

# PYTHON | БЕЗ ВОДЫ

Практический курс по Python

# #Блок 6 | Строки

Для быстрого старта

# О курсе

- ▶ Это **ПРАКТИЧЕСКИЙ** КУРС по программированию
- максимальная польза от курса в самостоятельном решении наибольшего числа задач
- вопросы на занятиях – самое ценное для учеников
- разбор задач учеников **НА ЗАНЯТИЯХ**
- разбор вопросов по задачам **НА ЗАНЯТИЯХ**

# Регламент

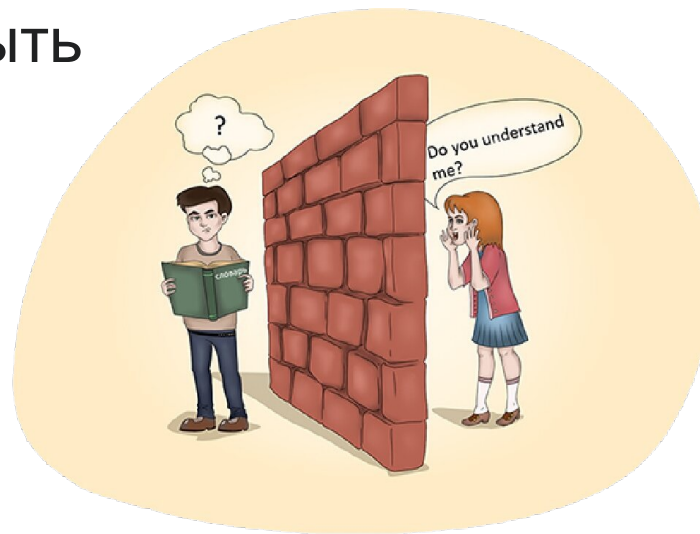
- ▶ Длительность одного занятия 90 минут (может +)
- ▶ 1 занятие - теория | 3 занятия - практика
- ▶ Регулярность занятий 1 раз в неделю
- ▶ Проходим блоками
- ▶ Занятия по скайпу
- ▶ Запись теоретического занятия
- ▶ Видео доступно на youtube в закрытом плейлисте

# **#1 | Ещё раз про объекты**

# Ещё раз про объекты

## ► Объект в программировании:

- сущность в цифровом пространстве, обладающая состоянием и поведением, имеющая поля и методы
- с каждым объектом можно взаимодействовать **строго определенным образом** через методы объектов
- **для одних объектов методы одни – для других другие**
- для разных объектов могут быть одинаковые методы или действия, но срабатывать будут по-разному



Помните пакетик с яйцами?



# Объекты одного типа могут быть похожи на другие

- ▶ Состоять из элементов на пронумерованных местах
- ▶ К элементам можно обращаться по индексу
- ▶ Можно делать срезы
- ▶ Можно перебирать с помощью циклов
- ▶ Можно сцеплять
- ▶ Можно изменять с помощью методов



# **#2 | Строки**

# Что такое строка

▶ Строка — тип данных, хранящий в себе набор символов произвольной длины

- класс *str*
- при создании заключаются в кавычки ‘ ’ или “ ”

```
s1 = 'Python без воды'  
s2 = "Python без воды"
```

- получается при присвоении значения функции *input()*, при считывании файлов, при введении вручную или генерации выражения (и т.д.)

# Пустая строка

- ▶ Пустая строка имеет вид "" – bool тип **False**
- ▶ Строка из одного пробела " " – bool тип **True**
- ▶ Функция *input()* передает в переменную пустую строку при нажатии ENTER
- ▶ Bool значение пустой строки можно использовать при составлении условных конструкций и циклов

```
s = 'Python без воды'
while s:
    print(s[-1],end="-")
    s = s[:-1]
```

block\_6 x

/Library/Frameworks/Python.framework

ы-д-о-в- -з-е-б- -п-о-х-т-у-Р-

# **#3 | Операторы**

# Конкатенация строк

► С помощью оператора сложения (+) можно сцепить строки

- сцепление строк создает новую строку, в которой строки, которые сцепляли идут одна за другой подряд без пробелов

```
a = 'Python'
b = 'Обучение'
print(a + b)
```

