МГТУ им. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Дисциплина «Разработка интернет приложений»

Отчёт по лабораторной работе №3

## «Функциональные возможности языка Python.»

Выполнил:

Лункин В. И.

ИУ5-55Б

Преподаватель:

Гапанюк Ю.Е.

Москва, 2020г

**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):

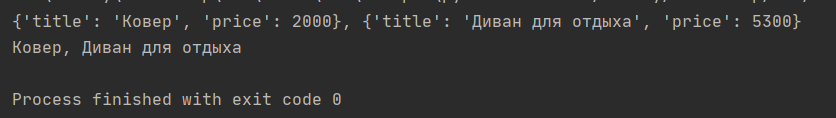
assert len(args) > 0

# Необходимо реализовать генератор

**field.py**

def field(items, \*args):  
 try:  
 assert len(args) > 0  
 except AssertionError:  
 print("Нет второго рагумента")  
 if len(args)==1:  
 for i in range(len(goods)):  
 if args[0] in goods[i] and goods[i].get(args[0])!=None:  
 if i==len(goods)-1:  
 print(goods[i].get(args[0]),end="\n")  
 else:  
 print(goods[i].get(args[0]),end=', ')  
 else:  
 for i in range(len(goods)):  
 s={}  
 for j in range(len(args)):  
 if args[j] in goods[i] and goods[i].get(args[j]) != None:  
 s.update({args[j]: goods[i].get(args[j])})  
 if i == len(goods) - 1:  
 print(s,end="\n")  
 else:  
 print(s, end=', ')  
  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'blue'}  
]  
field(goods,'title', 'price')  
field(goods,'title')

Результаты выполнения программы:



**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):

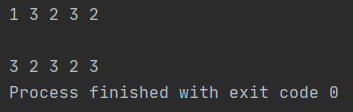
pass

# Необходимо реализовать генератор

**gen\_random.py**

from random import randint  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 yield randint(begin, end)  
  
  
def main():  
 gen = gen\_random(5, 1, 3)  
 for i in gen:  
 print(i, end=' ')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Результаты выполнения программы:



**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(1, 3, 10)

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

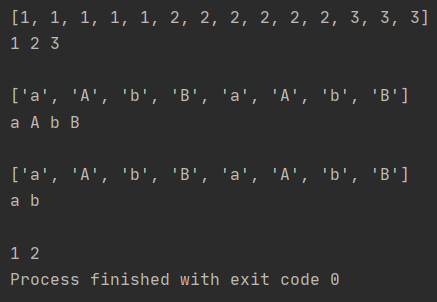
def \_\_iter\_\_(self):

return self

**unique.py**

import gen\_random  
  
  
class Unique(object):  
 def \_\_init\_\_(self,items, \*\*kwargs):  
 self.used\_elements=set()  
 self.items=items  
 self.index=0  
 if len(kwargs)!=0:  
 self.ignore\_case=kwargs  
 else:  
 self.ignore\_case=False  
  
 def \_\_next\_\_(self):  
 while True:  
 for item in self.items:  
 temp\_item=item  
 self.index+=1  
 if (temp\_item not in self.used\_elements) \  
 and not(self.ignore\_case and temp\_item.swapcase() in self.used\_elements):  
 self.used\_elements.add(temp\_item)  
 return temp\_item  
 else:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
def main():  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, ]  
 print(data1)  
 iter1 = Unique(data1)  
 for i in iter1:  
 print(i, end=' ')  
 print('\n')  
 data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print(data2)  
 iter2 = Unique(data2)  
 for i in iter2:  
 print(i, end=' ')  
 print('\n')  
 print(data2)  
 iter3 = Unique(data2, ignore\_case=True)  
 for i in iter3:  
 print(i, end=' ')  
 print('\n')  
 gen = gen\_random.gen\_random(3, 1, 3)  
 iter4 = Unique(gen)  
 for i in iter4:  
 print(i, end=' ')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Результаты выполнения программы:



**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)

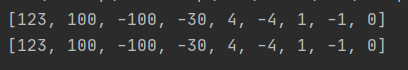
result\_with\_lambda = ...

print(result\_with\_lambda)

**sort.py**

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

Результаты выполнения программы:



**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

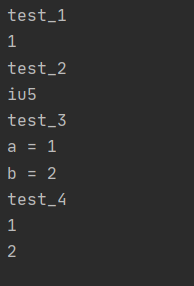
1

2

**print\_result.py**

def print\_result(func\_to\_decorate):  
 def decorated\_func():  
 print(func\_to\_decorate.\_\_name\_\_)  
 result = func\_to\_decorate()  
 if type(result) is list:  
 for i in result:  
 print(i)  
 elif type(result) is dict:  
 for i in result:  
 print(i, result.get(i), sep=' = ')  
 else:  
 print(result)  
 return decorated\_func()  
  
  
@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()

Результаты выполнения программы:



**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)

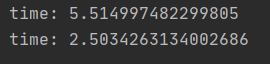
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

cm\_timer\_1 и cm\_timer\_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами.

**cm\_timer.py**

from contextlib import contextmanager  
from time import time, sleep  
  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.begin\_time = time()  
  
 def \_\_enter\_\_(self):  
 pass  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 if exp\_type is not None:  
 print(exp\_type, exp\_value, traceback)  
 else:  
 print('time:', time() - self.begin\_time)  
  
  
@contextmanager  
def cm\_timer\_2():  
 begin\_time=time()  
 yield 1  
 print('time:', time() - begin\_time)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 with cm\_timer\_1():  
 sleep(5.5)  
  
 with cm\_timer\_2():  
 sleep(2.5)

Результаты выполнения программы:



**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.

**process\_data.py**

import json  
from unique import Unique  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
from random import randint  
from gen\_random import gen\_random  
from field import field  
import re  
import sys  
  
path = 'data\_light.json'  
  
  
with open(path) as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return Unique(field(data, "job-name"), ignore\_case=True)  
  
  
@print\_result  
def f2(arg):  
 return filter(lambda x: re.search(r'\bПрограммист\b', x) or re.search(r'\bпрограммист\b', x), arg)  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x+' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 price = gen\_random(len(arg), 100000, 200000)  
 res = list(zip(arg, (list(map(lambda x: ', зарплата' + x + 'руб', ''.join(str(list(price)))[1:-1].split(','))))))  
 return [''.join(i) for i in res]  
  
  
def main():  
 with cm\_timer\_1():  
  
 f4(f3(f2(f1(data))))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Результаты выполнения программы:

