



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

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

Информатика и системы управления

КАФЕДРА

Системы обработки информации и управления

О т ч ё т п о л а б о р а т о р н о й р а б о т е

п о к у р с у

« Р а з р а б о т к а и н т е р н е т - п р и л о ж е н и й »

Python. Функциональные возможности.

Исполнитель: студентка группы **РТ5-51**

Галичий Д.А.

Преподаватель: **Гапанюк Ю.Е.**

Цель работы: ознакомление с функциональными возможностями Python, создание модуля для выполнения задания на реальных данных.

Задача 1. Необходимо реализовать генераторы `field` и `gen_random`. Генератор `field` последовательно выдаёт значения ключей словарей массива. Генератор `gen_random` последовательно выдаёт заданное количество случайных чисел в заданном диапазоне.

Содержание файла «gens.py»:

```
import random

# Генератор вычленения полей из массива словарей
# 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
    if (len(args) == 1):
        for item in items:
            val = item.get(args[0])
            if val is None:
                continue
            else:
                yield val
    else:
        for item in items:
            result = {}
            for el in item:
                if (el in args) and (item[el] is not None):
                    result[el] = item[el]
            if result != {}:
                yield result

# Генератор списка случайных чисел
# Пример:
# gen_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1
# Hint: реализация занимает 2 строки
def gen_random(begin, end, num_count):
    for i in range(num_count):
        yield random.randint(begin, end)
```

pass

Необходимо реализовать генератор

Содержание файла «ex_1.py»:

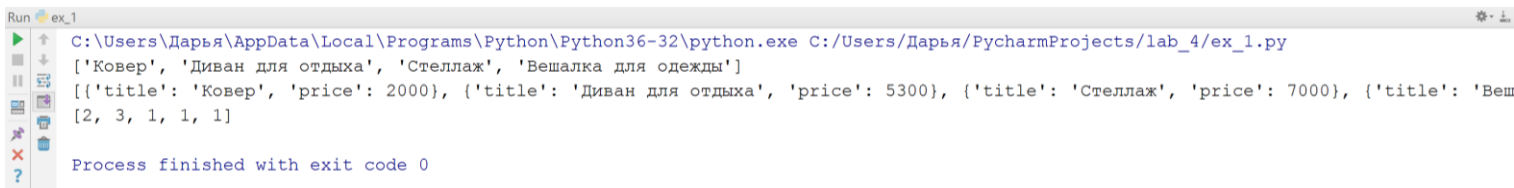
```
#!/usr/bin/env python3
from librip.gens import field, gen_random

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color':
'black'},
    {'title': 'Стеллаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color':
'white'}
]

# Реализация задания 1
print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))

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

Результат работы «ex_1.py»:



```
Run ex_1
C:\Users\Дарья\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/Дарья/PycharmProjects/lab_4/ex_1.py
['Ковер', 'Диван для отдыха', 'Стеллаж', 'Вешалка для одежды']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стеллаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}]
[2, 3, 1, 1, 1]
Process finished with exit code 0
```

Задача 2. Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной bool-параметр `ignore_case`, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен `False`. Итератор не должен модифицировать возвращаемые значения.

Содержание файла «iterators.py»:

```
# 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
        одинаковые строки в разном регистре
        # Например: ignore_case = True, Абв и АБВ разные строки
```

```

# ignore_case = False, Абв и АБВ одинаковые
строки, одна из них удалится
# По-умолчанию ignore_case = False
if kwargs.get('ignore_case') == True:
    self.ignore_case = True
else:
    self.ignore_case = False
self.data = iter(items)
self.res = set()
pass

def __next__(self):
    # Нужно реализовать __next__
    while True:
        item = self.data.__next__()
        if self.ignore_case:
            new_item = str(item).lower()
        else:
            new_item = item
        if new_item not in self.res:
            self.res.add(new_item)
            return item
    pass

def __iter__(self):
    return self

```

Содержание файла «ex_2.py»:

```

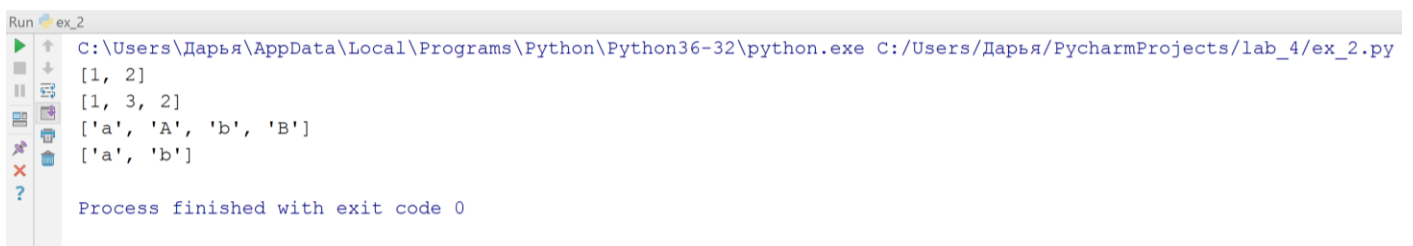
#!/usr/bin/env python3
from librip.gens import gen_random
from librip.iterators import Unique

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)

# Реализация задания 2
print(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique(['a', 'A', 'b', 'B'])))
print(list(Unique(['a', 'A', 'b', 'B'], ignore_case=True)))

```

Результат работы «ex_2.py»:



```

Run ex_2
C:\Users\Дарья\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/Дарья/PycharmProjects/lab_4/ex_2.py
[1, 2]
[1, 3, 2]
['a', 'A', 'b', 'B']
['a', 'b']
Process finished with exit code 0

```

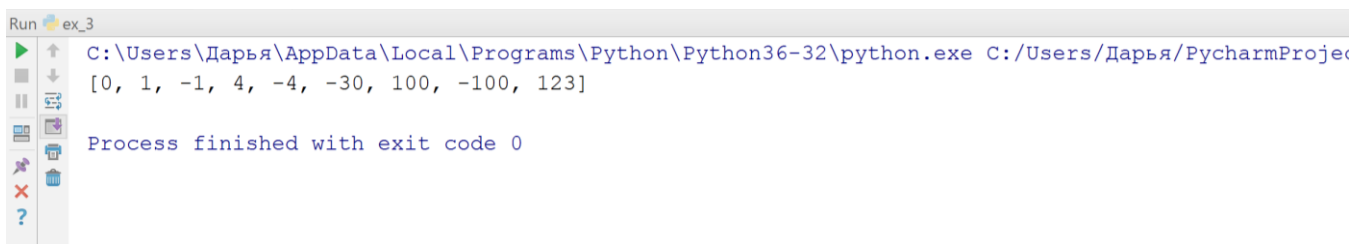
Задача 3. Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив, отсортированный по модулю. Сортировку осуществлять с помощью функции `sorted`.

Содержание файла «ex_3.py»:

```
#!/usr/bin/env python3

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
print(sorted(data, key=lambda x: abs(x)))
```

Результат работы «ex_3.py»:



Задача 4. 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, печатать результат и возвращать значение. Если функция вернула список (`list`), то значения должны выводиться в столбик. Если функция вернула словарь (`dict`), то ключи и значения должны выводиться в столбик через знак равно.

Содержание файла «decorators.py»:

```
# Здесь необходимо реализовать декоратор, print_result который
# принимает на вход функцию,
# вызывает её, печатает в консоль имя функции, печатает
# результат и возвращает значение
# Если функция вернула список (list), то значения должны
# выводиться в столбик
# Если функция вернула словарь (dict), то ключи и значения
# должны выводиться в столбик через знак равно
# Пример из ex_4.py:
# @print_result
# def test_1():
#     return 1
#
# @print_result
# def test_2():
#     return 'iu'
#
# @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()
#
# На консоль выведется:
# test_1
# 1
# test_2
# iu
# test_3
# a = 1
# b = 2
# test_4
# 1
# 2

def print_result(func_to_decorate):
    def decorated_func(*args, **kwargs):
        print(func_to_decorate.__name__)
        result = func_to_decorate(*args, **kwargs)
        if type(result) == type(list()):
            print("\n".join(map(str, result)))
        elif type(result) == type(dict()):
            print("\n".join(map(lambda x: "{} = {}".format(x[0],
x[1]), result.items())))
        else:
            print(result)
        return result
    return decorated_func

```

Содержание файла «ex_4.py»:

```

from librip.decorators import print_result

# Необходимо верно реализовать print_result
# и задание будет выполнено

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu'