block\_6 ×

/Library/Frameworks/  
PythonОбучение

- аналогичного результата можно добиться, если использовать функцию print с параметрами *sep = ""* и *end = ""*

```
a = 'Python'
b = 'Обучение'
print(a, b, sep="")
```

block\_6 ×

/Library/Frameworks/Python  
PythonОбучение

# Конкатенация строк

- ▶ С помощью оператора умножения (\*) можно повторить строку несколько раз

```
a = 'Python курс '  
print(a*3)
```

block\_6 ×

```
/Library/Frameworks/Python.framework  
Python курс Python курс Python курс
```

# Многострочный текст

- ▶ Заключив текст в тройные кавычки, можно выводить его на экран без специальных операторов

```
a = '''--- Тройные кавычки ---  
Можно как одинарные тройные кавычки  
    Так и двойные тройные кавычки  
        И отступы получаются неплохо  
    И не нужно использовать специальный символ  
для перехода на новую строку  
'''  
print(a)
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10  
--- Тройные кавычки ---  
Можно как одинарные тройные кавычки  
    Так и двойные тройные кавычки  
        И отступы получаются неплохо  
    И не нужно использовать специальный символ  
для перехода на новую строку
```

# Экранирование

► Чтобы вывести кавычки в тексте, заключенном в одинарных (двойных) кавычках, можно использовать знак экранирования \

```
a = 'Курс \'Python без воды\' набирает обороты'  
print(a)
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.  
/Users/raybin/PycharmProjects/course/block_6.py  
Курс 'Python без воды' набирает обороты
```

# Проверка вхождения подстроки в строку

- ▶ Чтобы проверить, что подстрока входит в строку, используют оператор *in*
- Проверяется, что либо один символ является частью строки, либо слово является частью строки

```
a = 'Курс "Python без воды" набирает обороты'  
s = 'Много интересных курсов без лишнего кода по Python есть в интернете'.split()  
for x in s:  
    if x in a:  
        print(x)
```

Run

block\_6 ×

/Library/Frameworks/Python.framework/Versions/3.10/bin/python3

/Users/raybin/PycharmProjects/course/block\_6.py

без

Python

в

# Проверка вхождения подстроки в строку

- ▶ Порой надо проверить обратное утверждение, для чего используют конструкцию ***not in***

```
a = 'Курс "Python без воды" набирает обороты'  
s = 'Много интересных курсов без лишнего кода по Python есть в интернете'.split()  
for x in s:  
    if x not in a:  
        print(x)
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/  
Много  
интересных  
курсов  
лишнего  
кода  
по  
есть  
интернете
```

# **#4 | Индексация строк**

# Индексация строк

- ▶ У каждого символа в строке есть порядковый номер (индекс). Индексы начинаются с 0 (как в списках)

```
s = 'Как много нам открытий чудных готовит просвещения дух'  
print(s[0])  
print(s[2])  
print(s[-1])  
print(s[-3])
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3
```

```
К
```

```
к
```

```
х
```

```
д
```

# Индексация строк

- ▶ При обращении к элементу, индекса которого нет в строке, будет ошибка

```
s = 'Как много нам открытий чудных готовит просвещенья дух'  
print(s[100])
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/Py  
Traceback (most recent call last):  
  File "/Users/raybin/PycharmProjects/course/block_6.py", line 2, in <module>  
    print(s[100])  
IndexError: string index out of range
```

# Обход символов строк

- ▶ Чтобы пройти по символам строк, используют циклы. Наиболее удобен для этого цикл *for*

```
s = 'Как много нам открытий чудных готовит просвещенья дух'  
for i in range(len(s)):  
    print(s[i], end="-")
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/course/block_6.py  
К-а-к- -м-н-о-г-о- -н-а-м- -о-т-к-р-ы-т-и-й- -ч-у-д-н-ы-х- -г-о-т-о-в-и-т- -п-р-о-с-в-е-щ-е-н-ь-я- -д-у-х-
```

```
s = 'Как много нам открытий чудных готовит просвещенья дух'  
for x in s:  
    print(x, end="-")
```

