

PYTHON СИМВОЛЬНЫЕ СТРОКИ В PYTHON (часть 1)

(конкатенация строк; функции len();
str(); Функция chr(); Функция ord();
id, оператор in; Обращение по
индексу; Срезы;)
(Лекция 6)

#1 Основы работы со строками

#1 В Python есть несколько способов задания строк

```
str1 = 'Hello1'
```

```
str2 = "Hello2"
```

```
str3 = '''Многострочные  
строки 1''';
```

```
str4 = """Многострочные  
строки2""";
```

```
print(str1)
```

```
print(str2)
```

```
print(str3)
```

```
print(str4)
```

```
print('hello\nworld')
```

#2

z="""

PYTHON

СИМВОЛЬНЫЕ СТРОКИ

В PYTHON

(Лекция 6)

"""

print(z)

z="""

"""

print(z,len(z))#1

z="""""

print(z,len(z))#0

z="""

"""

print(z,len(z))#2

Для работы со строками в Python предусмотрено большое число встроенных функций, например, `len`. Эта функция определяет длину строки, которая передается ей в качестве аргумента.

```
>>> help(len)
```

```
Help on built-in function len in module builtins:
```

```
len(obj, /)
```

```
Return the number of items in a container.
```

```
>>> len('Привет!')
```

```
7
```

```
>>>
```

объединить несколько строк в одну - с помощью **операции конкатенации** (обычный символ + для строк):

```
>>> 'Привет, ' + 'земляне!'
'Привет, земляне!'
>>>
```

что если сложить число и строку?

```
>>> 'Марс' + 5
```

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

```
'Марс' + 5
```

TypeError: Can't convert 'int' object to str implicitly

```
>>>
```

Для этого с помощью функции `str` преобразуем число 5 в строку '5' и выполним объединение:

```
>>> 'Марс' + str(5)
```

```
'Марс5'
```

```
>>>
```

```
>>> int("-5")
```

```
-5
```

```
>>>
```

```
>>> "СПАМ" * 10
```

```
'СПАМСПАМСПАМСПАМСПАМСПАМСПАМ  
СПАМСПАМСПАМ'
```

#3 конкатенация строк

```
str1 = 'Hello'
```

```
str2 = "world!"
```

```
msg = str1+str2
```

```
print(msg)#Helloworld!
```

```
msg = str1+" "+str2
```

```
print(msg)#Hello world!
```

```
print("str1=",str1,"id(str1)=",id(str1))
```

```
str1=str1+str2
```

```
print("str1=",str1,"id(str1)=",id(str1))
```

```
# Helloworld!
```

```
# Hello world!
```

```
# str1= Hello id(str1)= 48792320
```

```
# str1= Helloworld! id(str1)= 48679736
```

#4 дублирование строки

```
one = 'ай '
```

```
msg = one*10
```

```
print(msg)
```

```
msg = one*3.5
```

```
# TypeError: can't multiply sequence by non-int of type 'float'
```

```
print(msg)
```

#5 len(<строка>)

```
one = 'ай '
```

```
print(len(one))
```


#6 соединять между собой можно только строки

```
dig = 5
```

```
msg = "число = "+dig
```

```
print(msg)
```

```
# msg = "число = "+dig
```

```
# TypeError: can only concatenate str (not "int") to str
```

#7 соединять между собой можно только строки

```
dig = 5
```

```
msg = "число = "+str(dig)
```

```
print(msg)#число = 5
```

#8 для проверки наличия в строке подстроки,
используется оператор in:
#<подстрока> in <строка>
Он возвращает True, если подстрока
присутствует и False, если отсутствует
s = "abcdefg0123"
print("abc" in s)
print('0' in s)
print('43' in s)

Символьные строки

Начальное значение:

```
s = "Привет!"
```



Строка – это
последовательность
символов!

Вывод на экран:

```
print ( s )
```

Сложение:

```
s1 = "Привет"
```

```
s2 = "Вася"
```

```
s = s1 + ", " + s2 + "!"
```

"Привет, Вася!"

