Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»
Лабораторная работа №3 на тему:
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-34Б:
Даниелян Алла Арамановна
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.В.
Подпись и дата:

Москва, 2022 г.

Постановка задачи:
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).
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.

Текст программы:
```
# Пример:
from cgi import print_arguments


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
    if len(args) == 1:
        a = args[0]
```

```
        b = []
        for g in items:
            if g[a]!=None:
                yield g[a]
        #print (b)
    else:
        d = dict().fromkeys(args)
        for g in items:
            for a in args:
                if g[a]!=None:
                    d[a] = g[a]
            yield d
            #print (d, end = " ")
            #d.clear()
fiell = field (goods, 'title')
#field(goods, 'title', 'price', 'color')
fiel = field(goods, 'title', 'price')
for i in range (2):
    print (fiel.__next__())
for i in range(2):
    print (fiell.__next__())
```

Результат выполнения программы:

```
енты/lab3/field.py
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
Ковер
Диван для отдыха
PS C:\Users\allad\OneDrive\Документы\lab3>
```

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:

Текст программы:

```
import random
def gen_random(num_count, begin, end):
    #r = [random.randrange(begin, end+1) for i in range( num_count)]
       return (random.randrange(begin, end+1) for i in range( num_count))



g =gen_random(5, 1, 3)
print(next(g))
print(next(g))
```

```
print(next(g))
print(next(g))
print(next(g))
print(next(g))
```

Результат выполнения программы:

```
PS C:\Users\allad\OneDrive\Документы\lab3> & "C:/Program Files/Python310/python.exe" c:/Users
енты/lab3/gen_random.py
1
2
1
1
2
Traceback (most recent call last):
  File "c:\Users\allad\OneDrive\Документы\lab3\gen_random.py", line 19, in <module>
    print(next(g))
StopIteration
```

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:

```
# Итератор для удаления дубликатов
#from binhex import *
class Unique(object):
    def __init__(self, items, ignore_case = 0):
        # Нужно реализовать конструктор
        self.data = items
        self.count = 0
        self.cash = set() #Тут хранятся просмотренные элементы
        self.ignore_case = ignore_case
        # 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, Абв и АБВ - разные строки
```

```
        #           ignore_case = False, Абв и АБВ - одинаковые строки, одна из
которых удалится
        # По-умолчанию ignore_case = False

    def __next__(self):
        # Нужно реализовать __next__
        # Кэш будем хранить в строковом виде. В зависимости от ignore_case будем
переводить все в верхний регистр или хранить в исходном виде
        while True:
            if self.count<len(self.data):
                self.count+=1
                if not self.ignore_case:
                   if  not str (self.data[self.count - 1]).upper() in self.cash:
                        self.cash.add(str(self.data[self.count - 1]).upper())
                        return  self.data [ self.count - 1]
                else:
                        if not str(self.data[self.count - 1]) in self.cash:
                            self.cash.add(str(self.data[self.count - 1]))
                            return self.data[self.count - 1]
            else:
                raise StopIteration


    def __iter__(self):
        return self


a = Unique(['A', 'a', 'Bb', 'BB', 'Ab', 'ab'],ignore_case = True)
b = Unique(['A', 'a', 'Bb', 'BB', 'Ab', 'ab'], ignore_case=False)
for x in a:
    print (x, end = ' ')
for x in b:
    print (x, end = ' ')
```

Результат выполнения программы:

```
PS C:\Users\allad\OneDrive\Документы\lab3> & "C:/Program Files/Python310/python.exe" c:/Users/allad/OneDr
енты/lab3/unique.py
A a Bb BB Ab ab
A Bb Ab
PS C:\Users\allad\OneDrive\Документы\lab3>
```

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.