Run

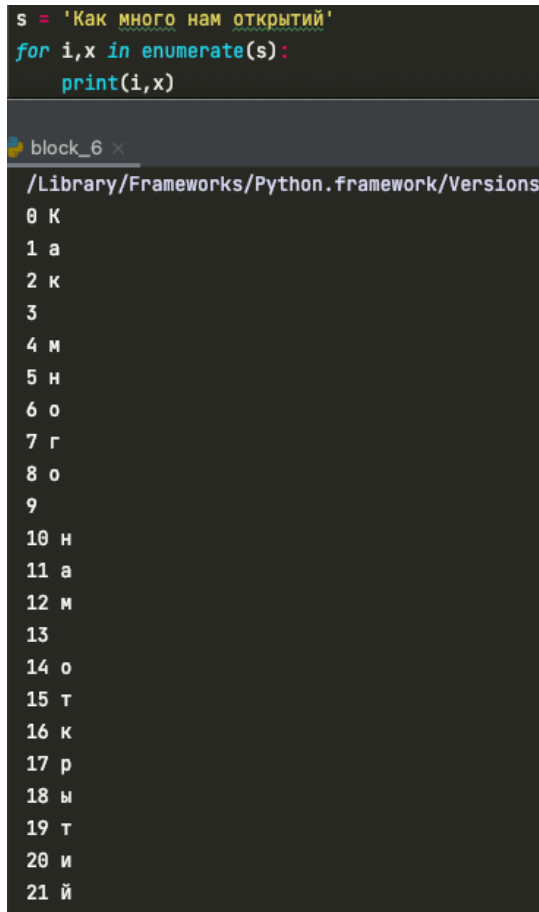
block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/course/block_6  
К-а-к- -м-н-о-г-о- -н-а-м- -о-т-к-р-ы-т-и-й- -ч-у-д-н-ы-х- -г-о-т-о-в-и-т- -п-р-о-с-в-е-щ-е-н-ь-я- -д-у-х-
```

# Обход символов строк

► Чтобы пройтись по символам строк, используют циклы. Наиболее удобен для этого цикл *for*

```
s = 'Как много нам открытий'
for i,x in enumerate(s):
    print(i,x)
```



```
block_6 x
/Library/Frameworks/Python.framework/Versions
0 К
1 а
2 к
3
4 м
5 н
6 о
7 г
8 о
9
10 н
11 а
12 м
13
14 о
15 т
16 к
17 р
18 ы
19 т
20 и
21 й
```



# Срезы строк

- ▶ Срез позволяет получить часть строки **между указанным индексами с заданным шагом**

```
s = '123456789'
print(s[0:len(s):1])
print(s[:len(s):1])
print(s[:len(s):])
print(s[::])
print(s[:])
```

block\_6 ×

/Library/Frameworks/Python.framework/Versions/

123456789

123456789

123456789

123456789

123456789

```
s = '123456789'
print(s[1:len(s)-1:])
print(s[1:len(s)-1])
print(s[1:-1:])
print(s[1:-1])
```

block\_6 ×

/Library/Frameworks/Python.framework/

2345678

2345678

2345678

2345678

# Срезы строк

- ▶ Получение части строки **между указанным индексами с шагом 2**

```
s = '123456789'
print(s[0:-1:2])
print(s[:-1:2])
```

block\_6 ×

/Library/Frameworks/Python

1357  
1357

```
s = '123456789'
print(s[::2])
print(s[0::2])
```

block\_6 ×

/Library/Frameworks/Python.

13579  
13579

# Срезы строк

- ▶ Получение части строки **между указанным индексами с шагом -1 (переписать наоборот)**

