

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»

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических
наук, доцент, доцент кафедры
инфокоммуникаций

(подпись)

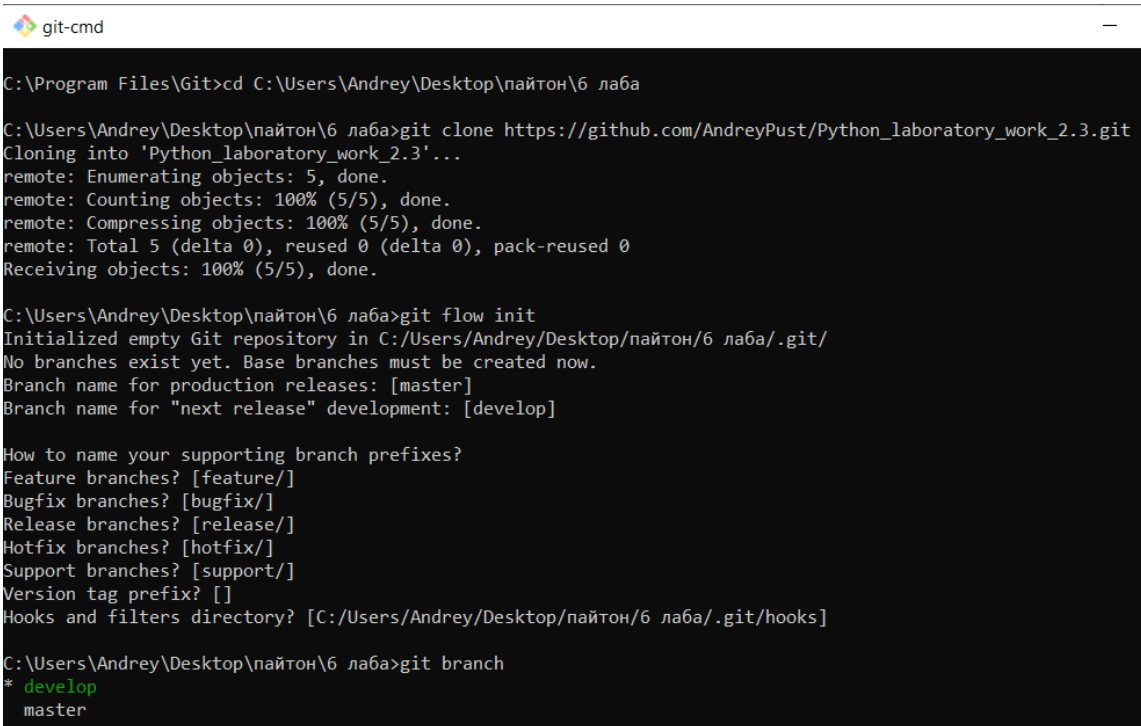
Ставрополь, 2023 г.

Тема: Работа со строками в языке Python.

Цель: приобрести навыки по работе со строками при написании программ с помощью языка программирования Python 3.x.

Ход работы:

Создание репозитория, клонирование репозитория и организация работы согласно ветвлению «git-flow» (рис. 1).



```
git-cmd
C:\Program Files\Git>cd C:\Users\Andrey\Desktop\пайтон\6 лаба
C:\Users\Andrey\Desktop\пайтон\6 лаба>git clone https://github.com/AndreyPust/Python_laboratory_work_2.3.git
Cloning into 'Python_laboratory_work_2.3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
C:\Users\Andrey\Desktop\пайтон\6 лаба>git flow init
Initialized empty Git repository in C:/Users/Andrey/Desktop/пайтон/6 лаба/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master]
Branch name for "next release" development: [develop]
How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/Andrey/Desktop/пайтон/6 лаба/.git/hooks]
C:\Users\Andrey\Desktop\пайтон\6 лаба>git branch
* develop
master
```

Рисунок 1 – Клонирование репозитория и модель ветвления «git-flow».

Проработка примеров лабораторной работы:

Пример 1.

Дано предложение, необходимо заменить все пробелы в таком предложении на символы «_».

Код программы примера №1 и результаты работы программы (рис. 2, 3).

```

DME.md example_1.py x
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Дано предложение, необходимо все пробелы в нем заменить символом «_».

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

```

Рисунок 2 – Код программы примера №1.

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\Desktop\пайтон
Введите ваше предложение: Война стара, как само человечество, но мир – современное изобретение.
Предложение после замены: Война_стара,_как_само_человечество,_но_мир_–_современное_изобретение.

Process finished with exit code 0

```

Рисунок 3 – Результат работы программы примера №1.

Пример 2.

Дано слово. Необходимо удалить среднюю букву в случае, если его длина нечетная, а если четная – то удалить две средние буквы.

Код программы примера №2 и результаты работы программы для слов разной четности (рис. 4, 5).

```

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

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

if __name__ == '__main__':
    word = input("Введите слово: ")
    idx = len(word) // 2
    if len(word) % 2 == 1:
        # Длина слова нечетная.
        r = word[:idx] + word[idx+1:]
    else:
        # Длина слова четная.
        r = word[:idx-1] + word[idx+1:]
    print(r)

```

Рисунок 4 – Код программы примера №2.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:
Введите слово: четное
чеое

Process finished with exit code 0
```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\p
Введите слово: нечетно
нечтно

Process finished with exit code 0
```

Рисунок 5 – Результаты работы программы примера №1 для слов разной четности.

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.

Код программы примера №3 и результаты работы программы для различных предложений и различных необходимых длин этих предложений не менее длины самого предложения (рис. 6, 7).

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

import sys

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

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

    # Разделить предложение на слова.
    words = s.split(' ')
```

```

# Проверить количество слов в предложении.
if len(words) < 2:
    print(
        "Предложение должно содержать несколько слов",
        file=sys.stderr
    )
    exit(1)

# Количество пробелов для добавления.
delta = n
for word in words:
    delta -= len(word)

# Количество пробелов на каждое слово.
w, r = delta // (len(words) - 1), delta % (len(words) - 1)

# Сформировать список для хранения слов и пробелов.
lst = []

# Пронумеровать все слова в списке и перебрать их.
for i, word in enumerate(words):
    lst.append(word)

    # Если слово не является последним, добавить пробелы.
    if i < len(words) - 1:
        # Определить количество пробелов.
        width = w
        if r > 0:
            width += 1
            r -= 1

        # Добавить заданное количество пробелов в список.
        if width > 0:
            lst.append(' ' * width)

# Вывести новое предложение, объединив все элементы списка lst.
print(''.join(lst))

```

Рисунок 6 – Код программы примера №3.

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\Desktop\пайтон\6 лаб
Введите предложение: Война стара, как само человечество, но мир – современное изобретение.
Введите длину: 100
Война    стара,    как    само    человечество,    но    мир    –    современное    изобретение.

Process finished with exit code 0

```

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe
Введите предложение: Целью войны является мир.
Введите длину: 50
Целью        войны        является        мир.

```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe
Введите предложение: Целью войны является мир.
Введите длину: 10
Заданная длина должна быть больше длины предложения
```

Рисунок 7 – Результаты работы программы примера №3.

Выполнение индивидуальных заданий:

Задание 1.

Дано предложение. Необходимо определить в нем количество гласных букв (Вариант 26). Так как в предложении могут быть как строчные, так и заглавные гласные, то в список гласных нужно включить и заглавные тоже.

Код программы индивидуального задания №1 и результаты работы программы с использованием различных предложений (рис. 8, 9).

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

# Дано предложение. Определить, сколько в нем гласных букв.

if __name__ == '__main__':
    s = input("Введите ваше предложение: ")
    # Создадим список, содержащий все гласные буквы
    list_gl = ['a', 'e', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я',
               'A', 'E', 'Ё', 'И', 'O', 'У', 'Э', 'Ю', 'Я']
    count_gl = 0 # счетчик гласных
    for i in s:
        if i in list_gl:
            count_gl += 1
    print("Количество гласных букв в предложении:", count_gl)
```

Рисунок 8 – Код программы индивидуального задания №1.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\Desktop\пайтон\6
Введите ваше предложение: Война стара, как само человечество, но мир – современное изобретение.
Количество гласных букв в предложении: 25
Process finished with exit code 0
```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\Desktop\пайтон\6 лаба\Python_
Введите ваше предложение: У Андрея мы сидим, на Андрея мы глядим, не сказать, тревожимся – но все время ежмся.
Количество гласных букв в предложении: 28
```

Рисунок 9 – Результаты работы программы индивидуального задания №1.

Задание 2.

Дано слово из 12 букв. Необходимо переставить в обратном порядке буквы, расположенные между второй и десятой буквами (с третьей по девятую букву) (Вариант 26).

Код программы индивидуального задания №2 и результаты работы программы с различными словами (рис. 10, 11).

