Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-34Б:
Козак А.А.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е,
Подпись и дата:

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 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

```python
def field(items, *args):
    if len(args) > 1:
        for item in items:
            dict = {}
            for i in args:
                if item.get(i) is not None:
                    dict[i] = item[i]
            if dict.keys(): yield dict
    else:
        for item in items:
            for i in args:
                if item.get(i) is not None:
                    yield item.get(i)


goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

print(*list(field(goods,"title")))
print(*list(field(goods,"title","color")))
```

Тестирование

```
Ковер Диван для отдыха
{'title': 'Ковер', 'color': 'green'} {'title': 'Диван для отдыха', 'color': 'black'}
```

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

Текст программы

```
import random as rd

def gen_random(number,start,stop):
    for i in range(number):
      yield rd.randint(start,stop)

print(*list(gen_random(10,1,5)))
print(*list(gen_random(10,1,1)))
```

Тестирование

```
5 2 4 2 2 5 2 1 4 1
1 1 1 1 1 1 1 1 1 1
```

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.

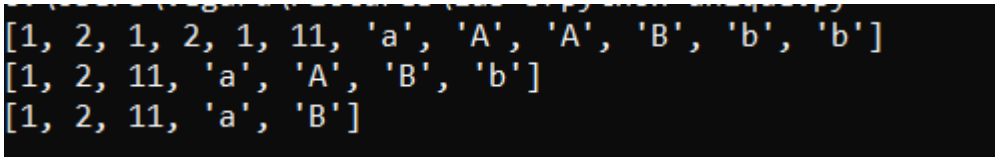
Текст программы

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = items
        count = []
        uniq = []
        if kwargs.get("ignore_case") == True:
          self.items = items
          for i in range(len(self.items)):
            if type(self.items[i]) == str:
              if self.items[i].casefold() not in count:
                uniq.append(self.items[i])
                count.append(self.items[i].casefold())
            elif self.items[i] not in count:
              count.append(self.items[i])
              uniq.append(self.items[i])
          self.items = uniq
        else:
          self.items = items
          for i in range(len(self.items)):
            if type(self.items[i]) == str:
              if self.items[i] not in count:
                uniq.append(self.items[i])
                count.append(self.items[i])
            elif self.items[i] not in count:
              count.append(self.items[i])
              uniq.append(self.items[i])
          self.items = uniq
    def __next__(self):
      try:
        return self.items.pop(0)
      except IndexError:
        raise StopIteration

    def __iter__(self):
        return self

if __name__ == "__main__":
    data = [1,2,1,2,1,11,'a','A','A','B','b','b']
    print(data)
    print(list(Unique(data)))
    print(list(Unique(data,ignore_case = True)))
```

Тестирование



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

```
def sort_without_lambda(data):
    return sorted(data, key = abs,reverse= True)

def sort_with_lambda(data):
    return sorted(data,key = lambda x: -abs(x))

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

if __name__ == '__main__':
    result = sort_without_lambda(data)
    print(result)

    result_with_lambda = sort_with_lambda(data)
    print(result_with_lambda)
```

Тестирование

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

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

```python
def print_result(func):
    def dec(*args,**kwargs):
        print(func.__name__)
        res = func(*args,**kwargs)
        string = ""
        if not isinstance(res,(dict,list)):
            string = str(res) + '\n'
        if type(res) == dict:
          for k,v in res.items():
              string += f"{k}={v}\n"
        if type(res) == list:
            for i in res:
                string += str(i) + '\n'
        print(string,end = '')
        return res
    return dec

@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()
```

Тестирование

```
!!!!!!!!
test_1
1
test_2
iu5
test_3
a=1
b=2
test_4
1
2
```

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

```python
from contextlib import contextmanager
from time import sleep , time


class cm_timer_1():
    def __init__(self):
      self.beg = 0
    def __enter__(self):
      self.beg = time()
    def __exit__(self,type, value, traceback):
      print(f"Time: {time()-self.beg}")


@contextmanager
def cm_timer_2(*args,**kargs):
    beg = time()
    try:
      yield beg
    finally:
      print(f"Time: {time()-beg}")

if __name__ == "__main__":
  with cm_timer_1():
    sleep(2)

  with cm_timer_2():
    sleep(2)
```

Тестирование

```
Time: 2.0011298656463623
Time: 2.0000617504119873
```

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.

Текст программы

```python
import json
import sys
from unique import Unique
from cm_timer import cm_timer_1
from print_result import print_result
# Сделаем другие необходимые импорты

path = "data_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

with open(path) as f:
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(list(Unique([x['job-name'] for x in arg],ignore_case=True)))


@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith("программист") or x.startswith("Программист"),arg))


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


@print_result
def f4(arg):
    sal = [", зарплата " + str(rd.randint(100_000,200_000)) + " руб" for i in range(len(arg))]
    return [x + y for x,y in zip(arg,sal)]



with cm_timer_1():
    f4(f3(f2(f1(data))))
```