Министерство образования и науки 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)

Отчёт по лабораторной работе №3 по курсу

«Разработка интернет-приложений»

Тема работы: «Функциональные возможности языка Python»

Выполнила: Попова Дарья, РТ5-51Б

Проверил: _____

17 ноября 2020 г.

ЗАЧТЕНО / НЕ ЗАЧТЕНО_____

(подпись)

Москва, 2020

Задание №1 (файл field.py)

```
# генератор выдаёт по очереди значения ключей словаря
def field(items, *args):
    if len(args) > 1:
        for d in items:
            for arg in args:
                if arg not in d.keys():
                    break
                else:
                    yield d
    elif len(args) > 0:
        for d in items:
            for dictKey in d.keys():
                for arg in args:
                    if dictKey == arg:
                        yield d.get(dictKey)


if __name__ == "__main__":
    goods = [
        {"title": "Шкаф", "price": 10000, "colour": "белый"},
        {"title": "Столешница", "price": 7500, "colour": "бежевый"},
        {"colour": "фиолетовый", "price": 3000},
        {"title": "Кухонный гарнитур", "price": 50000, "colour": "красный"},
        {"title": "Карниз", "price": 2300},
        {"title": "Подушка", "price": "розовый"},
        {"price": 1200, "colour": "чёрный"}
    ]
    print("Проверка для одного аргумента:")
    obj = field(goods, 'title')
    for i in obj:
        print(i)
    print("\nПроверка для наименования и цены:")
    obj = field(goods, "title", "price")
    for i in obj:
        print(i)
    print("\nПроверка для наименования и цвета:")
    obj = field(goods, "title", "color")
    for i in obj:
        print(i)
    print("\nПроверка для цены и цвета:")
    obj = field(goods, "price", "color")
    for i in obj:
        print(i)
```

Скрин с примером выполнения:

```
Проверка для одного аргумента:
Шкаф
Столешница
Кухонный гарнитур
Карниз
Подушка

Проверка для наименования и цены:
{'title': 'Шкаф', 'price': 10000, 'colour': 'белый'}
{'title': 'Шкаф', 'price': 10000, 'colour': 'белый'}
{'title': 'Столешница', 'price': 7500, 'colour': 'бежевый'}
{'title': 'Столешница', 'price': 7500, 'colour': 'бежевый'}
{'title': 'Кухонный гарнитур', 'price': 50000, 'colour': 'красный'}
{'title': 'Кухонный гарнитур', 'price': 50000, 'colour': 'красный'}
{'title': 'Карниз', 'price': 2300}
{'title': 'Карниз', 'price': 2300}
{'title': 'Подушка', 'price': 'розовый'}
{'title': 'Подушка', 'price': 'розовый'}

Проверка для наименования и цвета:
{'title': 'Шкаф', 'price': 10000, 'colour': 'белый'}
{'title': 'Столешница', 'price': 7500, 'colour': 'бежевый'}
{'title': 'Кухонный гарнитур', 'price': 50000, 'colour': 'красный'}
{'title': 'Карниз', 'price': 2300}
{'title': 'Подушка', 'price': 'розовый'}


Проверка для цены и цвета:
{'title': 'Шкаф', 'price': 10000, 'colour': 'белый'}
{'title': 'Столешница', 'price': 7500, 'colour': 'бежевый'}
{'colour': 'фиолетовый', 'price': 3000}
{'title': 'Кухонный гарнитур', 'price': 50000, 'colour': 'красный'}
{'title': 'Карниз', 'price': 2300}
{'title': 'Подушка', 'price': 'розовый'}
{'price': 1200, 'colour': 'чёрный'}
```

Задание №2 (файл gen_random.py)

```python
import random


# генератор заданного числа рандомных чисел в заданном промежутке
def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end)


if __name__ == "__main__":
    numbers = gen_random(5, 3, 27)
    for i in numbers:
        print(i)
```