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

# Дано слово из 12 букв. Переставить в обратном порядке буквы, расположенные между
# второй и десятой буквами (т. е. с третьей по девятую).

import sys

if __name__ == '__main__':
    s = input("Введите ваше слово из 12-ти букв: ")
    # Проверим, состоит ли слово из 12 букв
    if len(s) > 12:
        print("В слове более 12-ти букв!", file=sys.stderr)
        exit(1)

    revers_s = list(s[1:9])          # превратим строку в список
    revers_s = reversed(revers_s)    # перевернем элементы списка
    revers_s = ''.join(revers_s)     # превратим список в строку

    # Пересоберем итоговое слово
    s = s[0:2] + revers_s + s[9:12]
    print("Итоговое слово с переставленными буквами:", s)
```

Рисунок 10 – Код программы индивидуального задания №2.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\User
Введите ваше слово из 12-ти букв: 001234567000
Итоговое слово с переставленными буквами: 007654321000

Process finished with exit code 0
```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\pyt
Введите ваше слово из 12-ти букв: изобретатель
Итоговое слово с переставленными буквами: изтатербоель

Process finished with exit code 0
```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\pyth
Введите ваше слово из 12-ти букв: администрация
В слове более 12-ти букв!

Process finished with exit code 1
```

Рисунок 11 – Результаты работы индивидуального задания №2.

Задание 3.

Дана строка, состоящая только из букв. Необходимо заменить все буквы «а» на буквы «б» и наоборот, как заглавные, так и строчные. Например, при вводе строки «абвАБВ» должен получиться результат «бавБАВ».

Для того, чтобы после повторного использования метода `replace()` не заменить необходимые символы на другие, выполним метод `replace()` для замены необходимых символов на символы '1', '2', '3', '4' (так как в используемой строке содержатся только буквы по условию), а затем выполним замену этих цифр на требуемые буквы.

Код программы индивидуального задания №3 и результаты работы программы для разных строк (рис. 12, 13).


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

# Дана строка, состоящая только из букв. Необходимо заменить все
# буквы «а» на буквы «б» и наоборот, как заглавные, так и строчные.
# Например, при вводе строки «абвАБВ» должен получиться результат «бавБАВ».

if __name__ == '__main__':
    s = input("Введите вашу строку: ")

    # Замена символов на цифры
    s = s.replace(_old: 'a', _new: '1')
    s = s.replace(_old: 'A', _new: '2')
    s = s.replace(_old: 'б', _new: '3')
    s = s.replace(_old: 'Б', _new: '4')

    # Замена полученных цифр на требуемые буквы
    s = s.replace(_old: '1', _new: 'б')
    s = s.replace(_old: '2', _new: 'Б')
    s = s.replace(_old: '3', _new: 'а')
    s = s.replace(_old: '4', _new: 'А')
    print("Строка после замены букв:", s)
```

Рисунок 12 – Код индивидуального задания №3.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe
Введите вашу строку: абАБ
Строка после замены букв: баБА

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Us
Введите вашу строку: Видимость мира делает войну еще опаснее.
Строка после замены букв: Видимость мирб делбет войну еще олбснее.
```

Рисунок 13 – Результаты работы индивидуального задания №3.

Выполнение задания повышенной сложности:

Дано предложение. Напечатать все его слова, предварительно преобразовав каждое из них по следующему правилу:

- заменить первую встреченную букву а на о;

- удалить из слова все вхождения последней буквы (кроме нее самой);
- оставить в слове только первые вхождения каждой буквы;
- в самом длинном слове удалить среднюю (средние) букву(ы);
- принять, что такое слово – единственное.

Код программы задания повышенной сложности и результаты работы программы для различных предложений (рис. 14, 15).

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

# Дано предложение. Напечатать все его слова, предварительно
# преобразовав каждое из них по следующему правилу:
# • заменить первую встреченную букву а на о;
# • удалить из слова все вхождения последней буквы (кроме нее самой);
# • оставить в слове только первые вхождения каждой буквы;
# • в самом длинном слове удалить среднюю (средние) букву(ы);
# • принять, что такое слово – единственное.

if __name__ == '__main__':
    s = input("Введите ваше предложение: ")
    end_s = ' ' # конечное предложение

    # Найдем в предложении наибольшее слово
    s_words = s.split(' ') # разделим предложение на слова
    max_word = ''
    for i in s_words:
        if len(i) > len(max_word):
            max_word = i

    # Удалим из наибольшего слова средние буквы
    if len(max_word) % 2 == 1:
        # Индекс средней буквы для нечетного
        middle_index = len(max_word) // 2
        # Удаляем среднюю букву
        max_word_save = max_word # сохраняем слово для проверки
        max_word = max_word[:middle_index] + max_word[middle_index + 1:]
        # Собираем предложение обратно с учетом
        # отредактированного максимального слова
        s = ' '.join([word if word != max_word_save else max_word for word in
s_words])
    else:
        # Индексы средней буквы для четного
        middle_index = len(max_word) // 2
        # Удаляем среднюю букву
        max_word_save = max_word # сохраняем слово для проверки
        max_word = max_word[:middle_index-1] + max_word[middle_index + 1:]
        # Собираем предложение обратно с учетом
        # отредактированного максимального слова
        s = ' '.join([word if word != max_word_save else max_word for word in
s_words])

