Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

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

Отчет по лабораторной работе №3
по дисциплине «Базовые компоненты интернет-технологий»
Тема «Функциональные возможности языка Python»

Выполнил:
студент группы
ИУ5Ц-52Б Дзауров И.А.
_____, “___” _____ 2022 г.

Проверил:
преподаватель кафедры
ИУ5 - Гапанюк Ю.Е.
_____, “___” _____ 2022 г.

Москва, 2022 г.

Описание 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.

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
```

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
```

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(10, 1, 3)
```

`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, Абв и АБВ - разные строки
        #           ignore_case = False, Абв 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
```

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, 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)
```

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
```

```python
@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
```

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:

```python
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).

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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

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

with open(path) 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):
    raise NotImplemented


@print_result
def f2(arg):
```

```
    raise NotImplemented


@print_result
def f3(arg):
    raise NotImplemented


@print_result
def f4(arg):
    raise NotImplemented


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

Листинг программы

Структура проекта:

```
main.py
process-data.py
data_light.json
           lib/
           field.py
           gen_random.py
           unique.py
           sort.py
           cm_timer.py
           print_result.py
```

Основной файл - main.py:

```
# This is a sample Python script.

from lab_python_fp.field import field
from lab_python_fp.random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.sort import sort, sort_lambda
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1, cm_timer_2

from time import sleep

def main():
  print('\t<!----field----!>')
  example = [
    {'title': 'Кровать'},
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Торшер', 'price': 100, 'color': 'black'},
    {'title': 'бумага. Поле price - None', 'price': None, 'color': 'black'},
    {'titleNone': 'None. Оба поля для выборки - None', 'price': None, 'color':
None},
  ]
  for item in field(example, 'title', 'price'):
    print(item)
  print('\n')

  print('\t<!----random----!>')
  for item in gen_random(6, 10, 15):
    print(item)
  print('\n')

  print('\t<!----unique----!>')
  uniq_example = [1, 1, 1,4,3,4,3,3,3,3,3, 1, 1, 2, 2, 2, 2, 2]
  uniq_example_lower_strings = ["hello", "Hello", "hellO", "hi", "Hi"]
```

```
  uniq = Unique(uniq_example)
  print(f"-->test 1. ignore_case: {uniq.ignore_case()}")
  for item in uniq:
    print(item)

  uniq_str = Unique(uniq_example_lower_strings, ignore_case = True)
  print(f"\n-->test 2. ignore_case: {uniq_str.ignore_case()}")
  for item in uniq_str:
    print(item)
  print('\n')

  print('\t<!----sort----!>')
  sort_example = [4, -30, 100, -100, 123, 1, 0, -1, -4]
  print(sort(sort_example))
  print(sort_lambda(sort_example))

  print('\n\t<!----print_decorator_example----!>')
  @print_result
  def print_decorator_example1():
    return 1
  @print_result
  def print_decorator_example2():
      return 'iu5'
  @print_result
  def print_decorator_example4():
      return {'a': 1, 'b': 2}
  @print_result
  def print_decorator_example3():
      return [1, 2]
  print_decorator_example1()
  print_decorator_example2()
  print_decorator_example3()
  print_decorator_example4()

  print('\n<!----cm_timer----!>')
  with cm_timer_1():
      sleep(1)
  with cm_timer_2():
      sleep(1.5)

  print('\n<!----process_data----!>')


if __name__ == "__main__":
  main()
```

Файл - process data.py:

```
import json
import sys

from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.sort import sort
```

```python
from lab_python_fp.random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.field import field

path = 'data_light.json'

with open(path, 'r', encoding='utf8') 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(
    Unique(
      (el['job-name'] for el in list(field(arg, 'job-name'))[0]),
      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(map(lambda el: el + ' с опытом Python', arg))

@print_result
def f4(arg):
  salary = list(gen_random(len(arg), 100000, 200000))
  work = list(zip(arg, salary))
  return list(map(lambda el: el[0] + ', зарплата ' + str(el[1]) + '
руб.', work))

def main():
    print('\tproccess_data.py')
    with cm_timer_1():
        print(f4(f3(f2(f1(data)))))



if __name__ == "__main__":
    main()
```

Файл - lib/field.py:

```
def field(dist_list, *args):
    assert len(args) > 0
    yield [x for x in [
        {arg:dist_item[arg] for arg in
          args if arg in dist_item and dist_item[arg]} for dist_item in
dist_list if bool(dist_item)
      ] if bool(x)]
```

Файл - lib/gen_random.py:

```
from random import randint

def gen_random(num_count, begin, end):
  for i in range(num_count):
    yield randint(begin, end)
```

Файл - lib/unique.py:

```
class Unique(object):
    __ignore_case = False
    __items = []
    __iter = iter(__items)

    def __init__(self, items, **kwargs):
        self.__ignore_case = False
        self.__items = []
        self.__iter = iter(self.__items)

        if 'ignore_case' in kwargs:
            self.__ignore_case = bool(kwargs['ignore_case'])

        for item in items:
            if type(item) == type('') and self.__ignore_case:
                if not any(str(x).lower() == item.lower() for x in
self.__items):
                    self.__items.append(item)
            else:
                if not item in self.__items:
                    self.__items.append(item)

    def __next__(self):
        return next(self.__iter)

    def __iter__(self):
        return self.__iter
```

```
    def ignore_case(self):
        return self.__ignore_case
```

Файл - lib/sort.py:

```
def sort(arr):
    return sorted(arr, reverse=True, key = abs)

def sort_lambda(arr):
    return sorted(arr, reverse=True, key=lambda x: abs(x))
```

Файл - lib/cm_timer.py:

```
from time import time
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.__start = time()
    def __enter__(self):
        return self
    def __exit__(self, type, value, traceback):
        print('time: ', round(time() - self.__start, 2))

@contextmanager
def cm_timer_2():
    start = time()
    yield
    time_end = time()
    print('time: ', round(time() - start, 2))
```

Файл - lib/print_result.py:

```
def print_result(function):
    def decorated_func(*args):
        print(function.__name__)
        res = function(*args)
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for i in res.keys():
```

```
                print(i, ' = ', res[i])
        else:
            print(res)
        return res
    return decorated_func
```

Экранные формы с примерами выполнения программы

Файл - main.py:

```
     <!----field----!>
[{'title': 'Кровать'}, {'title': 'Ковер', 'price': 2000}, {'title':
'Диван для отдыха', 'price': 5300}, {'title': 'Торшер', 'price':
100}, {'title': 'бумага. Поле price - None'}]


     <!----random----!>
13
12
14
14
14
11


     <!----unique----!>
-->test 1. ignore_case: False
1
4
3
2

-->test 2. ignore_case: True
hello
hi


     <!----sort----!>
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

     <!----print_decorator_example----!>
print_decorator_example1
1
print_decorator_example2
iu5
print_decorator_example3
1
2
print_decorator_example4
a  =  1
b  =  2

<!----cm_timer----!>
time:  1.0
time:  1.51
```

<!----process_data----!>

Файл - process data.py:

f1

1С программист

2-ой механик

3-ий механик

4-ый механик

4-ый электромеханик

ASIC специалист

JavaScript разработчик

RTL специалист

Web-программист

[химик-эксперт

web-разработчик

Автожестянщик

Автоинструктор

Автомаляр

Автомойщик

Автор студенческих работ по различным дисциплинам

Автослесарь - моторист

Автоэлектрик

Агент

Агент банка

Агент нпф

Агент по гос. закупкам недвижимости

Агент по недвижимости

Агент по недвижимости (стажер)

Агент по недвижимости / Риэлтор

Агент по привлечению юридических лиц

Агент по продажам (интернет, ТВ, телефония) в ПАО Ростелеком в населенных пунктах Амурской области: г. Благовещенск, г. Белогорск, г. Свободный, г. Шимановск, г. Зея, г. Тында

Агент торговый

Агроном-полевод

...

БУХГАЛТЕР-Делопроизводитель

Бармен

Бармен-кассир в кафе

Бармен-официант

...

Машинист экскаватора

Машинист экскаватора 4-8 разряда

Машинист экскаватора 6 разряда

Машинист экскаватора одноковшового 5 разряда

Мед. сестра детского сада

Медбрат (медсестра) по массажу

Медицинская сестра (для работы в детских образовательных учреждениях)

Медицинская сестра

Медицинская сестра палатная

...

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 131458 руб.

Программист / Senior Developer с опытом Python, зарплата 129165 руб.

Программист 1С с опытом Python, зарплата 110880 руб.

Программист C# с опытом Python, зарплата 164037 руб.

Программист C++ с опытом Python, зарплата 104194 руб.

Программист C++/C#/Java с опытом Python, зарплата 109907 руб.

Программист/ Junior Developer с опытом Python, зарплата 182314 руб.

Программист/ технический специалист с опытом Python, зарплата 179450 руб.

Программистр-разработчик информационных систем с опытом Python, зарплата 135512 руб.

['Программист с опытом Python, зарплата 131458 руб.', 'Программист / Senior Developer с опытом Python, зарплата 129165 руб.', 'Программист 1С с опытом Python, зарплата 110880 руб.', 'Программист C# с опытом Python, зарплата 164037 руб.', 'Программист C++ с опытом Python, зарплата 104194 руб.', 'Программист C++/C#/Java с опытом Python, зарплата 109907 руб.', 'Программист/ Junior Developer с опытом

Python, зарплата 182314 руб.', 'Программист/ технический специалист с опытом Python, зарплата 179450 руб.', 'Программистр-разработчик информационных систем с опытом Python, зарплата 135512 руб.']

time: 2.69