Задание №3 (файл unique.py)

```python
from types import GeneratorType
from gen_random import gen_random
```

```python
# реализуем класс итератора
class Unique:
    def __init__(self, **kwargs):
        self.index = 0
        # обозначаем использованные элементы как (пока) пустое множество
        self.passedElements = set()
        self.ignore_Case = kwargs.get("ignoreCase")
        data = kwargs.get("data")
        if isinstance(data, GeneratorType):
            # если генератор, то создаётся пустой список, заполняющийся
значениями из data
            self.data = list()
            for genElement in data:
                print(genElement, end=' ')
                self.data.append(genElement)
            print("-->", end=' ')
        else:
            # если работа со списком, то просто заносим список в поле класса
            self.data = data
        # добавим поле с количеством элементов списка/генератора
        self.dataLength = len(self.data)
        pass

    def __next__(self):
        # если "курсор" текущего индекса выходит за границы списка, то
вызываем StopIteration
        if self.index >= self.dataLength:
            raise StopIteration
        else:
            element = self.data[self.index]
            # иначе сдвигаем "курсор" индекса на 1 вправо
            self.index += 1
            if isinstance(element, str) and self.ignore_Case is True:
                # если на вход был подан флаг ignoreCase, то переводим все
строки в нижний регистр
                elementReg = element.lower()
            else:
                # не строковые элементы передаём в использованные без
изменений
                elementReg = element
            if elementReg not in self.passedElements:
                # если строка в нижнем регистре не найдена в пройденных
элементах, добавляем её в множество
                self.passedElements.add(elementReg)
                return element

    def __iter__(self):
        return self


if __name__ == "__main__":
    print("\n\nПроверим работу итератора.")
    print("\nСписок с целыми числами:")
    dataList = [1, 1, 1, 2, 1, 2, 1, 3, 1]
    for i in dataList:
        print(i, end=' ')
    print("-->", end=' ')
    for i in Unique(data=dataList):
        if i is None:
            continue
        else:
            print(i, end=' ')
```

```
    print("\n\nГенератор с целыми числами:")
    for i in Unique(data=gen_random(8, 1, 5)):
        if i is None:
            continue
        else:
            print(i, end=' ')
    print("\n\nСписок с символами:")
    dataList = ['a', 'A', 'c', 'C', 'K']
    for i in dataList:
        print(i, end=' ')
    print("-->", end=' ')
    for i in Unique(data=dataList, ignoreCase=True):
        if i is None:
            continue
        else:
            print(i, end=' ')
```

Скрин с примером выполнения:

```
Проверим работу итератора.

Список с целыми числами:
1 1 1 2 1 2 1 3 1 --> 1 2 3

Генератор с целыми числами:
1 5 4 1 5 5 1 3 --> 1 5 4 3

Список с символами:
a A c C K --> a c K
```

Задание №4 (файл sort.py)

```
from gen_random import gen_random

if __name__ == "__main__":
    genList = gen_random(15, -30, 30)
    dataList = list()
    for i in genList:
        dataList.append(i)
    print("Исходный список: \n", dataList)
    print("Реализация без лямбда-функции: \n", list(i[1] for i in
reversed(sorted(zip([abs(num) for num in dataList], dataList)))))
    print("Реализация с лямбда-функцией: \n", list(i[1] for i in
reversed(sorted(zip(map(lambda x: abs(x), dataList), dataList)))))
```

Скрин с примером выполнения:

```
Исходный список:
 [-28, 30, -25, 12, 10, 1, -22, -25, -9, -21, -5, 4, -30, -13, -15]
Реализация без лямбда-функции:
 [30, -30, -28, -25, -25, -22, -21, -15, -13, 12, 10, -9, -5, 4, 1]
Реализация с лямбда-функцией:
 [30, -30, -28, -25, -25, -22, -21, -15, -13, 12, 10, -9, -5, 4, 1]
```

Задание №5 (файл print_result.py)

```
def decorator_print_result(function_for_printing):
    # реализация декоратора
    def my_decorator(*args, **kwargs):
        func_to_print = function_for_printing(*args, **kwargs)
        # вывод имени функции
        print("Функция {}
возвращает:".format(function_for_printing.__name__))
        # если функция возвращает список, выводим каждый элемент в отдельной
строчке
        if isinstance(func_to_print, list):
            for element in func_to_print:
                print(element)
        # если функция возвращает словарь, выводим в столбик пары ключ -
значение
        elif isinstance(func_to_print, dict):
            for element in func_to_print.items():
                print("Ключ = {} Значение = {}".format(element[0],
element[1]))
        else:
            # во всех остальных случаях просто организуем вывод
            print(func_to_print)
        return func_to_print
    return my_decorator


@decorator_print_result
def test_1():
    return 1


@decorator_print_result
def test_2():
    return 'ИУ5'


# функция, возвращающая словарь (выводит в столбик пары ключ - значение)
@decorator_print_result
def test_3():
    return {'key1': "value1", 'key2': "value2"}


# функция, возвращающая список (выводит элементы в стобик)
@decorator_print_result
def test_4():
    return [3.5, 6, 5.1]


if __name__ == '__main__':
    print('\n\nРезультат выполнения:')
    test_1()
    test_2()
    test_3()
    test_4()
```

Скрин с примером выполнения:

```
Результат выполнения:
Функция test_1 возвращает:
1
Функция test_2 возвращает:
ИУ5
Функция test_3 возвращает:
Ключ = key1 Значение = value1
Ключ = key2 Значение = value2
Функция test_4 возвращает:
3.5
6
5.1
```

