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 №5
«Разработка простого бота для Telegram с использованием языка Python.»

Выполнил:
студент группы ИУ5-31Б
Шилина А.Ю.
Подпись и дата: 23.09.24

Проверил:
преподаватель каф. ИУ5
Гапанюк Е.Ю.
Подпись и дата:

ОГЛАВЛЕНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

Задание: Задание лабораторной работы состоит в реализации нескольких задач, требующих разработки различных генераторов и итераторов на языке Python.

2. РАЗРАБОТКА АЛГОРИТМА

Задача 1: Генератор field

Описание задачи: необходимо реализовать генератор field, который будет последовательно выдавать значения ключей словаря, передаваемых в виде списка словарей. Генератор должен учитывать разные количества аргументов, фильтруя значения по заданным ключам.

Текст программы:

```
def field(items, *args):
    assert len(args) > 0 #обязательно верно
    for item in items:
        if len(args) == 1:
            # 1 АРГУМЕНТ = ЗНАЧЕНИЕ
            value = item.get(args[0])
            if value is not None:
                yield value #как ретурн но для генераторов, типа большой
набор значений который надо прочитать 1 раз
        else:
            # НЕСКОЛЬКО АРГ=СЛОВАРЬ
            result = {key: item.get(key) for key in args if item.get(key) is
not None}
            if result:
                yield result
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
```

Результат выполнения:

```
C:\Users\Francium\PycharmProjects\LAB2-3\.venv\bin\python.exe C:\Users
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]

Process finished with exit code 0
```

Задача 2: Генератор gen_random

Описание задачи: необходимо реализовать генератор, который будет

выдавать случайные числа в заданном диапазоне определенное количество раз.

Текст программы:

```
from random import randint
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
pl=[]
def gen_random(num_count, begin, end):
    if num_count > 0:
        a = randint(begin, end)
        pl.append(a)
        gen_random(num_count - 1, begin, end)

num_count, begin, end = map(int, input("Введите количество, минимум и
максимум через пробел: ").split())

gen_random(num_count, begin, end)
print(pl)
```

Результат выполнения:

```
C:\Users\Francium\PycharmProjects\LAB2-3\.venv\bin\python.exe C:\Use
Введите количество, минимум и максимум через пробел: 6 -100 100
[79, 48, -7, 1, 40, -63]

Process finished with exit code 0
```

Задача 3: Итератор Unique

Описание задачи: необходимо реализовать итератор, который будет пропускать дубликаты элементов в списке или генераторе. Итератор также должен учитывать параметр ignore_case, чтобы игнорировать регистр при сравнении строк.

Текст программы:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)
        self.seen = set() #встреченные элементы храним туут
        self.iterator = iter(items)
    def __iter__(self): #чтоб вернуть итератором для циклов
        return self
    def __next__(self):
        while True: #смотрим на элемент след. проверка есть ли в self.seen
сли нет то добавляем
            item = next(self.iterator)
            check_item = item.lower() if self.ignore_case and
isinstance(item, str) else item
            if check_item not in self.seen:
                self.seen.add(check_item)
                return item
if __name__ == "__main__":
```

```
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data):
        print(item)

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(list(Unique(data)))  # ['a', 'A', 'b', 'B']
    print(list(Unique(data, ignore_case=True)))  # ['a', 'b']
#итератор типа место где находится элемент
```

Результат выполнения:

```
C:\Users\Francium\PycharmProjects\LA
1
2
['a', 'A', 'b', 'B']
['a', 'b']

Process finished with exit code 0
```

Задача 4: Сортировка по модулю Sort

Описание задачи: необходимо отсортировать массив чисел по модулю в порядке убывания.

Текст программы:

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

if __name__ == '__main__':
    def abs_key(x):
        return abs(x)
    result = sorted(data, key=abs_key, reverse=True)
    print(result)


    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)
#key — правило сортировки.
#lambda x: abs(x) —берем x из списка и возвращаем (abs(x)).
```

Результат выполнения:

```
C:\Users\Francium\PycharmProjects\LAB2-3\.ven
[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: Декоратор print_result

Описание задачи: необходимо реализовать декоратор, который будет выводить результаты работы функции на экран, включая форматированный вывод для списков и словарей.

Текст программы:

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

        print(func.__name__)

        if isinstance(result, list): #выводим каждый элемент списка
            for item in result:
                print(item)
        elif isinstance(result, dict):# словарик ключ=знач
            for key, value in result.items():
                print(f"{key} = {value}")
        else:
            print(result) #все остальное как есть выводим
        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__':
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат выполнения:

```
C:\Users\Francium\PycharmProjects\L
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

Задача 6: Контекстные менеджеры cm_timer_1 и cm_timer_2

Описание задачи: необходимо реализовать два контекстных менеджера, которые будут измерять время выполнения блока кода.

Текст программы:

```
import time

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()  #начальное время
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time  # разницу времени
        print(f"time: {elapsed_time:.2f}")


from contextlib import contextmanager

@contextmanager #контекстный менеджер
def cm_timer_2():
    start_time = time.time()  # начало измер
    try:
        yield
    finally:
        elapsed_time = time.time() - start_time
        print(f"time: {elapsed_time:.2f}")
import time

if __name__ == "__main__":
    print("Using cm_timer_1:")
    with cm_timer_1():
        time.sleep(2.5)

    print("\nUsing cm_timer_2:")
    with cm_timer_2():
        time.sleep(3.0)

#генератор позволяет приостанавливать выполнение функции и
# возвращать значение, сохраняя состояние выполнения,
```

Результат выполнения:

```
C:\Users\Francium\PycharmProjects\
Using cm_timer_1:
time: 2.50

Using cm_timer_2:
time: 3.00

Process finished with exit code 0
```

Задача 7: Обработка данных из файла process_data

Описание задачи: необходимо реализовать цепочку функций для обработки данных из JSON файла, применяя различные инструменты, такие как сортировка, фильтрация и генерация зарплат.

Текст программы:

```
import json
import sys
from random import randint
from cm_timer import cm_timer_1
from print_result import print_result
from unique import Unique

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

@print_result
def f1(arg):
    # уник проф игнорируем регист
    return sorted(Unique([job["job-name"] for job in arg], ignore_case=True),
key=str.lower)

@print_result
def f2(arg):
    # программист, начинается с!!
    return list(filter(lambda x: x.lower().startswith("программист"), arg))

@print_result
def f3(arg):
    #опыт питон добавить строчку
    return list(map(lambda x: f"{x} с опытом Python", arg))

@print_result
def f4(arg):
    # зарплата
    salaries = [randint(100_000, 200_000) for _ in arg]
    return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg,
salaries)]
# зип объединяет список профессий и список зарплаты создавая новый типо
"{job},{salary} руб."
if __name__ == "__main__":
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Результат выполнения:

//не буду результат работы f1 т.к. там достаточно много записей.

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-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
Программистр-разработчик  информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 146898 руб.
Программист / Senior Developer с опытом Python, зарплата 117956 руб.
Программист 1С с опытом Python, зарплата 197367 руб.
Программист C# с опытом Python, зарплата 181543 руб.
Программист C++ с опытом Python, зарплата 137380 руб.
Программист C++/C#/Java с опытом Python, зарплата 103066 руб.
Программист/ Junior Developer с опытом Python, зарплата 163029 руб.
Программист/ технический специалист с опытом Python, зарплата 145081 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 189837 руб.
time: 0.01
```