**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: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Борисова К.А. |  | Гапанюк Ю.Е. |

Москва, 2024 г.

**Постановка задачи:**

Задача 1 (файл field.py)

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря. 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.

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.

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.

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. Необходимо решить задачу двумя способами:С использованием lambda-функции, 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. 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.

**Текст программы и анализ результатов:**

Задача 1 (файл field.py)

def field(*items*, \**args*):

*assert* len(args) > 0

*for* item *in* items:

*if* len(args) == 1:

            key = args[0]

*if* key in item and item[key] is not None:

*yield* item[key]

*else*:

            filtered\_item = {key: item[key] *for* key *in* args *if* key in item and item[key] is not None}

*if* filtered\_item:

*yield* filtered\_item

*# Пример использования*

goods = [

    {'title': 'Ковер', 'price': 2000, 'color': 'green'},

    {'title': 'Диван для отдыха', 'color': 'black'}

]

*# Проверка для одного аргумента*

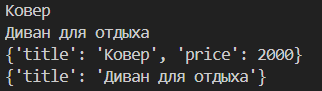
*for* title *in* field(goods, 'title'):

    print(title)

*# Проверка для двух аргументов*

*for* item *in* field(goods, 'title', 'price'):

    print(item)



Задача 2 (файл gen\_random.py)

*import* random

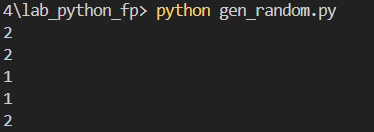
def gen\_random(*count*, *minim*, *maxim*):

*for* \_ *in* range(*count*):

*yield* random.randint(*minim*, *maxim*)

*for* num *in* gen\_random(5, 1, 3):

    print(num)



Задача 3 (файл unique.py)

class Unique(object):

    def \_\_init\_\_(*self*, *items*, \*\**kwargs*):

*self*.items = *items*

*self*.ignore\_case = *kwargs*.get('ignore\_case', False)

*self*.seen = set()

*self*.iterator = iter(*items*)

    def \_\_next\_\_(*self*):

*while* True:

*try*:

                item = next(*self*.iterator)

                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

*except* StopIteration:

*raise* StopIteration

    def \_\_iter\_\_(*self*):

*return* *self*

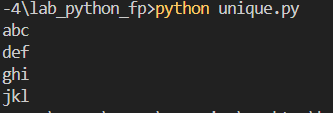
*if* \_\_name\_\_ == "\_\_main\_\_":

    data = ['abc', 'def', 'ABC', 'ghi', 'DEF', 'jkl']

    unique\_items = Unique(data, *ignore\_case*=True)

*for* item *in* unique\_items:

        print(item)



Задача 4 (файл sort.py)

*# with Lamda*

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

print(data)

sorted\_data = sorted(data, *key*=lambda *x*: abs(*x*), *reverse*=True)

print(sorted\_data)

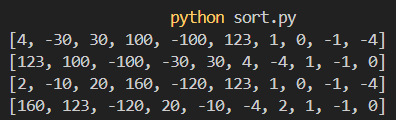
*# without Lamda*

data = [2, -10, 20, 160, -120, 123, 1, 0, -1, -4]

print(data)

sorted\_data = sorted(data, *key*=abs, *reverse*=True)

print(sorted\_data)



Задача 5 (файл print\_result.py)

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

@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\_\_':

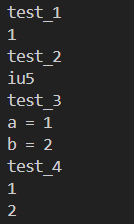
    print('!!!!!!!!')

    test\_1()

    test\_2()

    test\_3()

    test\_4()



Задача 6 (файл cm\_timer.py)

*import* time

*from* contextlib *import* contextmanager

class cm\_timer\_1:

    def \_\_enter\_\_(*self*):

*self*.start\_time = time.time()

*return* *self*

    def \_\_exit\_\_(*self*, *exc\_type*, *exc\_val*, *exc\_tb*):

        end\_time = time.time()

        elapsed\_time = end\_time - *self*.start\_time

        print(f"time: {elapsed\_time:.1f}")

@contextmanager

def cm\_timer\_2():

    start\_time = time.time()

*try*:

*yield*

*finally*:

        end\_time = time.time()

        elapsed\_time = end\_time - start\_time

        print(f"time: {elapsed\_time:.1f}")

*if* \_\_name\_\_ == "\_\_main\_\_":

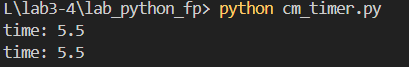
*from* time *import* sleep

*with* cm\_timer\_2():

        sleep(5.5)

*with* cm\_timer\_2():

        sleep(5.5)



Задача 7 (файл process\_data.py)

*import* json

*import* sys

*import* time

*from* contextlib *import* contextmanager

path = "C:/Users/ann18/OneDrive/Desktop/bmstu/3sem/PCPL/lab3-4/data\_light.json"

*with* open(path, *encoding*="utf8") *as* f:

    data = json.load(f)

@contextmanager

def cm\_timer\_1():

    start\_time = time.time()

*yield*

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    print(f"time: {elapsed\_time}")

def print\_result(*func*):

    def wrapper(\**args*, \*\**kwargs*):

        result = *func*(\**args*, \*\**kwargs*)

        print(result)

*return* result

*return* wrapper

@print\_result

def f1(*arg*):

*return* sorted(list(set([job['job-name'].lower() *for* job *in* *arg*])))

@print\_result

def f2(*arg*):

*return* list(filter(lambda *job*: *job*.startswith('программист'), *arg*))

@print\_result

def f3(*arg*):

*return* list(map(lambda *job*: *job* + ', с опытом Python', *arg*))

@print\_result

def f4(*arg*):

    salary = [str(i) *for* i *in* range(100000, 200001)]

*return* list(zip(*arg*, salary))

*if* \_\_name\_\_ == '\_\_main\_\_':

*with* cm\_timer\_1():

        f4(f3(f2(f1(data))))