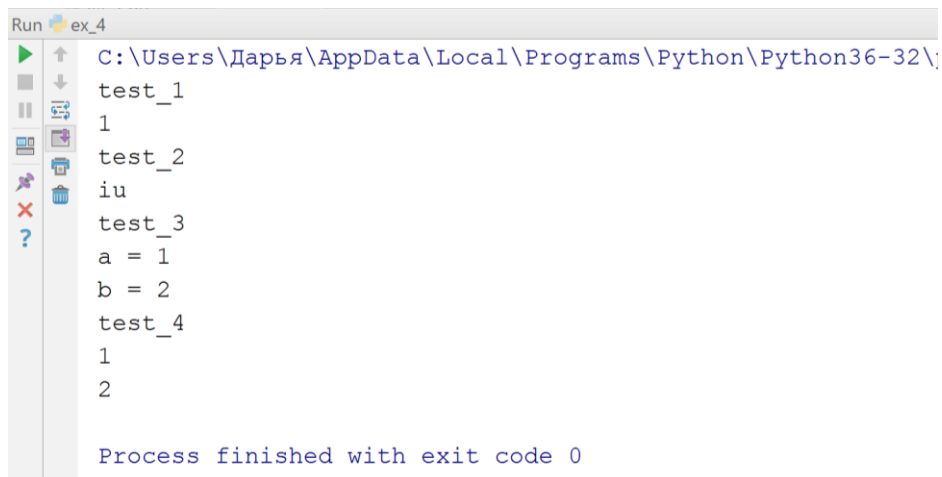
```

```
@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()
```

Результат работы «ex_4.py»:



```
Run ex_4
C:\Users\Дарья\AppData\Local\Programs\Python\Python36-32\
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

Задача 5. Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран.

Содержание файла «ctxmgrs.py»:

```
# Здесь необходимо реализовать
# контекстный менеджер timer
# Он не принимает аргументов, после выполнения блока он должен
вывести время выполнения в секундах
# Пример использования
# with timer():
#     sleep(5.5)
#
# После завершения блока должно вывестись в консоль примерно 5.5

from time import *
import contextlib
@contextlib.contextmanager
```

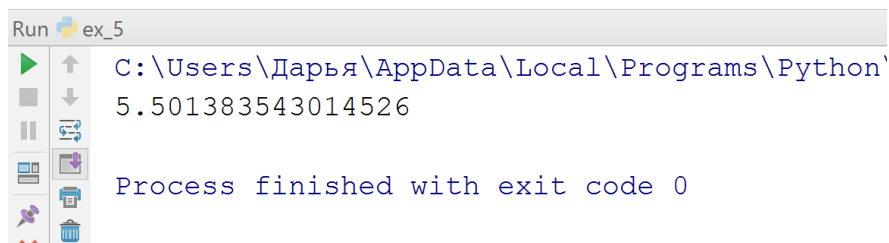
```
def timer():
    start = time()
    yield
    stop = time()
    print(stop - start)
```

Содержание файла «ex_5.py»:

```
from time import sleep
from librip.ctxmgrs import timer

with timer():
    sleep(5.5)
```

Результат работы «ex_5.py»:



Задача 6. В репозитории находится файл `data_light.json`. Он содержит облегченный список вакансий в России в формате `json`. Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д. В `ex_6.py` дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора `@print_result` печатается результат, а контекстный менеджер `timer` выводит время работы цепочки функций. Задача реализовать все 4 функции по заданию:

1. Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.

2. Функция `f2` должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами, нужно получить все специальности, связанные с программированием. Для фильтрации использовать функцию `filter`.

3. Функция `f3` должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: *Программист С# с опытом Python*. Для модификации использовать функцию `map`.

4. Функция `f4` должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: *Программист С# с опытом Python, зарплата 137287 руб.* Использовать `zip` для обработки пары специальность — зарплата.

