Московский государственный технический университет имени Н. Э.Баумана

Факультет Радиотехнический

Кафедра ИУ5 Системы обработки информации и управления

# Отчет по лабораторной работе №3 по курсу Базовые компоненты интернет-технологий

# "Функциональные возможности языка Python"

8
(количество листов)

Вариант № **17**

Исполнитель

студент группы РТ5-31б _____ Павлов С.Д.

"20"сентября 2021 г.

Проверил

Доцент кафедры ИУ5 _____ Гапанюк Ю.Е.

"___" _____ 2021 г.

Москва, 2021 г.

# Задание

## 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 количествово 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.

## 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`

## 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.

```
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.

## 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.

## 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 выводить в столбик через знак равенства.

## 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. Использовать zip для обработки пары специальность — зарплата.

## Текст программ

### Файл field.py:

```
#Последовательная выдача ключей из словаря:
elements = [
    {'title': None, 'color': 'red', 'title': 'some_title'},
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

def fild_generator(list, *args):
    #Если аргументов несколько:
    if len(args) > 1:
        #Для каждого словаря:
        for i0 in list:
            result = {}
            #Поиск ключа аналогичного аргументу с не None значением:
            for i1 in args:
                value = i0.get(i1, None)
                if value != None: result[i1] = value
            if result != {}: yield result
    #Если аргумент 1
    elif len(args) > 0:
         for i0 in list:
             value = i0.get(args[0], None)
             if value != None: yield value
    else: raise Exception('No arguments in fild_generator!')

#Проверка:
if __name__ == '__main__':
    f = fild_generator(elements, 'title', 'price', 'color')
    for i in f:
        print(i)
    print('\n\n')
    f = fild_generator(elements, 'title')
    for i in f: print(i)
```

### Файл gen_random.py:

```
#Выдача рандомного числа:
from random import randint

def random_generator(Number, A, B):
    for i in range(Number): yield randint(A, B)
#Проверка:
if __name__ == '__main__':
    G = random_generator(20, 1, 100)
    for i in G: print(i, end = '; ')
```

Файл unique.py:

```
#Пропуск дубликатов:
import inspect
class Unique:
    def __init__(self, Item, **kwargs):
        if type(Item) != list and inspect.isgenerator(Item) == False:
            raise Error('Некорректный тип аргумента итератора! (' + \
                        str(type(Item)) + ', а требуются: list/generator)')
        if type(Item) == list: self.arg_list = True
        else:                  self.arg_list = False

        self.ignore_case = kwargs.get('ignor_case', None)
        if self.ignore_case == None: self.ignore_case = False
        self.elements_list = []
        self.item = Item
        self.i = 0

    def __next__(self):
        #Если на входе имеется лист:
        if self.arg_list == True:
            if len(self.item) <= self.i: raise StopIteration
            Got_item = self.item[self.i]
            self.i+=1
        #Если использовался генератор:
        else:
            try: Got_item = next(self.item)
            except: raise StopIteration

        if Got_item in self.elements_list: return
        elif not self.ignore_case:#
            self.elements_list.append(Got_item)
            return Got_item
        elif type(Got_item) == str:
            found = False
            for i2 in self.elements_list:
                if Got_item.upper() == i2 or Got_item.lower() == i2:
                    found = True; break
            if not found:
                self.elements_list.append(Got_item)
                return Got_item
        else:
            self.elements_list.append(Got_item)
            return Got_item

    def __iter__(self):
        return self

#Проверка:
if __name__ == '__main__':
    from Python_Lab3_2 import random_generator
    listik = [1, 2, 3, 'HI', 'hi', 'халява', 'A', 'a']
    G = random_generator(10,1,5)
    d1 = Unique(G, ignor_case = True)
    d2 = Unique(listik, ignor_case = True)
    for i in d1:
        print(i)
    print('\n\n')
    for i in d2:
        print(i)
```

Файл sort.py:

```
#Сортировка по мудулю:
data = [1, -1, 2, -2, 3, -3, 4, -4, 5, -5]
import math

if __name__ == '__main__':
    print(sorted(data, key = abs, reverse = True), end = '\n\n')

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

Файл print_result.py:

```
#Декоратор вывода:
def print_result(function):
    def wrapper(x):
        print('Исполняется: ' + str(function.__name__) + '\n')
        result = function(x)
        if isinstance(result, list):
            for i in result:
                 if isinstance(i, zip): \
                 print(str(list(i)).strip('()[]').replace("'",""))
                 else: print(i)
        elif isinstance(result, dict):
            res_touple = result.fromkeys
            for i in result: print(res_touple[0] + ' = ' + res_touple[1])
        else: print(result)
        return result
    return wrapper

def Test(x):
    print('Идет исполнение Test')
    return x

#Проверка:
if __name__ == '__main__':
   test = print_result(Test)
   A = test(10)
   print('\n' + str(A))
```