```
s = '123456789'  
print(s[::-1])  
print(s[-1::-1])
```

block\_6 ×

/Library/Frameworks/Python

987654321

987654321

```
s = '123456789'  
print(s[::-2])  
print(s[-1::-2])
```

block\_6 ×

/Library/Frameworks/Python

97531

97531

# Срезы строк

## ▶ Срезы срезов

```
s = '123456789'
print(s[1:5][::-1])
```

block\_6 ×

/Library/Frameworks/  
5432

# Удаление символа по индексу

- ▶ Строка перезаписывает себя на новую строку из двух срезов без заменяемого символа

```
s = '123456789'  
s = s[:3] + s[4:]  
print(s)
```

block\_6 ×

/Library/Frameworks  
12356789

# Изменение символа по индексу

- ▶ Строка перезаписывает себя на новую строку из двух срезов без заменяемого символа, вместо которого добавляют новый. **Перезаписать символ как в списке нельзя!!!**

```
s = '123456789'
s = s[:3] + '*' + s[4:]
print(s)
```

block\_6 ×

/Library/Frameworks/Python

123\*56789

# **#5** | **Методы и функции по работе со строками**

# Длина строки

- ▶ Функция *len()* возвращает длину в виде целого числа

```
s = 'Python без воды'
print(len(s))
```

block\_6 ×  
/Library/Frameworks/Python.framework  
15

```
s = 'Python без воды'
for i in range(len(s)):
    print(s[i], end="->")
```

block\_6 ×  
/Library/Frameworks/Python.framework/Versions/  
P->y->t->h->o->n-> ->б->е->з-> ->в->о->д->ы->

- при подсчета количества символов пробелы тоже считаются

# Приведение к типу строка

- ▶ Функция ***str()*** переводит числа (int, float) в строки
  - объекты других типов не перевести в строковый тип используя конструкцию ***str(object)***
  - чтобы перевести итерируемый объект в строковый тип, нужно использовать функцию ***map()*** или генератор, а также метод ***.join()***

```
a = [1,6,7,False,879]
s = ''.join(map(str,a))
print(s)
```

block\_6 ×

/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6

167False879

```
a = [1,6,7,False,879]
s = ''.join(str(x) for x in a)
print(s)
```

block\_6 ×

/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6

167False879

```
a = [1,6,7,False,879]
print(*a, sep="")
```

block\_6 ×

/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6

167False879

# Изменение регистра первого символа

► Метод `.capitalize()` возвращает *копию строки*, в которой первый символ имеет верхний регистр, а все остальные символы имеют нижний регистр

```
s = 'кодим с интересом'  
print(s.capitalize())  
print(s)
```

block\_6 ×

```
/Library/Frameworks/Python  
Кодим с интересом  
кодим с интересом
```

# Замена регистра всех символов на противоположные

► Метод `.swapcase()` возвращает *копию строки*, в которой все символы, имеющие верхний регистр, преобразуются в символы нижнего регистра и наоборот.

```
s = 'Кодим С Интересом'
print(s.swapcase())
print(s)
```

block\_6 ×

```
/Library/Frameworks/Python
КОДИМ с ИНТЕРЕСОМ
Кодим С Интересом
```

# Перевод всех первых букв слов в верхний регистр

► Метод `.title()` возвращает *копию строки*, в которой первый символ каждого слова переводится в верхний регистр, а другие символы в нижний

```
s = 'кодим с интересом'  
print(s.title())  
print(s)
```

block\_6 ×

/Library/Frameworks/Python

Кодим С Интересом

кодим с ИНТЕРЕСОМ

# Перевод всех букв слов в нижний регистр

► Метод `.lower()` возвращает *копию строки*, в которой все символы имеют нижний регистр

```
s = 'кодим с ИНТЕРЕСОМ'  
print(s.lower())  
print(s)
```

block\_6 ×

/Library/Frameworks/Python

кодим с интересом

кодим с ИНТЕРЕСОМ

# Перевод всех букв слов в верхний регистр

► Метод `.upper()` возвращает *копию строки*, в которой все символы имеют верхний регистр

