

## Лабораторная работа 2.3

Тема: Работа со строками в языке Python.

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

Порядок выполнения работы


1. Создаём аккаунт в GitHub. Затем создаём новый общедоступный репозиторий, в котором будет использована лицензия MIT и язык программирования Python.

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*

 RomanGorchakov ▾

Repository name \*

/ Test

✓ Test is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-chainsaw](#) ?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

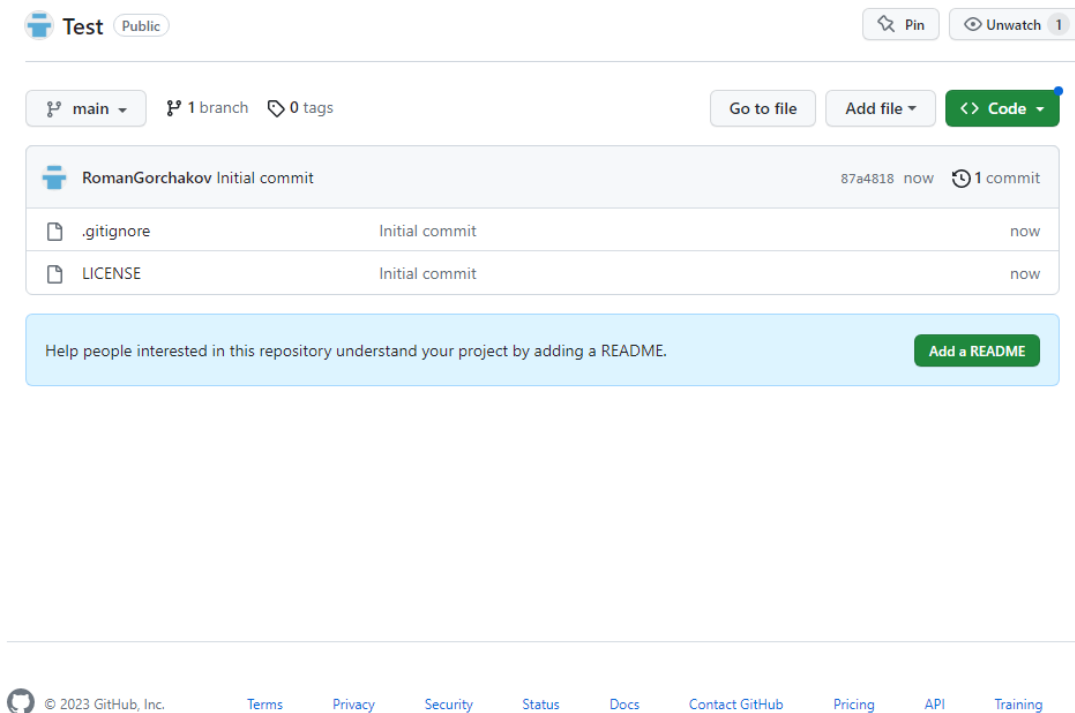
Choose a license

License: MIT License ▾


















A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository



2. Теперь необходимо дополнить файл `.gitignore` с необходимыми правилами для языка программирования Python. Для этого переходим по ссылке «<https://github.com/github/gitignore>» и скачиваем оттуда файл «Python.gitignore».

