

**Московский государственный технический
университет им. Н.Э. Баумана**

Факультет Радиотехнический
Кафедра РТ5

Курс «Программирование на основе классов и шаблонов»

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

Выполнил:

студент группы РТ5-31Б:
Топорин Б.Г

Подпись и дата:

Проверил:

преподаватель каф. ИУ5
Гапанюк Ю.Е.

Подпись и дата:

Москва, 2023

Описание Задания

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fr. Решение каждой задачи должно располагаться 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 количество 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 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, 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 выводиться в столбик через знак равенства.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора
```

```
@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:

```
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`).

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))))
```

Текст программы

1. fieled.py

```
def field(items, *args):  
    assert len(args) > 0  
    for item in items:  
        if len(args) == 1:  
            var = item.get(args[0])  
            if var is not None:  
                yield var  
        else:  
            new_dic = {}
```



```

        all_none = True
        for key in args:
            var = item.get(key)
            if var is not None:
                new_dic[key] = var
                all_none = False
            if not all_none:
                yield var

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

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))

```

```

/usr/bin/python3 /Users/bogdan/PycharmProjects/PCPL2023/LAB2/fieled.py
['Ковер', 'Диван для отдыха']
['Ковер', 2000, 'Диван для отдыха', None]

```

2. fieled.py

```

import random

def gen_random(count, start, finish):
    for _ in range(count):
        yield random.randint(start, finish)

random_numbers = list(gen_random(5, 1, 3))
print(random_numbers)

pass

/usr/bin/python3 /Users/bogdan/PycharmProjects/PCPL2023/LAB2/gen_random.py
[2, 3, 3, 2, 1]

```

3. unique.py

```

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

    def __next__(self):
        while True:
            item = next(self.items)
            key = item.lower() if self.ignore_case and isinstance(item, str) else item
            if key not in self.unique_set:
                self.unique_set.add(key)
                return item

    def __iter__(self):
        return self

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
unique_iter1 = Unique(data1)

```

```

print(list(unique_iter1)) # [1, 2]

from gen_random import gen_random

data2 = gen_random(10, 1, 3)
unique_iter2 = Unique(data2)
print(list(unique_iter2)) # [1, 2, 3]

data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
unique_iter3 = Unique(data3)
print(list(unique_iter3)) # ['a', 'A', 'b', 'B']

data4 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
unique_iter4 = Unique(data4, ignore_case=True)
print(list(unique_iter4)) # ['a', 'b']

[1, 2]
[1, 2, 3, 2, 1]
[1, 3]
['a', 'A', 'b', 'B']
['a', 'b']

```

4. sort.py

```

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
if __name__ == '__main__':
    # Решение без lambda-функции
    def key_function(x):
        return abs(x)

    result = sorted(data, key=key_function, reverse=True)
    print(result)

    # Решение с lambda-функцией
    result_with_lambda = sorted(data, key=lambda x: abs(x),
reverse=True)
    print(result_with_lambda)

/usr/bin/python3 /Users/bogdan/PycharmProjects/PCPL2023/LAB2/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

```

5. print_result.py

```

def print_result(func):
    def write(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(result, list):
            for item in result:
                print(item)
        if isinstance(result, dict):
            for key, var in result.items():
                print(f'{key} = func{var}')
        else:
            print(result)
        return result

    return write()

```

```

@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()

```

`/usr/bin/python3 /Users/bogdan/PycharmProjects/PCPL2023/LAB2/print_result.py`

```

test_1
1
test_2
iu5
test_3
a = func1
b = func2
test_4
1
2
[1, 2]
!!!!!!!

```

6. cm_timer.py

```

import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self.end_time = time.time()
        elapsed_time = self.end_time - self.start_time
        print(f"time: {elapsed_time}")

@contextmanager
def cm_timer_2():
    start_time = time.time()

```

```

        yield
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"time: {elapsed_time}")

with cm_timer_1():
    time.sleep(5.5)
with cm_timer_2():
    time.sleep(5.5)
/usr/bin/python3 /Users/bogdan/PycharmProjects/PCPL2023/LAB2/cm_timer.py
time: 5.50322699546814

```

7. process_data.py

```

8. import json
import sys

import time
from contextlib import contextmanager

@contextmanager
def cm_timer_1():
    start_time = time.time()
    yield
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"time: {elapsed_time}")

def print_result(func):
    def write(*args, **kwargs):
        result = func(*args, **kwargs)
        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f"{key} = {value}")
        else:
            print(result)
        return result

    return write

path = "/Users/bogdan/Documents/data_light.json"

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

@print_result
def f1(arg):
    return (sorted(set(job['job-name'].lower() for job in arg if
isinstance(job, dict)), key=str.lower)) if isinstance(arg, list) else
[]

@print_result
def f2(arg):
    return list(filter(lambda job: job.startswith('программист'), arg))

```

```

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

@print_result
def f4(arg):
    return [job + f", зарплата {salary} руб." for job, salary in
zip(arg, range(100000, 200001, 1000))]

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
программист, с опытом Python, зарплата 100000 руб.
программист / senior developer, с опытом Python, зарплата 101000 руб.
программист 1с, с опытом Python, зарплата 102000 руб.
программист с#, с опытом Python, зарплата 103000 руб.
программист с++, с опытом Python, зарплата 104000 руб.
программист с++/с#/java, с опытом Python, зарплата 105000 руб.
программист/ junior developer, с опытом Python, зарплата 106000 руб.
программист/ технический специалист, с опытом Python, зарплата 107000 руб.
программист-разработчик информационных систем, с опытом Python, зарплата 108000 руб.
time: 0.008047819137573242

```