Умножение:

```
s = "Ау"
```

```
s5 = s*5
```

```
s5 = s + s + s + s + s
```

АУАУАУАУАУАУ



Что получим?

Символьные строки

Вывод символа на экран:

```
print ( s[5] )
```

```
print ( s[-2] )
```

0	1	2	3	4	5	6
П	р	и	в	е	т	!
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]

s[len(s)-2]

Длина строки:

```
n = len ( s )
```

Обращение к символу по номеру

```
print ( s[5] )
```

```
print ( s[-2] )
```

0	1	2	3	4	5	6	<code>s[len(s)-2]</code>
П	р	и	в	е	т	!	
<code>s[0]</code>	<code>s[1]</code>	<code>s[2]</code>	<code>s[3]</code>	<code>s[4]</code>	<code>s[5]</code>	<code>s[6]</code>	



Символы нумеруются с нуля!

СОСТАВИТЬ «КОТ»

```
s = "информатика"
```

```
kot = s[-2]+s[3]+s[-4]
```

Каждый символ строки имеет свой порядковый номер (индекс). Нумерация символов начинается с нуля. Теперь мы можем обратиться к заданному символу строки следующим образом:

```
>>> s = 'Я люблю писать программы!'
```

```
>>> s[0]
```

```
'Я'
```

```
>>> s[-1]
```

```
'!'
```

```
>>>
```

В квадратных скобках указывается индекс символа. Нулевой индекс – первая буква строки. А -1 индекс? Можно догадаться, что последний.

#9 Обращение по индексу

```
s='Python'
```

```
print(s[1])
```

```
print('absd'[-1])
```

```
#print(s[10])#IndexError: string index out of range
```

```
print(s[len(s)-1])#n
```

```
for i in range(len(s)):
```

```
    print('i=',i,'s['',i,']= ',s[i])
```

```
    print('s['',i,']= ',s[i],sep='')
```

```
    print('s['',i,']= ',s[i],sep='')
```

```
#10 При работе с переменными в Python
# всегда следует помнить,
# что это лишь ссылки на соответствующие объекты
str1 = "сообщение"
str2 = str1
print('id(str1)=', id(str1))#id(str1)= 23167960
print('id(str2)=', id(str2))#id(str2)= 23167960
#получим две ссылки на один и тот же объект (одну строку).
# То есть, копирование строки не происходит!
```


В отличие от большинства современных языков программирования, в Python нельзя изменить символьную строку, поскольку строка — это неизменяемый объект.

Это значит, что оператор присваивания `s[5]="a"` не сработает — будет выдано сообщение об ошибке.

Тем не менее, можно составить из символов существующей строки новую строку, и внести в неё нужные изменения.

```
>>> s = 'Я люблю писать программы!'
```

```
>>> s[0] = 'J'
```

```
Traceback (most recent call last):
```

```
File "<pyshell#41>", line 1, in <module>
```

```
s[0] = 'J'
```

```
TypeError: 'str' object does not support item  
assignment
```

```
>>>
```

Попытка изменить нулевой символ в строке `s` привела к ошибке. Дело в том, что в Python строки, как и числа, являются неизменяемыми.

#11 строка - неизменяемый объект

```
s='vasya'
```

```
print(s,id(s))#58419968
```

```
#s[0]='V'#TypeError: 'str' object does not support  
item assignment
```

```
s_new=s='V'+s[1:]
```

```
print(s_new,id(s_new))#Vasya 62155104
```

```
s='V'+s[1:]
```

```
print(s,id(s))#Vasya 68774752
```

Цикл перебора символов

```
sNew = ""
```

```
for c in s:
```

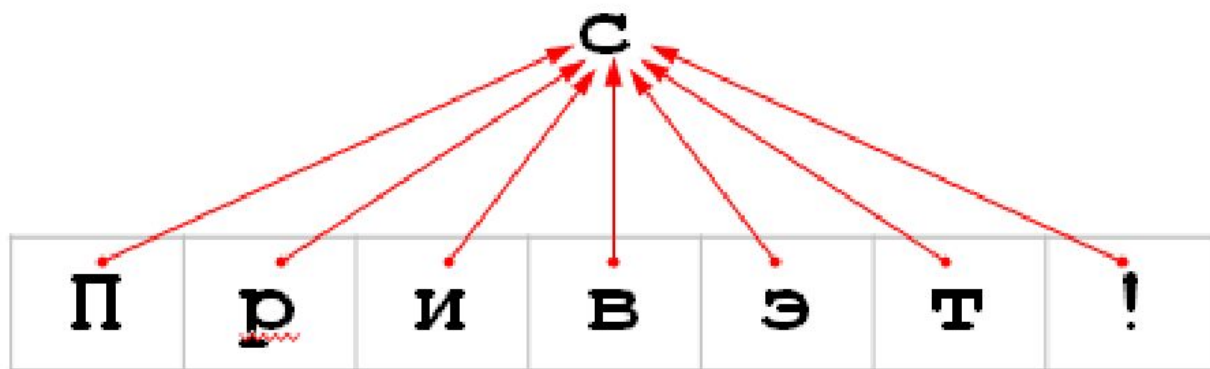
```
    if c == "э":
```

```
        sNew += "e"
```

```
    else:
```

```
        sNew += c
```

перебрать
все символы
строки



Посимвольная обработка строк

`s[4] = "a"` ❌



Строка неизменна!

Задача. Ввести строку и заменить в ней все буквы «э» на буквы «е».

строим новую строку!

```
sNew = ""
for i in range(len(s)):
    if s[i] == "э":
        sNew += "е"
    else:
        sNew += s[i]
```

для каждого символа строки

`len(s) - 1`

0	1	2	3	4	5	6
П	р	и	в	э	т	!

12

"""

перебираются все символы, входящие в строку s.

В теле цикла проверяем значение переменной c (это очередной символ исход-ной строки): если оно совпадает с буквой «э», то заменяем его на букву «е»

```
if c == "э": c = "е"
```

"""

```
s="ПривЭт"
```

```
sNew = ""
```

```
print("sNew",sNew,"id",id(sNew))
```

```
for c in s:
```

```
    if c == "э": c = "е"
```

```
    sNew += c
```

```
    print("sNew",sNew,"id",id(sNew))
```

```
print( sNew )
```

sNew id 16259808

sNew П id 16906064

sNew Пр id 46617232

sNew При id 46617232

sNew Прив id 46617232

sNew Приве id 46617232

sNew Привет id 16876504

Привет

Срезы

```
>>> s = 'Питоны водятся в Африке'
```

```
>>> s[1:3]
```

```
'ит'
```

```
>>>
```

`s[1:3]` – срез строки `s`, начиная с индекса 1, заканчивая индексом 3 (не включительно).

Срезы строк (выделение части строки)

```
s = "0123456789"  
s1 = s[3:8]          # "34567"
```

с какого
символа

до какого
(не **включая** 8)

Срезы (slices)

```
s = "0123456789"  
s1 = s[:8]           # "01234567"
```

от начала строки

```
s = "0123456789"  
s1 = s[3:]           # "3456789"
```

до конца строки

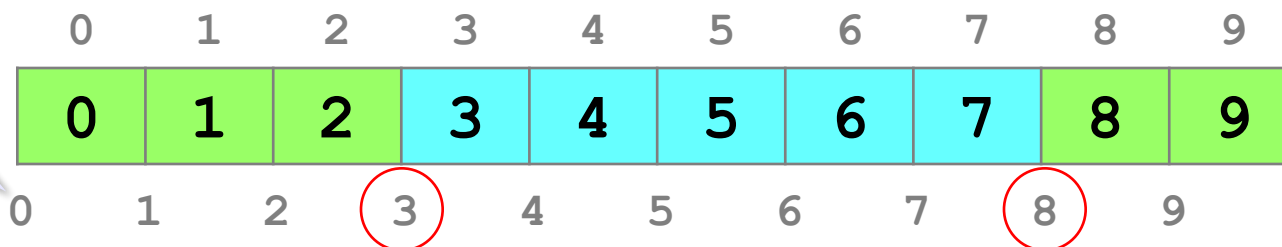
Срез (slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Срезы

```
s = "0123456789"
```

```
s1 = s[3:8] # "34567"
```

