**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №3

«Функциональные возможности языка Python.»

|  |  |
| --- | --- |
| Выполнил: Сироткин Сергей |  |
| ИУ5-35Б |  |
|  |  |
|  |  |

[**Задание:**](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_oop#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5)

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)**](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-1-%D1%84%D0%B0%D0%B9%D0%BB-fieldpy)

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

def field(items, \*args):

assert len(args) > 0

c\_i = len(items)

c\_a = len(args)

for i in range(c\_i):

for j in range(c\_a):

if args[j] in items[i] and args[j] is not None:

yield items[i][args[j]]

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

goods = [

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

{'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}

]

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

print(ii, end=' ')

print('\n')

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

print(jj, end=' ')

print('\n')



[**Задача 2 (файл gen\_random.py)**](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-2-%D1%84%D0%B0%D0%B9%D0%BB-gen_randompy)

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.

import random

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

for i in range(num\_count):

yield random.randint(begin, end)

def main():

for num in gen\_random(5, 1, 3):

print(num, end=' ')

print('\n')

main()



[**Задача 3 (файл unique.py)**](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-3-%D1%84%D0%B0%D0%B9%D0%BB-uniquepy)

* 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.

from gen\_random import gen\_random

class unique(object):

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

self.ignore\_case = ignore\_case

self.items = items

self.index = 0

self.unique\_list = []

self.seen = set()

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

while self.index < len(self.items):

item = self.items[self.index]

self.index += 1

if self.ignore\_case:

item = item.lower()

if item not in self.seen:

self.seen.add(item)

return item

raise StopIteration()

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

return self

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

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

unique\_data1 = unique(data1)

for item in unique\_data1:

print(item, end=' ')

print('\n')

data2 = []

for num in gen\_random(5, 1, 3):

data2.append(num)

unique\_data2 = unique(data2)

for item in unique\_data2:

print(item, end=' ')

print('\n')

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

unique\_data3 = unique(data3, ignore\_case=True)

for item in unique\_data3:

print(item, end=' ')

print('\n')

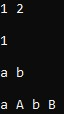
data4 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

unique\_data4 = unique(data3)

for item in unique\_data4:

print(item, end=' ')

print('\n')



### [Задача 4 (файл sort.py)](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-4-%D1%84%D0%B0%D0%B9%D0%BB-sortpy)

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.

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

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

def abs\_key(number):

return abs(number)

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

print(result)

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

print(result\_with\_lambda)



### [Задача 5 (файл print\_result.py)](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-5-%D1%84%D0%B0%D0%B9%D0%BB-print_resultpy)

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.

import inspect

def print\_result(func):

def wrapper(\*args):

print(func.\_\_name\_\_)

result = func(\*args)

if isinstance(result, list) or inspect.isgenerator(result):

for el in result:

print(el)

elif isinstance(result, dict) :

for key in result:

print(key, "=", result[key])

else:

print(result)

return wrapper

@print\_result

def test\_1():

return 1

@print\_result

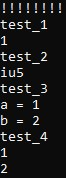
def test\_2():

return 'iu5'

@print\_result

def test\_3():

return {'a': 1, 'b': 2}

@print\_result

def test\_4():

return [1, 2]

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

print('!!!!!!!!')

test\_1()

test\_2()

test\_3()

test\_4()

### [Задача 6 (файл cm\_timer.py)](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-6-%D1%84%D0%B0%D0%B9%D0%BB-cm_timerpy)

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

import time

from contextlib import contextmanager

class cm\_timer\_1:

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

self.start\_time = time.time()

return self

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

end\_time = time.time()

print("Время выполнения кода:", end\_time - self.start\_time, "секунд")

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

yield

end\_time = time.time()

print("Время выполнения кода:", end\_time - start\_time, "секунд")

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

with cm\_timer\_1():

time.sleep(5.5)

with cm\_timer\_2():

time.sleep(5.5)



### [Задача 7 (файл process\_data.py)](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-7-%D1%84%D0%B0%D0%B9%D0%BB-process_datapy)

* 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

from field import field

from gen\_random import gen\_random

from unique import unique

from print\_result import print\_result

from cm\_timer import cm\_timer\_1

path = 'lab3-4/data\_light.json'

with open(path, encoding="utf-8") as f:

data = json.load(f)

@print\_result

def f1(arg):

return sorted(unique(list(field(data, 'job-name')), True))

@print\_result

def f2(arg):

return list(filter(lambda x: 'программист' in x, field(data, 'job-name')))

@print\_result

def f3(arg):

return list(map(lambda x: x + ' с опытом Python', filter(lambda x: 'программист' in x, field(data, 'job-name'))))

@print\_result

def f4(arg):

for job in list(filter(lambda x: 'программист' in x, field(data, 'job-name'))):

salary = list(gen\_random(1, 100000,200000))

yield job + ", зарплата " + str(salary[0]) + " руб."

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

with cm\_timer\_1():

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