Содержание файла «ex_6.py»:

```
#!/usr/bin/env python3
import json
from librip.decorators import print_result
from librip.iterators import Unique
from librip.gens import field, gen_random
from librip.ctxmgrs import timer

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске
path = './data_light.json'

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

# Далее необходимо реализовать все функции по заданию, заменив
# `raise NotImplemented`
# Важно!
# Функции с 1 по 3 должны быть реализованы в одну строку
# В реализации функции 4 может быть до 3 строк
# При этом строки должны быть не длиннее 80 символов

@print_result
def f1(arg):
    return list(sorted(Unique(field(arg, 'job-name'),
    ignore_case=True), key=lambda x: x.lower()))

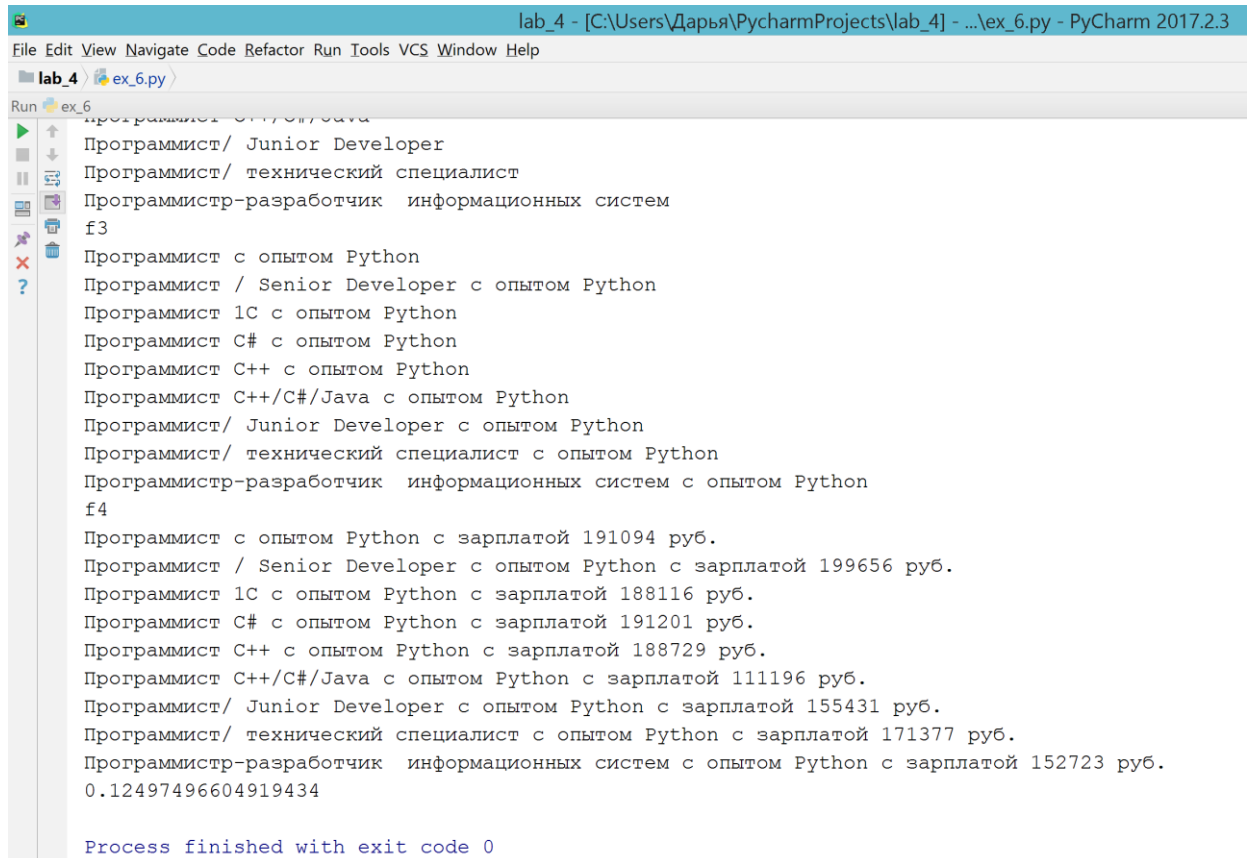
@print_result
def f2(arg):
    return list(filter(lambda x:
x.lower().startswith('программист'), arg))

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

@print_result
def f4(arg):
    return list('{} с зарплатой {} руб.'.format(profession,
salary) for (profession, salary)
    in zip(arg, gen_random(100000, 200000,
len(arg))))

with timer():
    f4(f3(f2(f1(data))))
```

Результат работы «ex_6.py»:



```
lab_4 - [C:\Users\Дарья\PycharmProjects\lab_4] - ...\\ex_6.py - PyCharm 2017.2.3
File Edit View Navigate Code Refactor Run Tools VCS Window Help
lab_4 \\ex_6.py
Run ex_6
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 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
f4
Программист с опытом Python с зарплатой 191094 руб.
Программист / Senior Developer с опытом Python с зарплатой 199656 руб.
Программист 1C с опытом Python с зарплатой 188116 руб.
Программист C# с опытом Python с зарплатой 191201 руб.
Программист C++ с опытом Python с зарплатой 188729 руб.
Программист C++/C#/Java с опытом Python с зарплатой 111196 руб.
Программист/ Junior Developer с опытом Python с зарплатой 155431 руб.
Программист/ технический специалист с опытом Python с зарплатой 171377 руб.
Программист-разработчик информационных систем с опытом Python с зарплатой 152723 руб.
0.12497496604919434

Process finished with exit code 0
```