	Phalcon.gitignore	Remove trailing asterisks in Phalcon rules	10 years ago
	PlayFramework.gitignore	Added /project/project to PlayFramework.gitignore	7 years ago
	Plone.gitignore	Covered by global vim template	10 years ago
	Prestashop.gitignore	Update for Prestashop 1.7 (#3261)	3 years ago
	Processing.gitignore	Ignore transpiled .java and .class files (#3016)	4 years ago
	PureScript.gitignore	Update PureScript adding .spago (#3278)	3 years ago
	<a href="#">Python.gitignore</a>	Update Python.gitignore	last year
	Qooxdoo.gitignore	Add gitignore for qooxdoo apps	13 years ago
	Qt.gitignore	Remove trailing whitespace	2 years ago
	R.gitignore	Merge pull request #3792 from jl5000/patch-1	2 years ago
	README.md	Merge pull request #3854 from AnilSeervi/patch-1	2 years ago
	ROS.gitignore	Added ignore for files created by <code>catkin_make_isolated</code>	6 years ago
	Racket.gitignore	Update Racket.gitignore	2 years ago
	Rails.gitignore	Ignore Rails .env according recommendations	2 years ago
	Raku.gitignore	Changes the name of Perl 6 to Raku (#3312)	3 years ago
	RhodesRhomobile.gitignore	Add Rhodes mobile application framework gitignore	13 years ago
	Ruby.gitignore	Ruby: ignore RuboCop remote inherited config files (#3197)	4 years ago

```
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

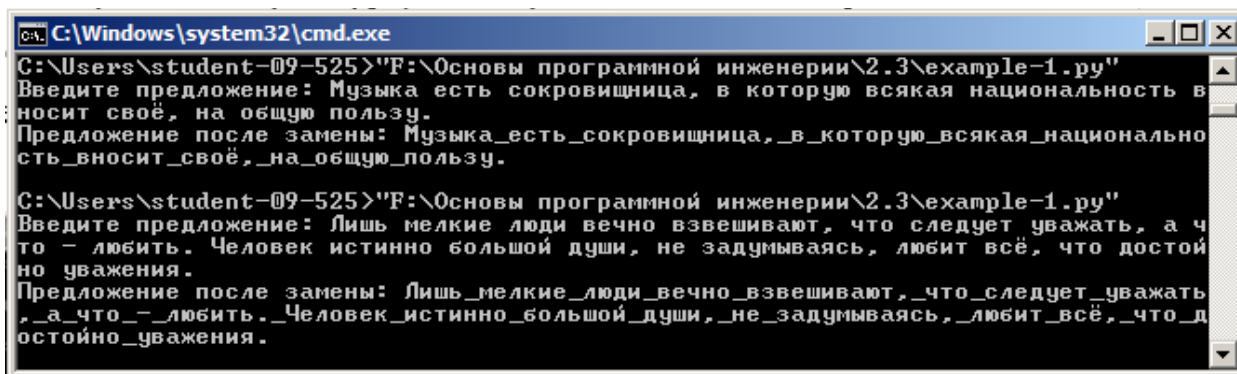
3. Теперь создаём файл «README.md», где вносим информацию о своей группе и ФИО. Сохраняем набранный текст через кнопку «Commit changes».



4. После этого нужно организовать репозиторий в соответствии с моделью ветвления Git-flow. Для этого В окне «Codespace» выбираем опцию «Create codespace on main», где введём команды: «git branch develop» и «git push -u origin develop» для создания ветки разработки; «git branch feature\_branch» для создания ветки функций; «git branch release/1.0.0» для создания ветки релиза; «git checkout main» и «git branch hotfix» для создания веток hotfix.

```
@RomanGorchakov →/workspaces/Py3 (main) $ git checkout -b develop
Switched to a new branch 'develop'
● @RomanGorchakov →/workspaces/Py3 (develop) $ git checkout -b feature_branch
Switched to a new branch 'feature_branch'
● @RomanGorchakov →/workspaces/Py3 (feature_branch) $ git checkout develop
Switched to branch 'develop'
● @RomanGorchakov →/workspaces/Py3 (develop) $ git checkout -b release/1.0.0
Switched to a new branch 'release/1.0.0'
● @RomanGorchakov →/workspaces/Py3 (release/1.0.0) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py3 (main) $ git checkout -b hotfix
Switched to a new branch 'hotfix'
● @RomanGorchakov →/workspaces/Py3 (hotfix) $ git checkout develop
Switched to branch 'develop'
● @RomanGorchakov →/workspaces/Py3 (develop) $ git branch
* develop
  feature_branch
  hotfix
  main
  release/1.0.0
○ @RomanGorchakov →/workspaces/Py3 (develop) $
```

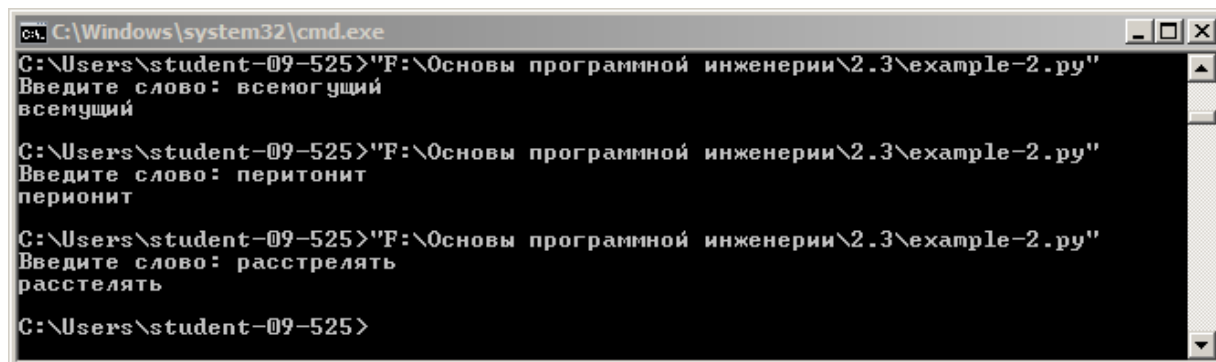
5. Создаём файл «example-1.py», в котором пользователю нужно ввести предложение, и программа должна заменить все пробелы в этом предложении на «\_».



```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-1.py"
Введите предложение: Музыка есть сокровищница, в которую всякая национальность вносит своё, на общую пользу.
Предложение после замены: Музыка_есть_сокровищница,_в_которую_всякая_национальность_вносит_своё,_на_общую_пользу.

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-1.py"
Введите предложение: Лишь мелкие люди вечно взвешивают, что следует уважать, а что - любить. Человек истинно большой души, не задумываясь, любит всё, что достойно уважения.
Предложение после замены: Лишь_мелкие_люди_вечно_взвешивают,_что_следует_уважать,_а_что_-_любить._Человек_истинно_большой_души,_не_задумываясь,_любит_всё,_что_достойно_уважения.
```

6. Создаём файл «example-2.py», в котором пользователю нужно ввести слово, а программа удаляет в этом слове одну среднюю букву, если длина слова нечётная, и две средние буквы, если длина слова чётная.



```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-2.py"
Введите слово: всемогущий
всенущий

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-2.py"
Введите слово: перитонит
перинит

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-2.py"
Введите слово: расстрелять
расстелать

C:\Users\student-09-525>
```

7. Создаём файл «example-3.py», в котором пользователю нужно ввести предложение и натуральное число, а программа сравнивает длину предложения и введённое число, и если оно больше длины предложения, то она добавит в предложение недостающее количество пробелов.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-3.py"
Введите предложение: Музыка есть сокровищница, в которую всякая национальность в
носит своё, на общую пользу.
Введите длину: 85
Заданная длина должна быть больше длины предложения

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\example-3.py"
Введите предложение: Лишь мелкие люди вечно взвешивают, что следует уважать, а ч
то - любить. Человек истинно большой души, не задумываясь, любит всё, что достой
но уважения.
Введите длину: 200
Лишь мелкие люди вечно взвешивают, что следует уважать, а
что - любить. Человек истинно большой души, не задумываясь, л
юбит всё, что достойно уважения.
```

8. Создаём файл «individual-1.py», в котором пользователю нужно ввести слово, а программа добавит к нему четыре знака «плюс» и пять знаков «минус».

