МГТУ им. Н.Э. Баумана

Отчёт по лабораторной работе №3-4
по курсу «Парадигмы и конструкции языков программирования»
Тема: Функциональные возможности языка Python

Проверил:
Нардид А.Н.

Студент группы ИУ5-36Б
Мохаммед М. Н.

2024 г.

Задание:

Задача 1 (файл field.py)

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

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`

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.

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.

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.

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.

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.

Листинг:

Cm_timer.py

```
import time as t
from contextlib import contextmanager
class cm_timer_1:
    def __enter__(self):
        self.start = t.time()
        return self
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.exit = t.time()
        self.total_time = self.exit - self.start
        print(f'time: {self.total_time:.1f}')
@contextmanager
def cm_timer_2():
    start = t.time()  # Запоминаем время начала
    yield  # Позволяем выполнить блок кода
    exit = t.time()  # Запоминаем время завершения
    total = exit - start  # Вычисляем разницу
    print(f'time: {total:.1f}')  # Выводим результат

with cm_timer_1():
    t.sleep(1)
with cm_timer_2():
    t.sleep(1)
```

field.py

```
def field(goods, *args):
    for item in goods:
        if len(args) == 1:
            value = item.get(args[0])
            if value is not None:
                yield value
        else:
            filtered_item = {}
            for arg in args:
                if item.get(arg) is not None:
                    filtered_item[arg] = arg
            if filtered_item:
                yield filtered_item
```

gen_random.py

```
import random
def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randrange(begin, end)

print(list(gen_random(5, 1, 3)))
```

main.py

```
import time as t
import gen_random
import print_result
import cm_timer
import unique
import field
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'title': 'Стол', 'price': None}
]
print('Задача 1:')
print(list(field.field(goods, 'title')))
```

```
for i in field.field(goods, 'title', 'price'):
    print(i)

print('Задача 2: ')
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
if __name__ == '__main__':
    result = sorted(data, key = abs, reverse = True)
    print(result)
    result_with_lambda = sorted(data, key = lambda i: abs(i), reverse = True)
    print(result_with_lambda)

print('Задача 3: ')
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
u = unique.Unique(data)
print(list(u))

print('Задача 4: ')
print(list(gen_random.gen_random(5, 1, 3)))

print('Задача 5: ')
with cm_timer.cm_timer_1():
    t.sleep(1)


print('Задача 6: ')
if __name__ == '__main__':
    print('!!!!!!!!!')
    print_result.test_1()
    print_result.test_2()
    print_result.test_3()
    print_result.test_4()
```

print_result.py

```
def print_result(f):
    def wrapper(*args, **kwargs):
        func = f(*args, **kwargs)
        h = f.__name__
        print(h)
        if isinstance(func, dict):
            for key, item in func.items():
                print(f'{key} = {item}')
        elif isinstance(func, list):
            for i in func:
                print(i)
        else:
            print(func)
        return func
    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]
```

process_data.py

```
import json
from gen_random import gen_random
from print_result import print_result
from cm_timer import cm_timer_1

path = "data_light.json"
with open(path, encoding= 'utf-8') as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(set(i['job-name'].lower() for i in arg))


@print_result
def f2(arg):
    return list(filter(lambda x: 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 = gen_random(len(arg), 100000, 200000)
    new = zip(arg, sal)
    return ['{} зарплата {}'.format(job, salary) for job, salary in new]


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

sort.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
if __name__ == '__main__':
    result = sorted(data, key = abs, reverse = True)
    print(result)
    result_with_lambda = sorted(data, key = lambda i: abs(i), reverse = True)
    print(result_with_lambda)
```

unique.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = iter(items)
        self.new_set = set()
        self.ignore_case = kwargs.get('ignore_case', False)
```

```
    def __next__(self):
        while True:
            try:
                item = next(self.items)
                lookup_item = item.lower() if self.ignore_case and
isinstance(item, str) else item

                if lookup_item not in self.new_set:
                    self.new_set.add(lookup_item)
                    return item
            except StopIteration:
                raise StopIteration

    def __iter__(self):
        return self


data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
u = Unique(data)
print(list(u))
```

Анализ результатов

```
[1, 1, 2, 2, 2]
time: 1.0
time: 1.0
[1, 2]
Задача 1:
['Ковер', 'Диван для отдыха', 'Стол']
{'title': 'title', 'price': 'price'}
{'title': 'title'}
{'title': 'title'}
Задача 2:
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
Задача 3:
[1, 2]
Задача 4:
[2, 1, 1, 1, 1]
Задача 5:
time: 1.0
Задача 6:
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```