white'},  
 {'title': None, 'price': None, 'color': 'black'},  
 {'title': 'Кровать', 'price': 15000, 'color': 'yellow'}  
 ]  
 data1 = list()  
 data2 = list()  
  
 for i in field(goods, 'title'):  
 data1.append(i)  
 print(str(data1))  
  
  
 for i in field(goods, 'title', 'price','color'):  
 data2.append(i)  
 print(data2)

**Экранные формы с примерами выполнения программ**

**C:\Users\User\Desktop\Безымянный.png**

**C:\Users\User\Desktop\Безымянный.png**

### **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\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 number = random.randrange(begin, end)  
 yield number  
 #print(number)  
 pass  
  
def main():  
 data = list()  
 for i in gen\_random(5, 1, 3):  
 data.append(i)  
 print(data)  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Экранные формы с примерами выполнения программ**

**C:\Users\User\Desktop\Безымянный.png**

### **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.

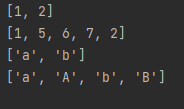
Итератор должен поддерживать работу как со списками, так и с генераторами.

Итератор не должен модифицировать возвращаемые значения.

**Текст программы**

from gen\_random import gen\_random  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.used\_elements = set()#пустое множество  
 self.data = items  
 self.ignore\_case = False  
 if len(kwargs) > 0:  
 self.ignore\_case = kwargs['ignore\_case']  
  
 def \_\_next\_\_(self):  
 it = iter(self.data)  
 while True:  
 try:  
 current = next(it)  
 except StopIteration:  
 raise StopIteration  
 else:  
 if self.ignore\_case is True and isinstance(current, str):  
 current = current.lower()  
 if current not in self.used\_elements:  
 self.used\_elements.add(current)  
 return current  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 data2 = gen\_random(5, 1, 10)  
 data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
  
 print(list(Unique(data1)))  
 print(list(Unique(data2)))  
 print(list(Unique(data3, ignore\_case=True)))  
 print(list(Unique(data3, ignore\_case=False)))

**Экранные формы с примерами выполнения программ**



### **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:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

**Текст программы**

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 result = sorted(data, reverse=True, key=abs)  
 print(result)  
  
 result\_with\_lambda = sorted(data, reverse=True, key=lambda x: abs(x))  
 print(result\_with\_lambda)

**Экранные формы с примерами выполнения программ**

C:\Users\User\Desktop\Безымянный.png

### **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.

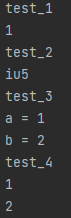
Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

**Текст программы**

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
 if isinstance(result, list):  
 for i in result:  
 print(i)  
 elif isinstance(result, dict):  
 for i in result:  
 print(str(i) + " = " + str(result[i]))  
 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\_\_':  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

**Экранные формы с примерами выполнения программ**



### **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.

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).

**Текст программы**

from contextlib import contextmanager  
import time  
  
@contextmanager  
def cm\_timer\_1():  
 start = time.perf\_counter()  
 yield  
 print("Время работы блока кода: {} секунд".format(time.perf\_counter() - start))  
  
class cm\_timer\_2:  
  
 def \_\_init\_\_(self):  
 self.start = time.perf\_counter()  
  
 def \_\_enter\_\_(self):  
 self.start = time.perf\_counter()  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 if exp\_type is not None:  
 print(exp\_type, exp\_value, traceback)  
 else:  
 print("Время работы блока кода: {} секунд".format(time.perf\_counter() - self.start))  
  
with cm\_timer\_1():  
 time.sleep(2)  
with cm\_timer\_2():  
 time.sleep(2)

**Экранные формы с примерами выполнения программ**

C:\Users\User\Desktop\Безымянный.png

### **Задача 7 (файл process\_data.py)**

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.

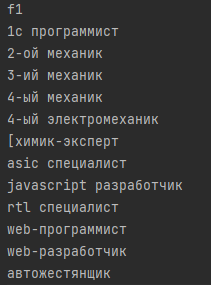
Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию map.

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.

**Текст программы**

import json  
import sys  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
from unique import Unique  
from field import field  
from gen\_random import gen\_random  
  
with open('data\_light.json', encoding='utf-8') as file:  
 data = json.load(file)  
  
@print\_result  
def f1(arg):  
 return sorted(list(Unique(field(arg, 'job-name'), ignore\_case=True)), key=str.lower)  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda string: str.startswith(str.lower(string), 'программист'), arg))  
  
@print\_result  
def f3(arg):  
 return list(map(lambda string: string + " с опытом Python", arg))  
  
@print\_result  
def f4(arg):  
 #объединение элементов из нескольких источников данных.  
 return dict(zip(arg, list('зарплата {} руб.'.format(val) for val in gen\_random(len(arg), 1000000, 2000000))))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Экранные формы с примерами выполнения программ**



… и еще очень много профессий

