 **Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления» Кафедра «Системы обработки информации и управления»**

Лабораторные работа №3-4

**«Функциональные возможности языка Python»**

по предмету

«Базовые компоненты интернет-технологий»

Выполнил:

студент группы № ИУ5-31Б

Целуйко Ульяна

Проверил:

Преподаватель кафедры ИУ-5

Гапанюк Юрий

2022 г.

**Задание:**

1. Задание лабораторной работы состоит из решения нескольких задач.
2. Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно располагаться в отдельном файле.
3. При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

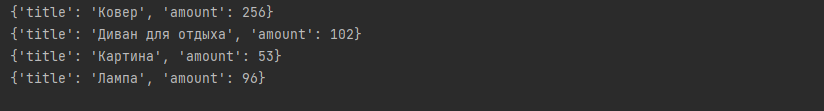
**Задание 1:**

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.

**field.py**

goods = [  
 {"title": "Ковер", "color": "green", "amount": 256},  
 {"title": "Диван для отдыха", "price": 5300, "color": "black", "amount":102},  
 {"title": "Картина", "price": 3120, "color": "white", "amount": 53},  
 {"title": "Лампа", "price": 550, "color": "blue", "amount": 96},  
]  
  
def field(items, \*args):  
 assert len(args) > 0  
 r = [{} for i in range(len(items))]  
 for i in range(len(items)):  
 for key in items[i]:  
 if key in args:  
 r[i].update({key:items[i][key]})  
   
 return r  
  
if \_\_name\_\_=="\_\_main\_\_":  
 for g in field(goods, "title", "amount"):  
 print(g)

****

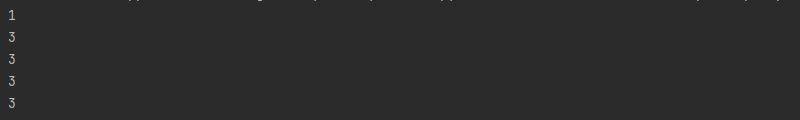
**Задание 2**

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

**gen\_random.py**

mport random  
  
def gen\_random(count, begin, end):  
 for \_ in range(count):  
 yield random.randint(begin, end)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 gen\_f = gen\_random(5, 1, 3)  
 for i in gen\_f:  
 print(i)



**Задание 3**

* 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 №1

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

**Unique(data)** будет последовательно возвращать только 1 и 2.

Пример №2

data = gen\_random(10, 1, 3)

**Unique(data)** будет последовательно возвращать только 1, 2 и 3.

Пример №3

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

Пример №4

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

**unique.py**

class Unique(object):  
 def \_\_init\_\_(self, items, ignore\_case=False):  
 self.r = []  
  
 if ignore\_case:  
 items = [i.lower() for i in items]  
  
 for i in items:  
 if i not in self.r:  
 self.r.append(i)  
  
 def \_\_next\_\_(self):  
 if self.begin < len(self.r):  
 x = self.r[self.begin]  
 self.begin += 1  
 return x  
 else:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 self.begin = 0  
 return self  
  
def print\_test(data):  
 print("-" \* 50)  
 for i in data:  
 print(i)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data = [1, 4, 87, 3, 5, 7, 2, 4, 6, 4, 3, 6, 3, 4, 2]  
 test = Unique(data)  
 print\_test(test)  
  
 data = ['A', 'a', 'b', 'B', 'a', 'A', 'b', 'B']  
 test = Unique(data)  
 print\_test(test)  
  
 test = Unique(data, ignore\_case=True)  
 print\_test(test)



**Задание 4**

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.

**sort.py**

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 a: -abs(a))  
 print(result\_with\_lambda)



**Задание 5**

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.

**print\_result.py**

def print\_result(f):  
 def wrapper(\*args, \*\*kwargs):  
 print(f.\_\_name\_\_)  
  
 res = f(\*args, \*\*kwargs)  
 if type(res) == list:  
 for i in res:   
 print(i)  
 elif type(res) == dict:  
 for k,v in res.items():  
 print(k, '=', v)  
 else:  
 print(res)  
  
 print()  
  
 return res  
  
 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]  
  
  
@print\_result  
def test\_5(a, b):  
 return a + b  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
 test\_5(10, 15)

****

**Задание 6**

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).

**cm\_timer.py**

import time  
from contextlib import contextmanager  
  
class cm\_timer\_1():  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
   
 def \_\_exit\_\_(self, type, value, traceback):  
 print("time:", time.time() - self.start\_time)  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield   
 print("time:", time.time() - start\_time)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 time.sleep(1.9)  
 with cm\_timer\_2():  
 i = 10  
 while (i < 10000000):  
 i += 1

**Задание 7**

* 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.

**process\_data.py**

import json  
import os  
from operator import concat  
from unique import Unique  
from field import field  
from gen\_random import gen\_random  
from cm\_timer import cm\_timer\_1  
  
path = os.path.dirname(\_\_file\_\_) + "\\data\_light.json"  
  
with open(path, encoding="utf-8") as f:  
 data = json.loads(f.read())  
  
def f1(arg):  
 return Unique([i["job-name"] for i in field(arg, "job-name")], ignore\_case=True)  
  
def f2(arg):  
 return filter(lambda a: a.startswith("программист"), arg)  
  
def f3(arg):  
 return list(map(lambda x: concat(x, " c опытом Python"), arg))  
  
def f4(arg):  
 \_zip = zip(arg, gen\_random(len(arg), 100000, 200000))  
 \_str = [f"{a}, зарплата {b} руб." for a, b in \_zip]  
 return \_str  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 for i in f4(f3(f2(f1(data)))):  
 print("'" + i + "'")