Файл cm_timer.py:

```
#Расчет времени исполнения:
from contextlib import contextmanager
from time import time; from time import sleep
#Для нормального человеческого округления:
def normal_people_round(number, ndigits=0):
    ndigits += 1
    n = round(number, ndigits)*(10**ndigits)
    m = n % 10
    n -= m
    if m >= 5:
        n += 10
    n /= (10**ndigits)
    return n

#С использованием библиотеки:
@contextmanager
def time_counter(ndigits):
```

```
    try:
        time_start = time()
        yield
    except:
        print("We had an error!")
    finally:
        time_end = time()
        proces_time = time_end - time_start
        print('Затрачено: ', normal_people_round(proces_time, ndigits), ' с')

#С использованием класса:
class Timer():
    def __init__(self, ndigits):
        self.time_start = 0
        self.ndigits = ndigits

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

    def __exit__(self, exc_type, exc_value, exc_traceback):
        proces_time = time() - self.time_start
        print('\nЗатрачено: ', normal_people_round(proces_time, self.ndigits),
              ' с')

def test(n=5):
        times = 0
        while True:
            print("f")
            times += 1
            if times > n: break

if __name__ == "__main__":
    with time_counter(8):
       sleep(1.5)

    with Timer(8):
        sleep(1.5)
```

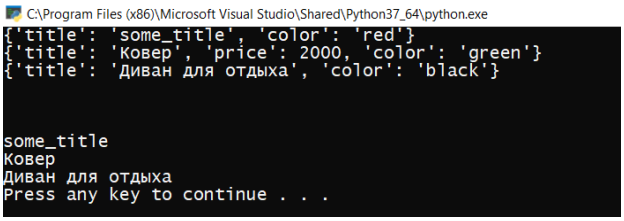
Файл process_data.py:

```
#Основное задание:
import json
import sys
from Python_Lab3_5 import print_result
from Python_Lab3_3 import Unique
from Python_Lab3_6 import Timer
from Python_Lab3_2 import random_generator
path = 'Data.json'

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

def f1(arg):
    return [i.capitalize() for i in Unique(arg, ignor_case = True) if i != None]

def f2(arg):
    return list(filter(lambda x: "Программист" in x or "программист" in x, arg))

def f3(arg):
    return [i + ' с опытом Python' for i in arg]
```

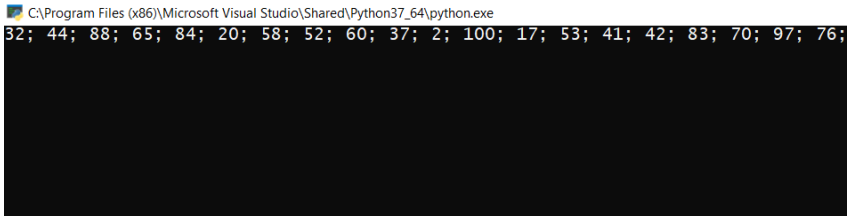
```
@print_result
def f4(arg):
    result = []
    for i in arg:
      result.append(zip([i], ['зарплата: ' +
                    str(list(random_generator(1, 100000, 200000))).strip('[]')]))
    return result

if __name__ == '__main__':
    with Timer(8):
        f4(f3(f2(f1(data))))
```

# Экранная форма с примерами выполнения программы

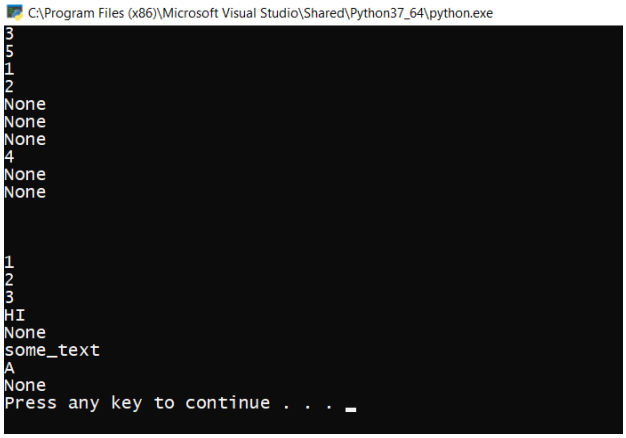
Файл field.py



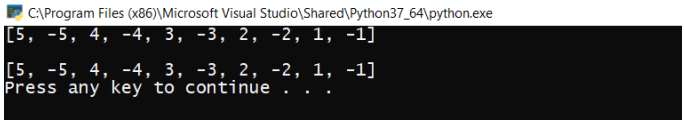

Файл gen_random.py:



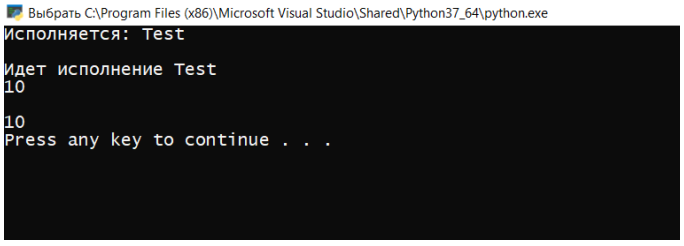

Файл unique.py



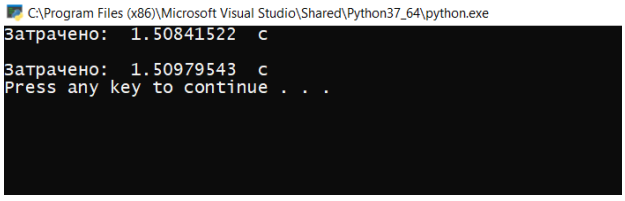

Файл sort.py:




Файл print_result.py:




Файл cm_timer.py




Файл process_data.py: