



**Министерство науки и высшего образования Российской
Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана (национальный исследовательский
университет)» (МГТУ им. Н.Э. Баумана)**

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

**Отчёт по лабораторной работе №3
«Функциональные возможности языка Python»**

Выполнил: Студент группы ИУ5-31Б – Толстолуцкий Д.А.

Москва, 2022

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}

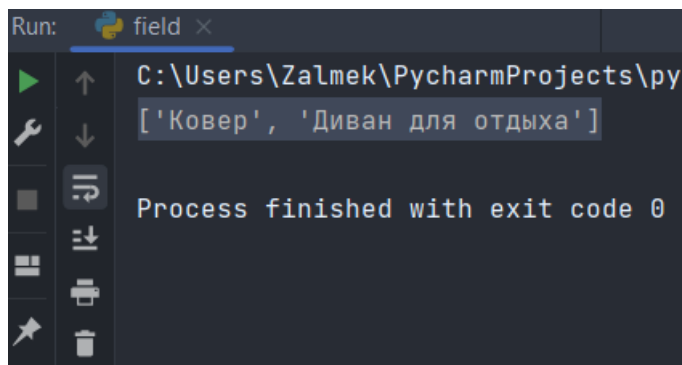
def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
```

Текст программы:

```
def field(dict_arr, *args):
    if len(args) == 1:
        list = []
        for i in range(len(dict_arr)):
            if dict_arr[i].get(args[0]) is not None:
                list.append(dict_arr[i].get(args[0]))
        return list
    else:
        for i in range(len(dict_arr)):
            dict = {}
            for j in range(len(args)):
                if dict_arr[i].get(args[j]) is not None:
                    dict[args[j]] = (dict_arr[i].get(args[j]))
            return dict

def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'},
    ]
    print(field(goods, 'title'))
main()
```

Результаты выполнения программы:



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:

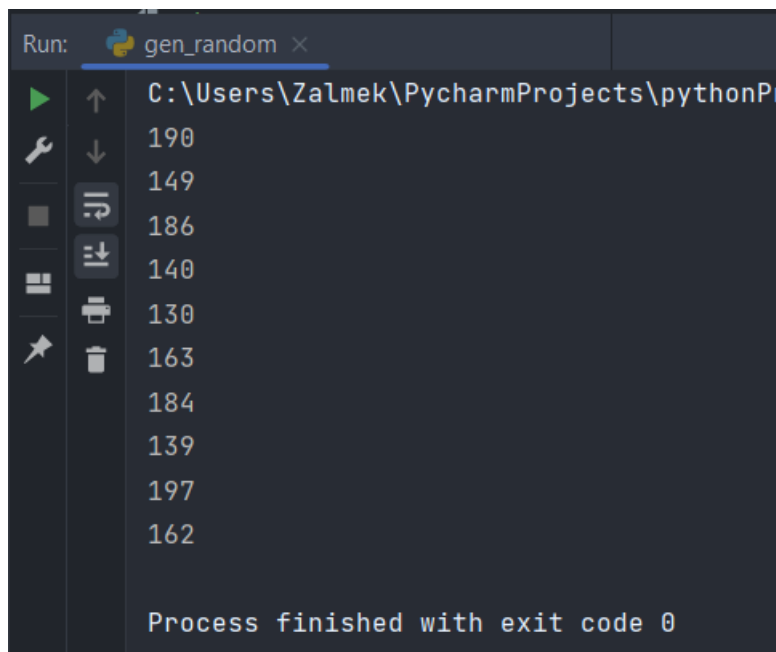
```
# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
def gen_random(num_count, begin, end):
    pass
# Необходимо реализовать генератор
```

Текст программы:

```
import random

def gen_random(num_count, begin, end):
    list=[]
    for i in range(num_count):
        list.append(random.randint(begin,end))
    return list
def main():
    list = gen_random(10,130,200)
    for i in list:
        print(i)
```

Результаты выполнения программы:



```
Run: gen_random x
C:\Users\Zalmek\PycharmProjects\pythonPr
190
149
186
140
130
163
184
139
197
162

Process finished with exit code 0
```

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
```

Текст программы:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.counter = -1
        if kwargs.get('ignore_case'):
            self.unique = list()
            for i in range(len(items)):
                items[i].lower()
                if items[i] not in self.unique:
                    self.unique.append(items[i])
        else:
            self.unique = list()
            for i in range(len(items)):
                if items[i] not in self.unique:
                    self.unique.append(items[i])

    def __next__(self):
        if self.counter < len(self.unique)-1 and self.unique[self.counter] is
        not None:
            self.counter += 1
            return self.unique[self.counter]
        else:
```

