

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

**Факультет «Информатика и вычислительная техника»
Кафедра ИУ5 «Система обработки информации и управления»**

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

**Выполнил
студент ИУ5-35Б
Рябова С.А.**

**Проверил
Преподаватель каф. ИУ5
Гапанюк Ю.Е.**

Москва 2024

Описание задания

Задание:

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 NotImplementedError
```

```
@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 программ

Cm_timer.py

```
import time
from contextlib import contextmanager

class cm_timer_1():
    def __init__(self):
        self.begin = None
        self.end = None

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

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.end = time.time()
        print('time: %.1f % (self.end - self.begin))

@contextmanager
def cm_timer_2():
    begin = time.time()
    try:
        yield begin
```



```
finally: #всегда
    print('time: %.1f' % (time.time() - begin))
```

```
with cm_timer_2(): #оператор
    time.sleep(5.5)
```

field.py

```
#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 #проверка длины на 0
    for item in items:
        if len(args) == 1:
            value = item.get(args[0])
            if value is not None:
                yield value
        else:
            result = {}
            for arg in args:
                value = item.get(arg)
                if value is not None:
                    result[arg] = value
            if result:
                yield result
```

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': None, 'price': None, 'color': 'red'}
]
```

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

gen_random.py

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

```
for i in gen_random(15, -100, 100):
```

```
print(i, end=' ')
```

print_result.py

```
#декоратор print_result
```

```
def print_result(func):  
    def inner1(*args, **kwargs):  
        result = func(*args, **kwargs)  
        print(func.__name__) #атрибут функции  
        if type(result) == list: #функция вернула список  
            for i in result:  
                print(i)  
        elif type(result) == dict: #функция вернула словарь  
            for key, val in zip(result.keys(), result.values()):  
                print(key, '=', val)  
        else:  
            print(result)  
    return inner1
```

```
@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()
```

process_data.py

```
import json
import time
import sys
import random

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

def print_result(func):
    def inner1(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if type(result) == list:
            for i in result:
                print(i)
        elif type(result) == dict:
            for key, val in zip(result.keys(), result.values()):
                print(key, '=', val)
        else:
            print(result)
        return result
    return inner1

class cm_timer_1():
    def __init__(self):
        self.begin = None
        self.end = None

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

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.end = time.time()
        print('time: %.3f % (self.end - self.begin)) #%% - str

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

```

result = {}
for arg in args:
    value = item.get(arg)
    if value is not None:
        result[arg] = value
if result:
    yield result

```

```

class Unique(object):
    def __init__(self, items, **kwargs):
        self.it = iter(items)
        self.cur = next(self.it)
        self.prev = ""
        if len(kwargs) > 0:
            self.cs = kwargs['ignore_case']
        else:
            self.cs = False

    def __next__(self):
        if type(self.cur) == str and self.cs:
            while self.cur.lower() == self.prev.lower():
                self.cur = next(self.it)
            self.prev = self.cur
            return self.cur
        else:
            while self.cur == self.prev:
                self.cur = next(self.it)
            self.prev = self.cur
            return self.cur

    def __iter__(self):
        return self

```

```

path = '/Users/sonyaryabova/PycharmProjects/pythonProject24/Lab_3-4/data_light.json'

```

```

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

```

```

#отсортированный список без повторений

```

```

@print_result

```

```

def fl(data):
    return list(Unique(sorted(list((field(data, 'job-name')))), ignore_case=False))

```

```

#фильтр массива, возвращение слов начинающиеся с программист

```

```
@print_result
def f2(seq):
    return list(filter(lambda st: len(st) >= 11 and st[:11].lower() == 'программист',
seq))
```

#добавление строки

```
@print_result
def f3(seq):
    return list(map(lambda x: x + ' с опытом Python', seq))
```

#Генерация зарплаты

```
@print_result
def f4(seq):
    return [x + ' ' + str(y) for x, y in zip(seq, list(gen_random(len(seq), 100000,
200000)))]
```

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

sort.py

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

```
if __name__ == '__main__':
    result = sorted(data, reverse=True, key=abs)
    print(result)
    result_with_lambda = sorted(data, key=lambda a: -1 * abs(a))
    print(result_with_lambda)
```

unique.py

```
import random
```

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

Итератор для удаления дубликатов

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.it = iter(items)
        self.cur = next(self.it)
```

