**Московский государственный технический университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе № 3

«Функциональные возможности языка Python»

Выполнил: Проверил:

студент группы ИУ5-31Б преподаватель каф. ИУ5  
 Абуховский Иван Александрович Гапанюк Юрий Евгеньевич

Подпись и дата: Подпись и дата:

Москва, 2021 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 располагаться в отдельном файле. При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

# Задача 1

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

# 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}

goods = [

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

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

{'title': 'Кресло', 'price': 7000, 'color': 'желтый'},

{'class': 'human', 'name': 'Bob'}

]

def field(items, \*args):

assert (len(args) > 0)

if len(args) == 1:

for arg in args:

for i in range(len(items)):

for it in items[i]:

if (it == arg) and (items[i][it]) != None:

yield items[i][it]

else:

for i in range(len(items)):

d = {}

for arg in args:

for it in items[i]:

if it == arg:

d[it] = items[i][it]

yield d

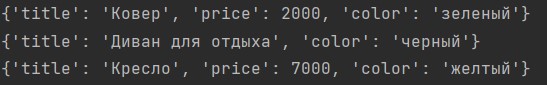
if \_\_name\_\_ == '\_\_main\_\_':

Vas = field(goods, "title", "price", "color")

for i in Vas:

print(i)

print()



# Задача 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 выдать 5 случайных чисел

# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

# Hint: типовая реализация занимает 2 строки

from random import randint

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

n = 0

while True:

if n < num\_count:

n += 1

yield randint(begin, end)

else:

break

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

a = gen\_random(5, 1, 3)

while True:

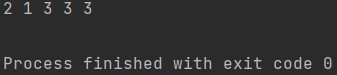
try:

print(next(a), end=' ')

except StopIteration:

print()

break



# Задача 3

from lab\_python\_fp.field import field

goods = [

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

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

{'title': 'Кресло', 'price': 7000, 'color': 'желтый'},

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

]

def unify(v):

return str(v).lower().strip()

class Unique(object):

n = 0

def \_\_init\_\_(self, items, ignore\_case=False):

# 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

self.ignore\_case = ignore\_case

self.data = items

self.dict = []

if ignore\_case == False:

for i in items:

if i not in self.dict:

self.dict.append(i)

else:

for i in items:

if unify(i) not in self.dict:

self.dict.append(unify(i))

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

if self.n < len(self.dict):

x = self.dict[self.n]

self.n += 1

return x

else:

raise StopIteration

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

return self

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

data1 = ['dDDdDa', "AAA", 'bbb', 'aaa', 'CCccC', 'aaa', 'ccccc']

data2 = [8, 7, 7, 1, 1, 2, 1, 3, 4, 5, 6]

for u in Unique(data1, ignore\_case=True):

print(u, end=' ')

print('\n')

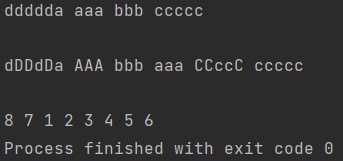
for u in Unique(data1):

print(u, end=' ')

print('\n')

for u in Unique(data2):

print(u, end=' ')



# Задача 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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.

2. Без использования lambda-функции.

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

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

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

print(result)

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

print(result\_with\_lambda)



# Задача 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.

def print\_result(funct):

def wrapper(\*args, \*\*kwargs):

print(funct.\_\_name\_\_)

res = funct(\*args, \*\*kwargs)

if isinstance(res, list):

for i in res:

print(i)

elif isinstance(res, dict):

for i in res:

print(i, '=', res[i])

else:

print(res)

return funct(\*args, \*\*kwargs)

return wrapper

# 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()



# Задача 6

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).

import time

class cm\_timer\_1:

def \_\_init\_\_(self):

self.start\_time=time.time()

def \_\_enter\_\_(self):

return 0

def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):

if exp\_type is not None:

print(exp\_type, exp\_value, traceback)

else:

print('time=' + str(time.time() - self.start\_time))

from contextlib import contextmanager

@contextmanager

def cm\_timer\_2():

start\_time=time.time()

yield 0

print('time='+str(time.time()-start\_time))

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

with cm\_timer\_1():

time.sleep(1.5)

with cm\_timer\_2():

time.sleep(1.0)



# Задача 7

* 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.

import json

import sys

from print\_result import \*

from cm\_timer import \*

from unique import Unique as uniqum

from gen\_random import gen\_random as gRand

path = "data\_light.json"

with open(path, 'r', encoding='utf-8') 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

def IT\_filter(el):

if el['job-name'][0:11].lower() == 'программист':

return True

else:

return False

@print\_result

def f1(arg):

return [u for u in uniqum([el.get('job-name') for el in arg], ignore\_case=True)]

@print\_result

def f2(arg):

return list(filter(lambda el: el[0:11].lower() == 'программист', arg))

@print\_result

def f3(arg):

return list(el+' с опытом Python' for el in arg)

@print\_result

def f4(arg):

zipped=list(zip(arg, gRand(len(arg), 100000, 200000)))

return [x+' с зарплатой '+str(y) for x, y in zipped]

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

with cm\_timer\_1():

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