```

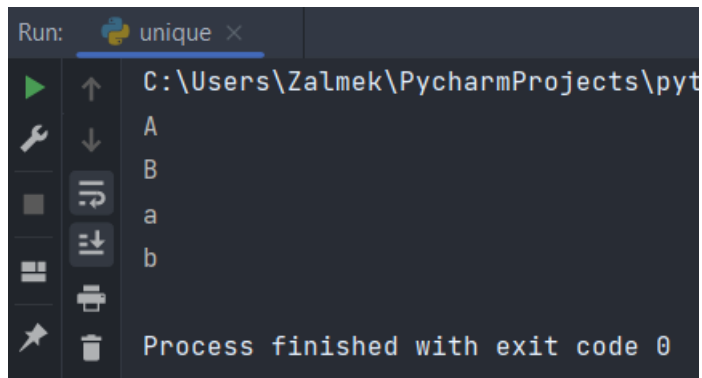
        raise StopIteration

    def __iter__(self):
        return self

def main():
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    un = Unique(data)
    un = sorted(un)
    for i in un:
        print(i)

```

Результат выполнения программы:



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)

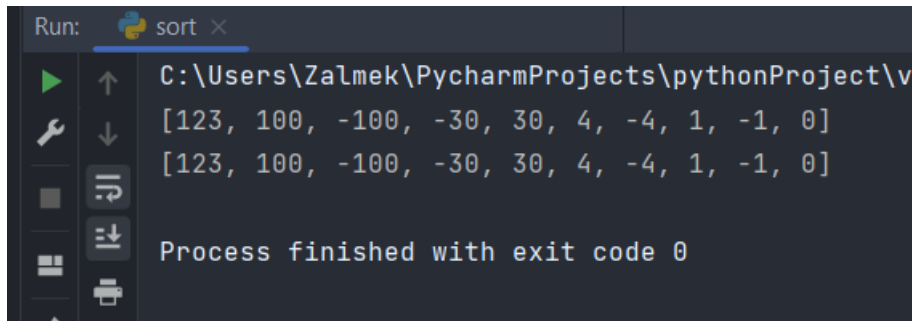
    result_with_lambda = ...
    print(result_with_lambda)

```

Текст программы:

```
print(sorted([4, -30, 30, 100, -100, 123, 1, 0, -1, -4], key=abs,
reverse=True))
print(sorted([4, -30, 30, 100, -100, 123, 1, 0, -1, -4], key=lambda a:
abs(a), reverse=True))
```

Результат выполнения программы:



```
Run: sort ×
C:\Users\Zalmek\PycharmProjects\pythonProject\ve
[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
```

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
```

Текст программы

```
def print_result(fun):  
    print(fun.__name__)  
  
    def wrapper_print_result(*args, **kwargs):  
        result = fun(*args, **kwargs)  
        if type(result) == dict:  
            for k, v in result.items():  
                print("%s=%s" % (k, v))  
            return result  
        elif type(result) == list:  
            for i in result:  
                print(i)  
            return result  
        else:  
            print(result)  
            return result  
    return wrapper_print_result  
  
@print_result  
def test_1(integer):  
    return integer  
  
@print_result  
def test_2(str):  
    return str  
  
@print_result  
def test_3(dict):  
    return dict  
  
@print_result  
def test_4(list):  
    return list
```

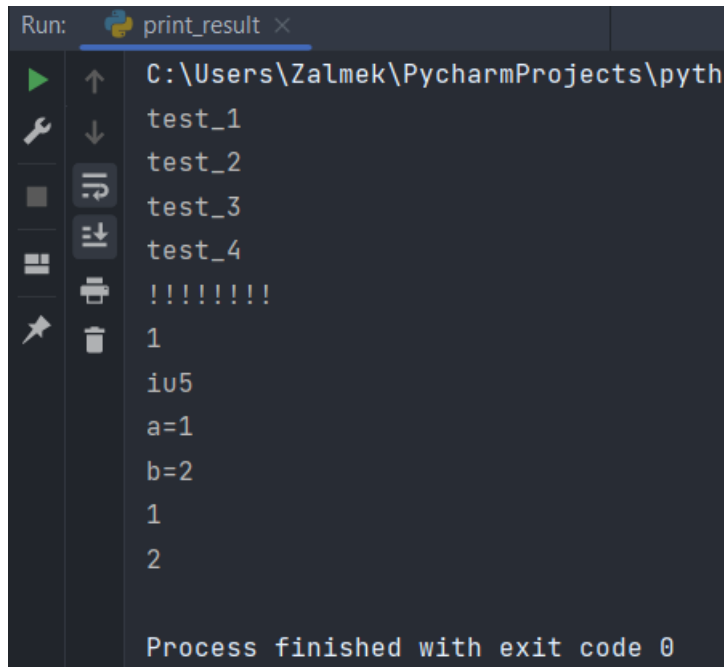