    # Оставим в слове только первые вхождения букв
    # и удалим все вхождения последней буквы
    for word in s_words:
        last_letter = word[-1] # последняя буква
        first_letters = [''] # создаем список нужных букв
```

```

# Дополним список первых вхождений
for letter in word:
    if letter not in first_letters:
        first_letters.append(letter)
# добавим последнее слово если такого нет в конце
if last_letter not in first_letters[-1]:
    first_letters.append(last_letter)
# Соберем слово обратно
result_word = ''.join(first_letters)
# И это слово отправим в предложение
end_s = end_s + result_word + ' '

# Заменяем первую встреченную букву а на о
end_s = end_s.replace("a", "o", 1)

print("Предложение после преобразований: ", end_s)

```

Рисунок 14 – Код программы задания повышенной сложности.

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe
Введите ваше предложение: Цеалью войны является мира
Предложение после преобразований: Цеолю войны явлется мира

Process finished with exit code 0

```

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\D
Введите ваше предложение: Лучше верный мир, чем расчет на победу Аристотель
Предложение после преобразований: Лучше верный мир, чем росчет на победу Аристоель

Process finished with exit code 0

```

Рисунок 15 – Результаты работы программы задания повышенной сложности.

Ответы на контрольные вопросы:

1. Строки – это упорядоченные последовательности символов, используемые для хранения и представления текстовой информации. С помощью строк можно работать со всем, что может быть представлено в текстовом виде.

2. Для задания строковых литералов, в Python можно использовать: кавычки “smth”, одинарные кавычки ‘smth’ и тройные кавычки (‘‘smth’’). Кроме того, есть возможность экранирования специальных символов – вставка специальных символов “\nsmth\n”. Если же нужно отключить экранирование, перед строкой необходимо добавить символ «r» - r“\nsmth\n”.

3. Строки можно складывать (Оператор «+») и умножать (Оператор «*»). Повторяет строку заданное количество («строка» * «число») раз). Также, можно проводить проверку на содержание в строке определённого фрагмента текста (Оператор принадлежности подстроки «in» – в случае, если в строке содержится заданная подстрока, возвращает True, иначе – False). Помимо операций, есть также функции: «chr()» – преобразует целое число в символ согласно таблице ASCII; «ord()» – обратная «chr» функция, преобразующая символ в число согласно таблице ASCII; «len()» – функция, возвращающая длину строки; «str()» – функция, преобразующая данные в строковый тип.

4. Доступ к отдельным символам в строке можно получить, указав имя строки и число в квадратных скобках [“номер”]. Индексация строк начинается с 0: у первого символа индекс 0, у следующего 1, затем 2 и т.д. 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 позволяет извлечь подстроку из строки, известную как «string slice». Выражение вида «строка»[m:n] возвращает часть строки, начинающуюся с индекса m, и до индекса n-1. Если пропустить первый индекс, то срез начинается с начала строки, а если пропустить второй индекс, то срез длится до конца строки. Пропуск обоих индексов возвращает ссылку на исходную строку. Если первый и второй индекс равны или первый больше второго, то возвращается пустая строка. Добавление дополнительного: и третьего индекса означает шаг, который указывает, сколько символов следует пропустить после извлечения каждого символа в срезе. Также можно указать отрицательное значение шага, в этом случае Python идет с конца строки. Начальный/первый индекс должен быть больше конечного/второго индекса.

6. Строки — один из типов данных, которые Python считает неизменяемыми, что означает невозможность их изменять. Однако, Python дает возможность изменять (заменять и перезаписывать) строки. Попытка заменить символы в строке по индексам приводят к ошибке TypeError, однако

нет особой необходимости изменять строки, ведь можно сгенерировать копию исходной строки с необходимыми изменениями (есть минимум 2 способа: использовать встроенный метод `string.replace(x, y)` или суммировать строку с необходимыми изменениями).

7. Функция `istitle()` позволяет проанализировать требуемую строку и узнать, с заглавной ли буквы начинается каждое слово. Если да – возвращает `True`, иначе – `False`.

8. Оператор «`in`» позволяет узнать, входит ли строка в состав другой строки. Если входит – возвращает `True`, иначе – `False`.

9. Функция «`find()`» позволяет найти подстроку внутри требуемой строки. Если подстрока найдена, то выводится индекс первой найденной подстроки, а если в строке вообще нет такой подстроки, то возвращается `-1`.