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-1.py"
Введите слово: всемогущий
++++всемогущий-----

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-1.py"
Введите слово: олива
++++олива-----

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-1.py"
Введите слово: перитонит
++++перитонит-----

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-1.py"
Введите слово: расстрелять
++++расстрелять-----
```

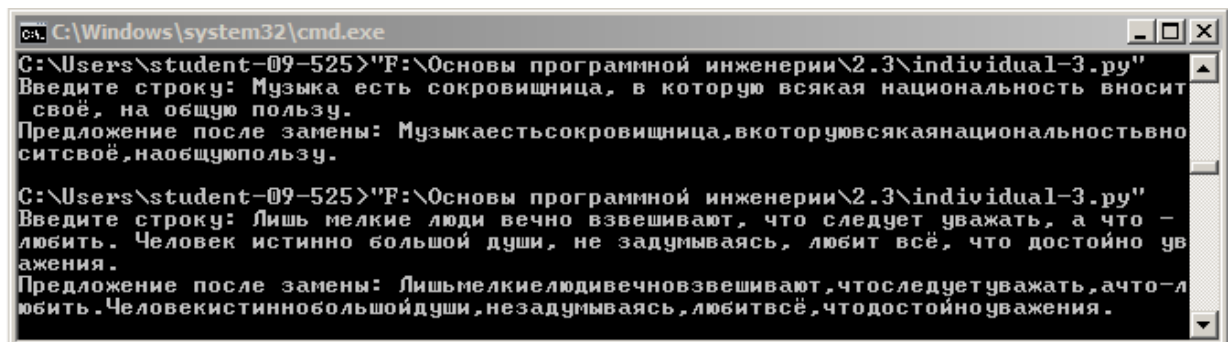
9. Создаём файл «individual-2.py», в котором пользователю нужно ввести последовательность символов, в начале которой имеется некоторое количество одинаковых символов, а программа определяет их количество.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-2.py"
Введите последовательность символов: ьь65фкои
Не все символы последовательности одинаковые.
2

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-2.py"
Введите последовательность символов: яё3к17жигц
Не все символы последовательности одинаковые.
1

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-2.py"
Введите последовательность символов: гgggg
Все символы последовательности одинаковые.
5
```

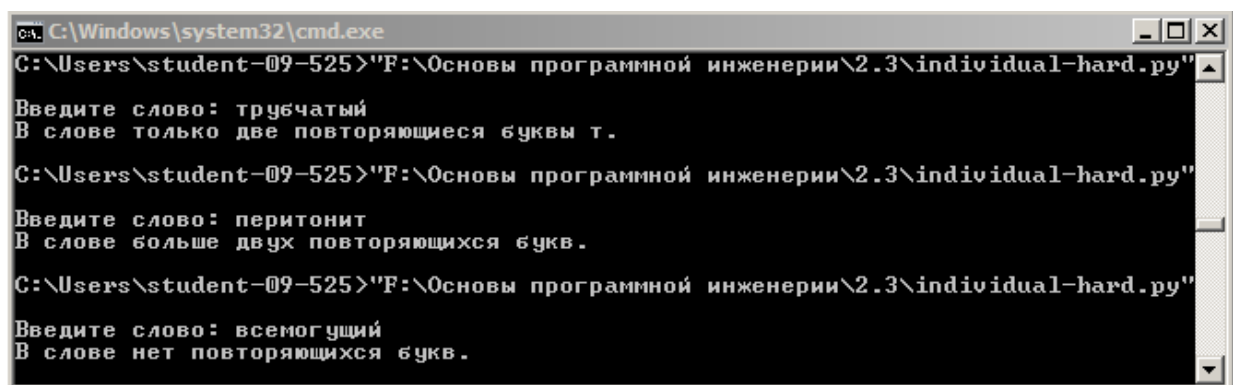
10. Создаём файл «individual-3.py», в котором пользователю нужно ввести строку, а программа удалит из неё все пробелы.



```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-3.py"
Введите строку: Музыка есть сокровищница, в которую всякая национальность вносит
своё, на общую пользу.
Предложение после замены: Музыкаестьсокровищница,вкоторуювсякаянациональноствно
ситсвоё,наобщуюпользу.

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-3.py"
Введите строку: Лишь мелкие люди вечно взвешивают, что следует уважать, а что –
любить. Человек истинно большой души, не задумываясь, любит всё, что достойно ув
ажения.
Предложение после замены: Лишьмелкиелюдивечновзвешивают,чтоследуетуважать,ачто–л
юбить.Человекистиннобольшойдуши,незадумываясь,любитвсё,чтодостойноуважения.
```

11. Создаём файл «individual-hard.py», в котором пользователю нужно ввести слово, а программа находит в нём две повторяющиеся буквы.



```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-hard.py"
Введите слово: трубчатый
В слове только две повторяющиеся буквы т.

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-hard.py"
Введите слово: перитонит
В слове больше двух повторяющихся букв.

C:\Users\student-09-525>"F:\Основы программной инженерии\2.3\individual-hard.py"
Введите слово: всемогущий
В слове нет повторяющихся букв.
```

12. Сохраняем лабораторную работу в качестве PDF-файла и выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    ПОРТЫ    КОММЕНТАРИИ

```
@RomanGorchakov →/workspaces/Py3 (develop) $ git add .
@RomanGorchakov →/workspaces/Py3 (develop) $ git commit "Python files"
error: pathspec 'Python files' did not match any file(s) known to git
@RomanGorchakov →/workspaces/Py3 (develop) $ git commit -m "Python files"
[develop f89d5e0] Python files
8 files changed, 113 insertions(+)
create mode 100644 example-1.py
create mode 100644 example-2.py
create mode 100644 example-3.py
create mode 100644 individual-1.py
create mode 100644 individual-2.py
create mode 100644 individual-3.py
create mode 100644 individual-hard.py
create mode 100644 "\320\233\320\2402.3_\320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf"
@RomanGorchakov →/workspaces/Py3 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
@RomanGorchakov →/workspaces/Py3 (main) $ git merge develop
Updating ca050d6..f89d5e0
Fast-forward
 example-1.py | 7 ++++++
 example-2.py | 13 ++++++++
 example-3.py | 40 ++++++++++++++++++++++++++++++++++++++
 individual-1.py | 7 ++++++
 individual-2.py | 18 ++++++++
 individual-3.py | 7 ++++++
 individual-hard.py | 21 ++++++++
 "\320\233\320\2402.3_\320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf" | Bin 0 -> 555362 bytes
8 files changed, 113 insertions(+)
create mode 100644 example-1.py
create mode 100644 example-2.py
create mode 100644 example-3.py
create mode 100644 individual-1.py
create mode 100644 individual-2.py
create mode 100644 individual-3.py
create mode 100644 individual-hard.py
create mode 100644 "\320\233\320\2402.3_\320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf"
@RomanGorchakov →/workspaces/Py3 (main) $ git checkout -b develop
fatal: a branch named 'develop' already exists
@RomanGorchakov →/workspaces/Py3 (main) $ git push -u
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 2 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 485.00 KiB | 15.16 MiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/RomanGorchakov/Py3
 ca050d6..f89d5e0 main -> main
branch 'main' set up to track 'origin/main'.
@RomanGorchakov →/workspaces/Py3 (main) $
```

main

1 Branch

0 Tags

Q Go to file

Add file

<> Code

RomanGorchakov

Python files

f89d5e0 · 5 minutes ago

4 Commits

.gitignore	Create .gitignore	last week
LICENSE	Create LICENSE	last week
README.md	Create README.md	last week
example-1.py	Python files	5 minutes ago
example-2.py	Python files	5 minutes ago
example-3.py	Python files	5 minutes ago
individual-1.py	Python files	5 minutes ago
individual-2.py	Python files	5 minutes ago
individual-3.py	Python files	5 minutes ago
individual-hard.py	Python files	5 minutes ago
ЛР2.3_ГорчаковРВ.pdf	Python files	5 minutes ago

README

MIT license

# Информация обо мне

Имя: Роман.

Фамилия: Горчаков.

Отчество: Владимирович.

Название группы: ПИЖ-6-о-22-1.



## Контрольные вопросы

### 1. Что такое строки в языке Python?

Строки в Python – это 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. Какие существуют способы задания строковых литералов в языке Python?

Существует несколько литералов строк. Строки в апострофах и в кавычках позволяют вставлять в литералы строк символы кавычек или апострофов, не используя экранирование. Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры. "Сырые" строки подавляют экранирование. Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

### 3. Какие операции и функции существуют для строк?

Операции: сложение строк, умножение строки на число, принадлежность подстроки строке.