```

self.prev = "
if len(kwargs) > 0:
    self.cs = kwargs['ignore_case']
else:
    self.cs = False

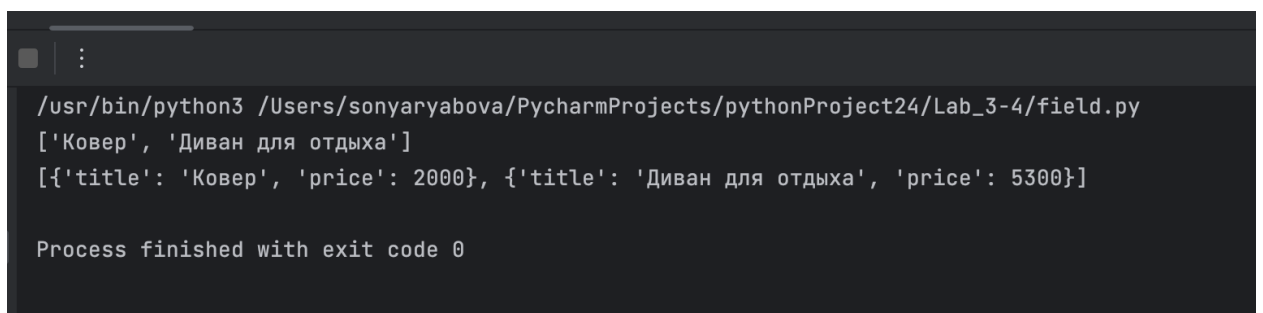
def __next__(self):
    if type(self.cur) == str and self.cs:
        while self.cur.lower() == self.prev.lower():
            self.cur = next(self.it)
        self.prev = self.cur
        return self.cur
    else:
        while self.cur == self.prev:
            self.cur = next(self.it)
        self.prev = self.cur
        return self.cur

def __iter__(self):
    return self

data = gen_random(10, 1, 3)
data1 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
for i in Unique(data):
    print(i, end=' ')
for i in Unique(data1, ignore_case=True):
    print(i, end=' ')

```

Экранные формы выполнения программ



```

/usr/bin/python3 /Users/sonyaryabova/PycharmProjects/pythonProject24/Lab_3-4/field.py
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]

Process finished with exit code 0

```

```
/usr/bin/python3 /Users/sonyaryabova/PycharmProjects/pythonProject24/Lab_3-4/gen_random.py  
12 3 -20 -66 78 -98 95 -82 -29 85 85 93 50 -77 57  
Process finished with exit code 0
```

```
/usr/bin/python3 /Users/sonyaryabova/PycharmProjects/pythonProject24/Lab_3-4/unique.py  
3 2 1 3 2 3 a b a b  
Process finished with exit code 0
```

```
/usr/bin/python3 /Users/sonyaryabova/PycharmProjects/pythonProject24/Lab_3-4/sort.py  
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]  
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]  
Process finished with exit code 0
```

```
0ms ===== test session starts =====
collecting ... collected 4 items

print_result.py::test_1 PASSED [ 25%]test_1
1

print_result.py::test_2 PASSED [ 50%]test_2
iu5

print_result.py::test_3 PASSED [ 75%]test_3
a = 1
b = 2

print_result.py::test_4 PASSED [100%]test_4
1
2

===== 4 passed in 0.03s =====

Process finished with exit code 0
```

```
/usr/bin/python3 /Users/sonyaryabova/PycharmProjects/pythonProject24/Lab_3-4/cm_timer.py
time: 5.5

Process finished with exit code 0
```

```
↑ юрисконсульт 2 категории
↓ юрист
f2
⇌ Программист
⇓ Программист / Senior Developer
📄 Программист 1C
🗑 Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
программист
программист 1C
```



```
↑
↓
⇌
⇅
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1C 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
Программист-разработчик информационных систем с опытом Python
программист с опытом Python
программист 1C с опытом Python
```

```
f4
Программист с опытом Python 111534
Программист / Senior Developer с опытом Python 192874
Программист 1C с опытом Python 146630
Программист C# с опытом Python 112306
Программист C++ с опытом Python 146162
Программист C++/C#/Java с опытом Python 165952
Программист/ Junior Developer с опытом Python 161825
Программист/ технический специалист с опытом Python 117387
Программист-разработчик информационных систем с опытом Python 139718
программист с опытом Python 107334
программист 1C с опытом Python 177576
time: 0.018
```