10. Функция «`len()`» возвращает число символов в требуемой строке.

11. Функция `count()` позволяет посчитать, сколько раз в требуемой строке встречается указанный символ.

12. f-строки – это способ форматирования строк в Python, который позволяет встраивать значения переменных и выражений непосредственно в строку с помощью префикса 'f'. Чтобы использовать f-строки, нужно просто создать строку с префиксом 'f' и вставить значения переменных или выражения в фигурные скобки внутри неё. Например: `extra = "wow!"`
`print(f'smth{extra}')` выведет «smth wow!».

13. Функция `find()` позволяет найти подстроку внутри требуемой строки. Если подстрока найдена, то выводится индекс первой найденной подстроки, а если в строке вообще нет такой подстроки, то возвращается `-1`.

14. Метод `format()` вставляет значения в фигурные скобки (`{...}`) внутри строки.

15. Функция `isdigit()` проверяет переданную ей строку. В случае, если в строке содержатся только цифры, функция вернёт `True`, иначе - `False`.

16. Функция `split()` разделяет строку на подстроки относительно требуемого символа-разделителя.

17. Функция `islower()` позволяет проверить, состоит ли строка только из строчных букв, возвращая `True` или `False`.

18. Если функция `islower()` анализирует полученную строку, если в ней все буквенные символы – строчные (маленькие), то возвращает `True`, иначе - `False`. Зная это, можно применить функцию `islower()` только к первому символу требуемой строки.

19. В Python возможно добавить число к строке, если при суммировании преобразовать число в строковый тип данных при помощи «`str()`».

20. Для переворачивания строки можно использовать срез. Требуется лишь указать отрицательный шаг = 1: `[::-1]`.

21. Функция `join()` объединяет элементы списков в строки, разделяя отдельные строки с использованием переданного ей символа. Так, передав ей требуемый список строк и указав разделительный символ «-», `join()` создаст новую требуемую строку с разделителем тире.

22. Функция `upper()` преобразует переданную ей строку, заменяя все символы в ней на заглавные. Функция `lower()` наоборот, делает все символы в строке строчными.

23. Функции `capitalize()` и `upper()` позволят перевести первый и последний символы строки в верхний регистр. Первая функция заменит первый символ на заглавный, в то время как передав функции `upper()` последний символ строки (При помощи строка[`len(строка)-1`] или строка[`-1`]), будет получен требуемый символ в верхнем регистре.

24. Функция «`islower()`» анализирует полученную строку, если в ней все буквенные символы – строчные (маленькие), то возвращает `True`, иначе - `False`.

25. Функция «`splitlines()`» делит строку на подстроки относительно присутствующих в строке разделителей (`\n`, `\r`, `\x1c`, прочее). Данный метод можно использовать при считывании данных из текстовых документов, так как убирает множество ненужных символов самостоятельно.

26. Функция «`replace()`» позволяет заменять в требуемой строке подстроку на переданное ей значение.

27. Чтобы проверить, начинается ли строка с требуемых символов, можно использовать функцию «startswith(“строка”)». Если же нужно проверить с конца строки, то можно использовать функцию «endswith(“строка”)».

28. Для проверки строки на содержание чего-либо кроме пробелов, можно использовать функцию «isspace()», возвращающую False, если есть символы, не являющиеся пробелами.

29. При умножении строки, создаётся новая строка, состоящая из повторённых 3 раза изначальных строк друг за другом.

30. Для замены всех первых символов слов в строке на заглавные, есть функция title.

31. Partition(“разделитель”) делит строку на основе полученного разделителя. Находя первое совпадение с разделителем, «partition» возвращает три строки – до разделителя, сам разделитель, строку после разделителя.

32. «rfind», в отличие от «find», ищет подстроку в строке начиная с конца, а не с начала.

Вывод: в ходе выполнения лабораторной работы была подробно изучена работа со строками в языке программирования Python. Были изучены способы создания строки, способы экранирования символов, разделение строки по словам и обратно, работа с элементами строки по индексам, индексация элементов строки, основные строковые операторы, операторы принадлежности, способы поиска элементов строке или подстроки в строке, основные встроенные функции в Python для работы со строками, способы срезы строк для редактирования существующей строки, методы форматирования строки, способы классификации строк с использование соответствующих методов, работа с регистром символов в строке при помощи соответствующих методов, а также преобразование строки в список и обратно.