Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2.3
дисциплины «Программирование на Python»
Вариант 2

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Работа со строками в языке Python

Цель: приобретение навыков по работе со строками при написании программ с помощью языка программирования Python версии 3.x.

Ход работы

Пример 1. *Дано предложение. Все пробелы в нем заменить символом «_».*

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


if __name__ == '__main__':
    s = input("Введите предложение: ")
    r = s.replace(' ', '_')
    print("Предложение после замены:", r)
```

Рисунок 1. Программа для решения поставленной задачи

```
Введите предложение: text text text
Предложение после замены: text_text_text
```

Рисунок 2. Результат выполнения программы

Пример 2. *Дано слово. Если его длина нечетная, то удалить среднюю букву, в противном случае – две средние буквы.*

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

if __name__ == '__main__':
    word = input("Введите слово: ")

    idx = len(word) // 2
    if len(word) % 2 == 1:
        r = word[:idx] + word[idx+1:]
    else:
        r = word[:idx-1] + word[idx+1:]

    print(r)
```

Рисунок 3. Программа для решения поставленной задачи

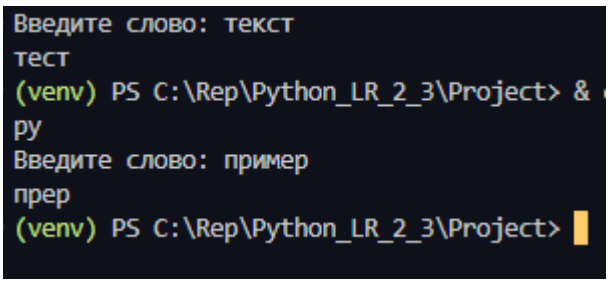

Рисунок 4. Результат выполнения программы

Пример 3. *Дана строка текста, в котором нет начальных и конечных пробелов. Необходимо изменить ее так, чтобы длина строки стала равна заданной длине (предполагается, что требуемая длина не меньше исходной). Это следует сделать путем вставки между словами дополнительных пробелов. Количество пробелов между отдельными словами должно отличаться не более чем на 1.*

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys


if __name__ == '__main__':
    s = input("Введите предложение: ")
    n = int(input("Введите длину: "))

    if len(s) >= n:
        print("Заданная длина должна быть больше длины предложения", file=sys.stderr)
        exit(1)

words = s.split(' ')
if len(words) < 2:
    print("Предложение должно содержать несколько слов", file=sys.stderr)
    exit(1)

delta = n
for word in words:
    delta -= len(word)

w, r = delta // (len(words) - 1), delta % (len(words) - 1)
lst = []
for i, word in enumerate(words):
    lst.append(word)
    if i < len(words) - 1:
        width = w
        if r > 0:
            width += 1
            r -= 1

    if width > 0:
        lst.append(' ' * width)

print(''.join(lst))
```

```
Введите длину: 23
текст    текст    текст
(venv) PS C:\Rep\Python_LR_2_3\Project> &
py
Введите предложение: текст текст
Введите длину: 15
текст     текст
```

Рисунок 5. Результат выполнения программы

2. Составить программу, которая печатает заданное слово, начиная с последней буквы.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

if __name__ == '__main__':
    s = input("Введите строку: ")
    print("Результат: ", s[-1:-len(s)-1:-1])
```

```
Введите строку: apple
Результат:  elppa
(venv) PS C:\Rep\Python_LR_2_3\Project> &
l1.py
Введите строку: program
Результат:  margorp
```

Рисунок 6. Результат выполнения программы

2. Дано слово. Проверить, является ли оно палиндромом (палиндром читается одинаково в обоих направлениях, например «потоп»).

```
Введите слово: потоп
Слово является палиндромом
(venv) PS C:\Rep\Python_LR_2_3\Project> &
l2.py
Введите слово: дом
Слово не является палиндромом
```

Рисунок 7. Результат выполнения программы

2. Дано слово.

- Удалить из него первую из букв о, если такая буква есть.
- Удалить из него последнюю из букв л, если такая буква есть.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


if __name__ == '__main__':
    s = input("Введите слово: ")
    if 'o' in s:
        idx = s.find('o')
        s = s[:idx] + s[idx+1:]
```

```
    if 'л' in s:
        idx = s.rfind('л')
        s = s[:idx] + s[idx+1:]
print(f"Результат: {s}")
```

```
Введите слово: полнолуние
Результат: плноуние
```

Рисунок 8. Результат выполнения программы

2. Дан текст. Найти наибольшее количество идущих подряд одинаковых символов.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


if __name__ == '__main__':
    max = 0
    cnt = 1
    st = 'ssaaadrgggggg trgddddmj'
    for i in range(len(st)-1):
        if st[i] == st[i+1]:
            cnt += 1
        elif max < cnt:
            max = cnt
        else:
            cnt = 1

    print(f"Исходная строка: {st}")
    print(f"Наибольшее количество идущих подряд символов: {max}")
```

```
Исходная строка: aaavaddmmdffffffa affgtfffffffghj
Наибольшее количество идущих подряд символов: 7
(venv) PS C:\Rep\Python_LR_2_3\Project> & c:/Rep/Python_LR_2_3/
14.py
Исходная строка: ssaaadrgggggg trgddddmj
Наибольшее количество идущих подряд символов: 5
```

