

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

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

Курс
«Парадигмы и конструкции языков программирования»

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

Выполнил:
студент группы ИУ5-31Б
Санников Н.А.

Проверил:
преподаватель каф. ИУ5

Москва, 2024 г.

Цель лабораторной работы

Изучение возможностей функционального программирования в языке Python.

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`)

Задание:

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
```

Текст программы:

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

# Тестирование
if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    print(list(field(goods, 'title')))
    print(list(field(goods, 'title', 'price')))

#
```

экранные формы с примерами выполнения программы:

```
4xtacy@4xtacys-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/field.py
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]
4xtacy@4xtacys-MacBook-Pro lab3-4 %
```

Задача 2 (файл gen_random.py)

Задание:

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
```

Текст программы:

```
import random
```

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

```
# Тестирование
if __name__ == "__main__":
    print(list(gen_random(5, 1, 3)))
```

```
#
```

экранные формы с примерами выполнения программы:

```
4xtacy@4xtacy-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/gen_random.py
[3, 1, 3, 3, 3]
```

```
4xtacy@4xtacy-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/gen_random.py
[1, 3, 2, 3, 2]
```

Задача 3 (файл unique.py)

Задание:

- 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
```

Текст программы:

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

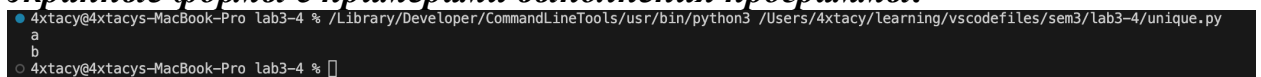
    def __iter__(self):
        return self

    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.seen:
                self.seen.add(key)
                return item

# Тестирование
if __name__ == "__main__":
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data, ignore_case=True):
        print(item)

#
```

экранные формы с примерами выполнения программы:



```
4xtacy@4xtacy-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/unique.py
a
b
4xtacy@4xtacy-MacBook-Pro lab3-4 %
```

Задача 4 (файл sort.py)

Задание:

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)

```

Текст программы:

```

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

if __name__ == "__main__":

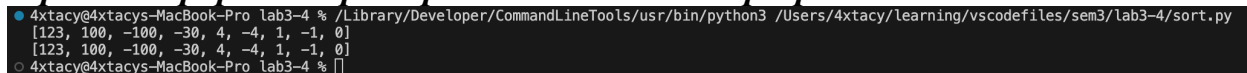
    result = sorted(data, key=abs, reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)

#

```

экранные формы с примерами выполнения программы:



```

4xtacy@4xtacys-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
4xtacy@4xtacys-MacBook-Pro lab3-4 %

```

Задача 5 (файл print_result.py)

Задание:

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
```

Текст программы:

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        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 wrapper
```

```
# Тестирование
if __name__ == "__main__":
    @print_result
    def test_1():
        return 1
```

```
@print_result
def test_2():
    return 'iu5'
```

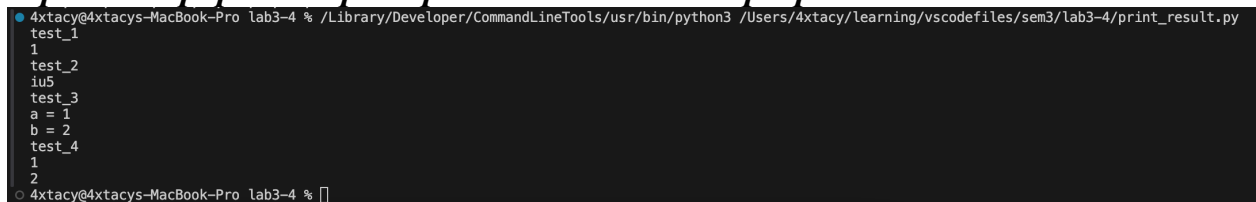
```
@print_result
def test_3():
    return {'a': 1, 'b': 2}
```

```
@print_result
def test_4():
    return [1, 2]
```

```
test_1()
test_2()
test_3()
test_4()
```

#

экранные формы с примерами выполнения программы:



```
4xtacy@4xtacys-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
4xtacy@4xtacys-MacBook-Pro lab3-4 %
```

Задача 6 (файл cm_timer.py)

Задание:

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`).

Текст программы:

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

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(f'time: {time.time() - self.start_time}')
```



```
@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print(f'time: {time.time() - start_time}')
```

```
# Тестирование
if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(1)

    with cm_timer_2():
        time.sleep(1)
```

```
#
```

экранные формы с примерами выполнения программы:

```
4xtacy@4xtacys-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/cm_timer.py
time: 1.0050561428070068
time: 1.0050790309906006
4xtacy@4xtacys-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/cm_timer.py
time: 1.0046792030334473
time: 1.0050287246704102
4xtacy@4xtacys-MacBook-Pro lab3-4 % /Library/Developer/CommandLineTools/usr/bin/python3 /Users/4xtacy/learning/vscodefiles/sem3/lab3-4/cm_timer.py
time: 1.002837896347046
time: 1.005084753036499
4xtacy@4xtacys-MacBook-Pro lab3-4 %
```

Задача 7 (файл process_data.py)

Задание:

- 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:

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

path = 'data_light.json'

with open(path) as f:
```

```

data = json.load(f)

@print_result
def f1(arg):
    return sorted(Unique([job['job-name'].lower() for job in arg if 'job-name' in job]), key=lambda
x: x.lower())
    #извлекает значения по ключу 'job-name', приводит их к нижнему регистру, удаляет
дубликаты и сортирует в алфавитном порядке.

@print_result
def f2(arg):
    # Фильтрация профессий, которые начинаются со слова "программист"
    return list(filter(lambda x: x.startswith('программист'), arg))

@print_result
def f3(arg):
    # Добавление строки "с опытом Python" к каждой профессии
    return list(map(lambda x: f'{x} с опытом Python', arg))

@print_result
def f4(arg):
    # Генерация зарплаты и объединение с названием профессии
    salaries = gen_random(len(arg), 100000, 200000)
    return list(map(lambda x, y: f'{x}, зарплата {y} руб.', arg, salaries))

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

#

```

экранные формы с примерами выполнения программы:

```

электросварщик ручной сварки
электросварщики ручной сварки
электрослесарь (слесарь) дежурный и по ремонту оборудования, старший
электрослесарь по ремонту и обслуживанию автоматики и средств измерений электростанций
электрослесарь по ремонту оборудования в карьере
электроэрозиялист
эндокринолог
энергетик
энергетик литейного производства
энтомолог
юриисконсульт
юриисконсульт 2 категории
юриисконсульт. контрактный управляющий
юриист
юриист (специалист по сопровождению международных договоров, английский – разговорный)
юриист волонтер
юриисконсульт
f2
программист
программист / senior developer
программист 1с
программист c#
программист c++
программист c++/c#/java
программист/ junior developer
программист/ технический специалист
программист-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
программист-разработчик информационных систем с опытом Python
f4
программист с опытом Python, зарплата 178767 руб.
программист / senior developer с опытом Python, зарплата 190191 руб.
программист 1с с опытом Python, зарплата 124596 руб.
программист c# с опытом Python, зарплата 183832 руб.
программист c++ с опытом Python, зарплата 199523 руб.
программист c++/c#/java с опытом Python, зарплата 158951 руб.
программист/ junior developer с опытом Python, зарплата 143402 руб.
программист/ технический специалист с опытом Python, зарплата 193806 руб.
программист-разработчик информационных систем с опытом Python, зарплата 133839 руб.
time: 0.0072839260131836
© 4xtacy@4xtacys-MacBook-Pro lab3-4 %

```