Задание №6 (файл cm_timer.py)

```
import time
from gen_random import gen_random
from contextlib import contextmanager


# первый способ создания менеджера контекста - с помощью класса
# класс должен включать в себя методы   enter   и   self
class TimerClass:
    def __init__(self):
        print("Создали класс Timer")
        self.timer = None

    def __enter__(self):
        self.timer = time.time()

    # метод   exit   должен принимать три обязательных параметра: type,
value, traceback
    def __exit__(self, timer_type, timer_value, timer_traceback):
        self.timer = time.time() - self.timer
        print("Время: {}".format(self.timer))
        print("Покинули метод __exit__ класса Timer")


# второй способ создания менеджера контекста - с помощью генератора и
декоратора
# всё до yield - это аналог метода __enter__, а после - __exit__
@contextmanager
def TimerLib():
    print("Вход в функцию с декоратором contextmanager")
    timer = time.time()
    # будем замерять время на создание списка на 10000000 элементов
    [i for i in gen_random(100000, -10000000, 10000000)]
    yield time.time() - timer
    print("Выход из функции с декоратором contextmanager")


if __name__ == "__main__":
    print("\nРабота контекстного менеджера - класса")
    timer1 = TimerClass()
    with timer1:
        [i for i in gen_random(100000, -10000000, 10000000)]
```

```
    print("\nРабота контекстного менеджера с декоратором")
    timer2 = TimerLib()
    with timer2 as tm2:
        print("Время: {}".format(tm2))
```

Скрин с примером выполнения:

```
Работа контекстного менеджера - класса
Создали класс Timer
Время: 0.11665797233581543
Покинули метод __exit__ класса Timer

Работа контекстного менеджера с декоратором
Вход в функцию с декоратором contextmanager
Время: 0.10870885848999023
Выход из функции с декоратором contextmanager
```

Задание №7 (файл process_data.py)

```
import json
from gen_random import gen_random
from print_result import decorator_print_result
from unique import Unique
from field import field
from cm_timer import TimerClass


# в переменной path сохранён путь к файлу, который был передан при запуске
сценария
path = "c:\\WAD\\data_light.json"


# функция f1 возвращает отсортированный список профессий без повторов
# сортировка должна игнорировать регистр (флажок ignore case = true)
@decorator_print_result
def f1(data_file):
    return sorted(job for job in Unique(data=field(data_file, "job-
name"),ignore_case=True))


# функция f2 фильтрует входной список словарей и возвращает только элементы,
# начинающиеся со слова "программист"
@decorator_print_result
def f2(sorted_data):
    return list(filter(lambda x: "Программист" in x, sorted_data))


# функция f3 модифицирует каждый элемент
# добавляя "с опытом Python"
@decorator_print_result
def f3(modified_data):
    return list(map(lambda x: x + "с опытом Python", modified_data))


# функция f4 генерирует для каждой специальности з/п от 100к до 200к
# и присоединяет её к названию специальности
@decorator_print_result
def f4(modified_data):
    return list(str(element[0]) + "З/п: " + str(element[1]) for element in
```

```
    zip(modified_data, gen_random(len(modified_data), 100000, 200000)))


    if __name__ == '__main__':
        data = list()

        with open(path, encoding='utf8') as datafile:
            data = json.load(datafile)
        with TimerClass():
            f4(f3(f2(f1(data))))
```

Не будем приводить полный вывод для первой функции, так как он слишком объёмный ☺. Кусочек:

f1:

```
веб-программист
ведущий агрохимик лаборатории полевых изысканий отдела агроэкологического мониторинга почв
ведущий бухгалтер, экономист
ведущий инженер отдела главного метролога
ведущий инженер-технолог
ведущий методист
ведущий специалист
ведущий специалист 3-го разряда
ведущий специалист отдела экономики
ведущий специалист по энергетике
весовщик
ветврач
ветеринарный врач
```

f2:

```
Функция f2 возвращает:
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:

```
Функция f3 возвращает:
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:

```
Функция f4 возвращает:
Программист с опытом Python з/п: 114714
Программист / Senior Developer с опытом Python з/п: 131627
Программист 1С с опытом Python з/п: 190552
Программист C# с опытом Python з/п: 101505
Программист C++ с опытом Python з/п: 113263
Программист C++/C#/Java с опытом Python з/п: 176551
Программист/ Junior Developer с опытом Python з/п: 147721
Программист/ технический специалист с опытом Python з/п: 131485
Программистр-разработчик  информационных систем с опытом Python з/п: 195243
Время: 0.06129622459411621
Покинули метод __exit__ класса Timer
```