Текст программы:

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

if __name__ == '__main__':
    result = sorted(data, key = abs)
    print(result)
    result_with_lambda =sorted (data, key = lambda a: abs(a))
    print(result_with_lambda)
```

Результат выполнения программы:

```
PS C:\Users\allad\OneDrive\Документы\lab3> & "C:/Program Files/Python310/python.exe" c:/Users/allad/OneDriv
енты/lab3/sort.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
[0, 1, -1, 4, -4, -30, 100, -100, 123]
PS C:\Users\allad\OneDrive\Документы\lab3>
```

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:

```
 Декоратор для печати
def print_result(func):
    def inner (*args, **kwargs):
        f = func(*args, **kwargs)
        print (func.__name__)
        # Если список
        if isinstance(f,list):
            for i in f:
                print (i)
        # Если словарь
        elif isinstance(f,dict):
            for i in f.keys():
                print ("{} = {}".format(i, f[i]))
        else:
            print (f)
        print("====================")
        return f
    return inner

@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 программы:

```
PS C:\Users\allad\OneDrive\Документы\lab3> & "C:/Program Files/Python310/python.exe" c:/Users/allad/OneDrive/Документы/lab3/print_result.py
!!!!!!!!
test_1
1
====================
test_2
iu5
====================
test_3
a = 1
b = 2
====================
test_4
1
2
====================
PS C:\Users\allad\OneDrive\Документы\lab3>
```

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 (на основе класса и с использованием библиотеки contextlib).

Текст программы:

```
from time import time
from time import sleep
from contextlib import contextmanager
```

```
class cm_timer_1 ():
    def __init__(self):
        pass
    def __enter__(self):
        self.t1 = time()
    def __exit__ (self,exp_type,exp_value,traceback):
        self.t2 = time()
        print ("time: {}".format (self.t2 - self.t1))

@contextmanager
def cm_timer_2():
    t1 = time()
    yield
    t2 = time()
    print ("time: {}".format(t2-t1))


with cm_timer_1():
    sleep(1.5)
with cm_timer_2():
    sleep(1.5)
```

Результат выполнения программы:

```
енты/lab3/cm_timer.py
time: 1.511336088180542
time: 1.5026867389678955
```

Задача №7 (Файл process_data.py)

- 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
# Сделаем другие необходимые импорты
from print_result import print_result
from gen_random import gen_random
from cm_timer import cm_timer_1
import unique
# Необходимо в переменную path сохранить путь к файлу, который был передан при
запуске сценария
path = "C:/Users/allad/OneDrive/Документы/lab3/data_light.json"
with open(path, 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

@print_result
def f1(arg):
    return sorted([i for i in unique.Unique([x["job-name"] for x in
data],ignore_case = False)])

@print_result
def f2(arg):
    return list(filter(lambda x:"Программист" in x, arg))

@print_result
def f3(arg):
    return list (map(lambda x: x+" с опытом Python", arg))

@print_result
def f4(arg):
    s= list( gen_random(len(arg),100000,200000))
    a= list(zip(arg,s))
    return  ["{}, зарплата {} рублей".format(x[0], x[1]) for x in a ]

if __name__ == '__main__':
    with cm_timer_1():
       f4(f3(f2(f1(data))))
```

Результат выполнения программы:

```
фтизиатрия
химик
художник-постановщик
швея - мотористка
шиномонтаж
шлифовщик 5 разряда
шлифовщик механического цеха
эколог
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер станционного телевизионного оборудования
электросварщик
энтомолог
юрисконсульт 2 категории
======================
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем


======================
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик  информационных систем с опытом Python
======================
f4
Программист с опытом Python, зарплата 180818 рублей
Программист / Senior Developer с опытом Python, зарплата 171009 рублей
Программист 1С с опытом Python, зарплата 196698 рублей
Программист C# с опытом Python, зарплата 108468 рублей
Программист C++ с опытом Python, зарплата 193600 рублей
Программист C++/C#/Java с опытом Python, зарплата 164846 рублей
Программист/ Junior Developer с опытом Python, зарплата 131873 рублей
Программист/ технический специалист с опытом Python, зарплата 101613 рублей
Программистр-разработчик  информационных систем с опытом Python, зарплата 144396 рублей
======================
time: 0.6990220546722412
PS C:\Users\allad\OneDrive\Документы\lab3>
```