Функции: `chr` (преобразует целое число в символ), `ord` (преобразует символ в целое число), `len` (возвращает длину строки), `str` (изменяет тип объекта на `string`).

### 4. Как осуществляется индексирование строк?

Индексация строк начинается с нуля: у первого символа индекс 0, следующего 1 и так далее. Индекс последнего символа в python – «длина строки минус один».

### 5. Как осуществляется работа со срезами для строк?

Если `s` это строка, выражение формы `s[m:n]` возвращает часть `s`, начинающуюся с позиции `m`, и до позиции `n`, но не включая позицию. Второй индекс указывает символ, который не включен в результат.

### 6. Почему строки Python относятся к неизменяемому типу данных?

Потому что в Python нельзя поменять какой-нибудь символ в строке.

7. Как проверить то, что каждое слово в строке начинается с заглавной буквы?

С помощью метода `str.istitle()`.

8. Как проверить строку на вхождение в неё другой строки?

С помощью оператора `in`.

9. Как найти индекс первого вхождения подстроки в строку?

С помощью метода `str.find()`.

10. Как подсчитать количество символов в строке?

С помощью функции `len()`.

11. Как подсчитать то, сколько раз определённый символ встречается в строке?

С помощью метода `str.count()`.

12. Что такое f-строки и как ими пользоваться?

В Python версии 3.6 был представлен новый способ форматирования строк. Эта функция официально названа литералом отформатированной строки, но обычно упоминается как f-строки (f-string).

Одной простой особенностью f-строк, которые вы можете начать использовать сразу, является интерполяция переменной. Вы можете указать имя переменной непосредственно в f-строковом литерале (f'string'), и python заменит имя соответствующим значением.

13. Как найти подстроку в заданной части строки?

С помощью метода `str.index()`.

14. Как вставить содержимое переменной в строку, воспользовавшись методом `format()`?

Если для подстановки требуется только один аргумент, то значение метода — сам аргумент, а если несколько, то значениями будут являться все аргументы со строками подстановки (обычных или именованных).

15. Как узнать о том, что в строке содержатся только цифры?

С помощью метода `str.isnumeric()`.

16. Как разделить строку по заданному символу?

С помощью метода `str.split()`.

17. Как проверить строку на то, что она составлена только из строчных букв?

С помощью метода `str.islower()`.

18. Как проверить то, что строка начинается со строчной буквы?

С помощью метода `str[0].islower()`.

19. Можно ли в Python прибавить целое число к строке?

Нет, нельзя.

20. Как «перевернуть» строку?

С помощью метода `«».join(reversed())`.

21. Как объединить список строк в одну строку, элементы которой разделены дефисами?

С помощью синтаксической конструкции `«-».join([«первый элемент», «второй элемент», «третий элемент»])`.

22. Как привести всю строку к верхнему или нижнему регистру?

С помощью методов `str.upper()` и `str.lower()` соответственно.

23. Как преобразовать первый и последний символы строки к верхнему регистру?

С помощью метода `str[0].upper() + str[1:-1] + str[-1].upper()`.

24. Как проверить строку на то, что она составлена только из прописных букв?

С помощью метода `str.isupper()`.

25. В какой ситуации вы воспользовались бы методом `splitlines()`?

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

26. Как в заданной строке заменить на что-либо все вхождения некоей подстроки?

С помощью метода `str.replace()`.

27. Как проверить то, что строка начинается с заданной последовательности символов, или заканчивается заданной последовательностью символов?

С помощью методов `str.startswith()` и `str.endswith()`.

28. Как узнать о том, что строка включает в себя только пробелы?

С помощью метода `str.isspace()`.

29. Что случится, если умножить некую строку на 3?

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

30. Как привести к верхнему регистру первый символ каждого слова в строке?

С помощью метода `str.title()`.

31. Как пользоваться методом `partition()`?

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. В каких ситуациях пользуются методом `rfind()`?

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.