```
s = 'кодим с ИНТЕРЕСОМ'  
print(s.upper())  
print(s)
```

block\_6 ×

/Library/Frameworks/Python

КОДИМ С ИНТЕРЕСОМ

кодим с ИНТЕРЕСОМ

# Поиск количества вхождений подстроки в строку

► Метод `.count(sub, start, stop)` считает количество непересекающихся вхождений подстроки `sub` в исходную строку

```
s = 'Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.'  
print(s.count('раз'))  
print(s.count('раз', 10))  
print(s.count('раз', 10, 30))
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/course  
3  
3  
1
```

# Поиск индекса первого вхождения подстроки в строку

► Метод `.find(sub, start, stop)` находит **индекс первого вхождения** подстроки `sub` в исходной строке. Если исходная строка не содержит подстроки `sub`, то метод возвращает значение **-1**. Метод `.rfind()` ищет вхождение справа.

```
s = 'Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.'  
print(s.find('Python'))  
if s.find('Python') >= 0:  
    print('Python присутствует')  
else:  
    print('Python отсутствует')
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/course  
-1  
Python отсутствует
```

# Удаление пробелов с краев строк

► Метод `.strip()` возвращает **копию** строки, у которой удалены все пробелы стоящие в начале и конце строки.

Доступны методы `.lstrip()` и `.rstrip()`

```
s = '  Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.  '
print(s.strip())
print(s)
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/course/block_6.py
Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.
  Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.
```

# Замена подстроки в строке

► Метод `.replace(old, new, k)` возвращает **копию** строки со всеми вхождениями подстроки **old**, замененными на **new** **k раз** (по умолчанию **k = бесконечность**)

```
s = 'Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.'  
print(s.replace('раз', '*РАЗ*'))  
print(s)
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/course  
Новый виток в *РАЗ*витии интернета. Разные технологии появляются в нашей жизни *РАЗ* за *РАЗ*ом.  
Новый виток в развитии интернета. Разные технологии появляются в нашей жизни раз за разом.
```

# Проверка что все элементы строки буквы или цифры

► Метод `.isalnum()` определяет, состоит ли исходная строка из буквенно-цифровых символов. Метод возвращает значение *True*, если исходная строка является непустой и состоит только из буквенно-цифровых символов и *False* в противном случае

```
s = 'Новый виток в развитии интернета! Разные технологии появляются в нашей жизни раз за разом'  
print(s.isalnum())  
print(s.replace('!', '').replace(' ', '').isalnum())
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/PycharmProjects/cou  
False  
True
```

# Проверка что все элементы строки буквы

► Метод *.isalpha()* определяет, состоит ли исходная строка из буквенных символов. Метод возвращает значение *True* если *исходная строка* является непустой и *состоит только из буквенных символов* и *False* в противном случае.

```
s = 'Исследуемая строка'
print(s.isalpha())
print(s.replace(' ', '').isalpha())
```

block\_6 ×

```
/Library/Frameworks/Python.framework/
False
True
```

# Проверка что все элементы строки числа

► Метод `.isdigit()` определяет, состоит ли исходная строка только из цифровых символов. Метод возвращает значение ***True*** если исходная строка является непустой и ***состоит только из цифровых символов*** и ***False*** в противном случае

```
s = '123.234.6.77'  
print(s.isdigit())  
print(s.replace('.', '').isdigit())
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions
```

```
False
```

```
True
```

# Проверка что все элементы строки строчные / прописных

► Метод *.islower()* / *.isupper()* определяет, состоит ли исходная строка *только из строчных / прописных букв*

```
s = 'все Строчные буквы'  
print(s)  
s = s.lower()  
print(s)  
print(s.islower())  
s = s.upper()  
print(s)  
print(s.isupper())
```

block\_6 ×

/Library/Frameworks/Python.framework/

все Строчные буквы

все строчные буквы

True

ВСЕ СТРОЧНЫЕ БУКВЫ

True

# **#6 | Маневры**

# Сумма цифр числа

► Используем *генератор* и функцию *sum* приводить к списку не обязательно

```
a = 7253478625765123123237452876345823764582736  
print(sum(int(x) for x in str(a)))
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10  
196
```

# Сумма чётных цифр числа

► Используем *генератор* и функцию *sum* приводить к списку не обязательно

```
a = 7253478625765123123237452876345823764582736  
print(sum(int(x) for x in str(a) if int(x)%2==0))
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10
```

```
94
```

# Получить список индексов гласных букв строки

- ▶ Используем *генератор*, метод *.lower()* и оператор *in*

```
a = 'автор программного кода неизвестен'  
print([i for i,x in enumerate(a) if x.lower() in 'ауеыюэёяию'])
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3  
[0, 3, 8, 11, 15, 17, 20, 22, 25, 27, 30, 34]
```

# Найти и вывести через пробел все слова-палиндромы

- ▶ Используем *генератор*, метод `.split()` и срезы

```
a = 'Шёл казак в шалаш на Кудыкину гору. Сзади дед летел в кабак.'  
print(*(x for x in a.split() if x[::-1]==x and len(x) > 1))
```

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Use  
казак шалаш дед летел
```

# Гласные налево, согласные направо

- ▶ Используем *генератор*, метод `.split()`, `.replace()`

```
a = 'Вывести гласные символы в начало строки, согласные в конец строки'  
s = ''  
for x in a.replace(' ', '').replace(',', ', '):  
    s = x + s if x.lower() in 'уеыаоэёяию' else s + x  
print(s)
```

Run

block\_6 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/ra  
иоеоеыаоиооааыоиеыаиеыВвстглснсмвлвнчлстркслснвкнцстрк
```

# Количество одинаковых пар символов подряд

- ▶ Используем цикл *for*, проход по индексам

```
a = 'ААРАВБОУМММММЛДЫВШШСТТЛВИЗЗ'
k = 0
for i in range(len(a) - 1):
    if a[i] == a[i+1]:
        k+=1
print(k)
```

block\_6 ×

/Library/Frameworks/Python.framework

9

# Число оканчивается на 123 / начинается на 765

► Используем *срезы*

```
def finish(x):  
    return str(x)[-3:] == '123'  
print(finish(765765867123))
```

block\_6 ×

/Library/Frameworks/Python.framework

True

# Все цифры числа чётные

- ▶ Используем функцию *all()*

```
def chet(x):  
    return all(int(i)%2==0 for i in str(x))  
print(chet(8064264806))
```

block\_6 ×

/Library/Frameworks/Python.framework/Versions

True

# Посчитать сколько пар четных и нечетных в числе

► Используем метод *.replace()*

```
def similar_count(x):  
    x = str(x)  
    for c in '02468':  
        x = x.replace(c, '*')  
    for c in '13579':  
        x = x.replace(c, '#')  
    print(x)  
    return sum(1 for i in range(len(x)) if x[i:i+2]=='**' or x[i:i+2]=='##')  
print(similar_count(806426480609234763255364563456354))
```

Run

block\_6 ×

/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin

\*\*\*\*\*#\*##\*#\*##\*#\*##\*#\*##\*#\*##\*#\*##\*

14

# Рекомендации

- ▶ Переменные лучше называть именем **s**, **названием строки**
- ▶ Лучше накапливать строки (**s += 'новые символы'**), а не выводить их через функцию **print()**, задавая в выводе параметр **end=""**
- ▶ Обращаться только к элементам строки по существующим индексам
- ▶ Перезаписывать строку после изменения её методами (**s = s.replace('11','\*')**)

# Задача блока 6

## ▶ НАУЧИТЬСЯ ГРАМОТНО РАБОТАТЬ СО СТРОКАМИ

```
s = 'Строка строченька'
```

```
s[1] = 'ёёёё'
```

```
s[-1:1000] = 'Новая  
строченька'
```

```
for i in range(len(s) + 2):  
    print(s[i])
```

```
del s[3]
```