```

if __name__ == '__main__':
    print('!!!!!!!')
    test_1(1)
    test_2("iu5")
    test_3({'a': 1, 'b': 2})
    test_4([1, 2])

```

Результат выполнения программы:



```

Run: print_result x
C:\Users\ZalmeK\PycharmProjects\pyth
test_1
test_2
test_3
test_4
!!!!!!!
1
iu5
a=1
b=2
1
2
Process finished with exit code 0

```

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).

Текст программы:

```

import datetime
from contextlib import contextmanager
from time import sleep, perf_counter
import time

def waiter():
    for i in range(1, 3):
        sleep(i)

```

```

class cm_timer_2(object):
    def __enter__(self):
        self.t = time.clock()
        return self

    def __exit__(self, type, value, traceback):
        self.t = time.clock() - self.t

@contextmanager
def cm_timer_1() -> float:
    start = perf_counter()
    yield lambda: perf_counter() - start

def main():
    with cm_timer_1() as t:
        sleep(2)
    print(t())

```

Результат выполнения программы:

```

Run: cm_timer x
C:\Users\Zalmek\PycharmProjects\python
2.0060539
Process finished with exit code 0

```

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:

```

import json
import gen_random
from lab3 import cm_timer, unique
from lab3.field import field
from lab3.print_result import print_result

path = r"C:\Users\Zalmek\PycharmProjects\pythonProject\lab3\my.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан
при запуске сценария

with open(path, "r", encoding="UTF-8") 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):
    from lab3 import unique
    return list(sorted(unique.Unique(field(arg, "job-name"))))

@print_result
def f2(arg):
    return list(filter(lambda x: x[0:11] == "программист", arg))

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

```

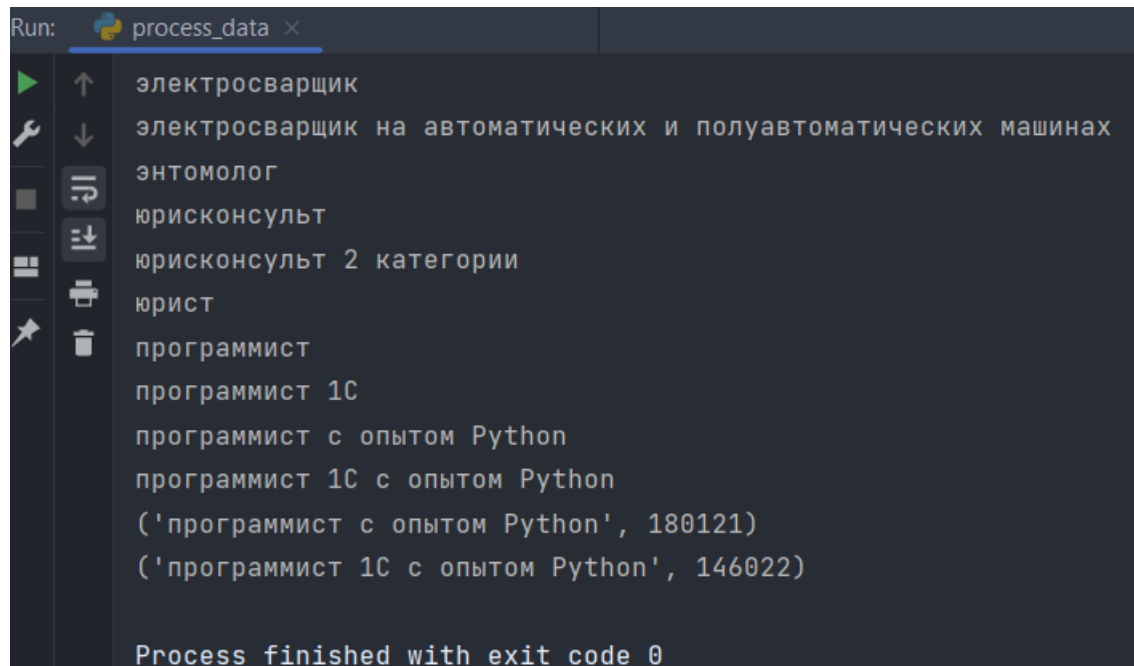
```

@print_result
def f4(arg):
    return list(zip(arg, gen_random.gen_random(len(arg), 100000, 200000)))
    #return list(map(lambda x: x + ", Зарплата " +
str(gen_random.gen_random(1, 100000, 200000)) + " руб.", arg))

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

```

Результат выполнения программы:



```

Run: process_data x
↑ электросварщик
↓ электросварщик на автоматических и полуавтоматических машинах
↺ энтомолог
↻ юристконсульт
↻ юристконсульт 2 категории
↻ юрист
↻ программист
↻ программист 1С
↻ программист с опытом Python
↻ программист 1С с опытом Python
↻ ('программист с опытом Python', 180121)
↻ ('программист 1С с опытом Python', 146022)

Process finished with exit code 0

```