Рисунок 9. Результат выполнения программы

Ответы на контрольные вопросы:

1. Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме

2. Строки в апострофах и в кавычках. Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы

позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование

3. Сложение строк: можно объединять строки с помощью оператора "+" или метода "join". Умножение строк: можно повторять строку заданное количество раз с помощью оператора "*". Длина строки: можно получить длину строки с помощью функции "len()". Изменение регистра: можно преобразовать строку в верхний регистр с помощью метода "upper()" или в нижний регистр с помощью метода "lower()". Поиск подстроки: можно найти индекс первого вхождения подстроки в строку с помощью метода "find()" или "index()". Замена подстроки: можно заменить все вхождения подстроки в строке на другую подстроку с помощью метода "replace()". Разделение строки: можно разделить строку на подстроки по определенному разделителю с помощью метода "split()". Удаление пробельных символов: можно удалить пробельные символы в начале и конце строки с помощью метода "strip()". Форматирование строк: можно использовать методы форматирования для вставки значений переменных в строку, такие как "format()" или f-строки. Проверка содержания: можно проверить, содержит ли строка определенную подстроку с помощью оператора "in" или метода "startswith()" и "endswith()"

4. Индексация строк начинается с нуля: у первого символа индекс 0 , следующего 1 и так далее. Индексы строк также могут быть указаны отрицательными числами. В этом случае индексирование начинается с конца строки: -1 относится к последнему символу, -2 к предпоследнему и так далее

5. Python также допускает возможность извлечения подстроки из строки, известную как ‘‘string slice’’. Если s это строка, выражение формы s[m:n] возвращает часть s , начинающуюся с позиции m , и до позиции n , но не включая позицию. Если опустить второй индекс s[n:] , срез длится от первого индекса до конца строки. Пропуск обоих индексов возвращает исходную строку. Это не копия, это ссылка на исходную строку

6. Строки в Python представляют собой последовательность символов, которая хранится в памяти как неизменяемый объект. Это делает их

более безопасными и предсказуемыми при работе с ними в многопоточных и распределенных приложениях

7. Проверить, что каждое слово в строке начинается с заглавной буквы: string.istitle()

8. Оператор in возвращает True , если подстрока входит в строку, и False , если нет

9. find(<подстрока>) возвращает первый индекс подстроки в строке

10. len() - возвращает длину строки

11. string.count() подсчитывает количество вхождений подстроки в строку

12. f-строки используются для форматирования. Напишите f или F перед кавычками строки. Это укажет python, что это f-строка вместо стандартной. Укажите любые переменные для воспроизведения в фигурных скобках ({})

13. Самый простой и удобный способ проверить, содержит ли строка определенную подстроку, — использовать оператор in

14. “{0} {1} {2}”.format(a, b, c)

15. Существует метод isnumeric() , который возвращает True в том случае, если все символы, входящие в строку, являются цифрами

16. Метод split() в Python разделяет строку на список подстрок по разделителю

17. Метод islower() возвращает True только в том случае, если строка составлена исключительно из строчных букв

18. Сделать это можно, вызвав вышеописанный метод islower() для первого символа строки

19. Python при попытке выполнения подобной операции будет выдана ошибка TypeError

20. Для того чтобы «перевернуть» строку, её можно разбить, представив в виде списка символов, «перевернуть» список, и, объединив его элементы, сформировать новую строку

21. Метод join() умеет объединять элементы списков в строки, разделяя отдельные строки с использованием заданного символа

22. Для решения этих задач можно воспользоваться методами upper() и lower(), которые, соответственно, приводят все символы строк к верхнему и нижнему регистрам

23. Методами upper() и lower(),обращаясь к символам строки по индексам

24. Имеется метод isupper(), который похож на уже рассмотренный islower(). Но isupper() возвращает True только в том случае, если вся строка состоит из прописных букв

25. Метод splitlines() разделяет строки по символам разрыва строки

26. Если обойтись без экспорта модуля, позволяющего работать с регулярными выражениями, то для решения этой задачи можно воспользоваться методом replace()

27. Методами startswith() и endswith()

28. Методом isspace(), который возвращает True только в том случае, если строка состоит исключительно из пробелов

29. Будет создана новая строка, представляющая собой исходную строку, повторенную три раза

30. Существует метод title(), приводящий к верхнему регистру первую букву каждого слова в строке

31. Метод partition() разбивает строку по заданной подстроке. После этого результат возвращается в виде кортежа. При этом подстрока, по которой осуществлялась разбивка, тоже входит в кортеж

32. Метод rfind() похож на метод find(), но он, в отличие от find(), просматривает строку не слева направо, а справа налево, возвращая индекс первого найденного вхождения искомой подстроки

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе со строками при написании программ с помощью языка программирования Python версии 3.x.