разрезы



Срезы строк

```
s = "0123456789"  
s1 = s[:8] # "01234567"
```

от начала строки

```
s = "0123456789"  
s1 = s[3:] # "3456789"
```

до конца строки

```
s1 = s[::-1] # "9876543210"
```

реверс строки

Операции со строками

Срезы с отрицательными индексами:

```
s = "0123456789"
```

```
s1 = s[: -2] # "01234567"
```

$\text{len}(s) - 2$

```
s = "0123456789"
```

```
s1 = s[-6: -2] # "4567"
```

$\text{len}(s) - 6$

$\text{len}(s) - 2$

Операции со строками

Удаление:

```
s = "0123456789"
s1 = s[:3] + s[9:]    # "0129"
      "012"      "9"
```

Вставка:

```
s = "0123456789"
s1 = s[:3] + "ABC" + s[3:]
      "012ABC3456789"
```

Со срезами можно производить различные манипуляции:

```
>>> s[:3] # с 0 индекса по 3-ий не включительно  
'Пит'
```

```
>>> s[:] # вся строка  
'Питоны водятся в Африке'
```

```
>>> s[::2] # третий аргумент задает шаг (по умолчанию  
один)  
'Птн ояс фие'
```

```
>>> s[::-1] # «обратный» шаг  
'екирфА в ясядов ынотиП'
```

```
>>> s[:-1] # вспомним, как мы находили отрицательный  
индекс
```

```
'Питоны водятся в Африк'
```

```
>>> s[-1:] # снова отрицательный индекс  
'е'
```

```
>>>
```

Срез с двумя параметрами: `S[a:b]`

возвращает подстроку из `b - a` символов, начиная с символа с индексом `a`, то есть до символа с индексом `b`, не включая его.

Например, `S[1:4] == 'ell'`, то же самое получится если написать `S[-4:-1]`.

Можно использовать как положительные, так и отрицательные индексы в одном срезе, например, `S[1:-1]` — это строка без первого и последнего символа (срез начинается с символа с индексом `1` и заканчивается индексом `-1`, не включая его).

Если опустить второй параметр (но поставить двоеточие), то срез берется до конца строки. Например, чтобы удалить из строки первый символ (его индекс равен 0), можно взять срез `S[1:]`.

Аналогично если опустить первый параметр, то можно взять срез от начала строки.

То есть удалить из строки последний символ можно при помощи среза `S[:-1]`.

Срез `S[:]` совпадает с самой строкой `S`.

Если задать срез с тремя параметрами `S[a:b:d]`, то третий параметр задает шаг, как в случае с функцией `range`, то есть будут взяты символы с индексами a , $a + d$, $a + 2 * d$ и т. д. При задании значения третьего параметра, равному 2, в срез попадет каждый второй символ, а если взять значение среза, равное -1, то символы будут идти в обратном порядке. Например, можно перевернуть строку срезом `S[::-1]`.

#Индексы и срезы строк

#13 Срезы

```
msg = "Hello World!"
```

```
print(msg[6:11]) #World
```

```
print(msg[6:12]) #World!
```

```
print(msg[2:5])#llo
```

```
print(msg[:5]) #Hello
```

```
print(msg[6:]) #World!
```

```
print(msg[:]) #Hello World!
```

```
copy = msg[:]
```

```
print(copy,id(copy), id(msg))#Hello World! 24070528 24070528
```

```
print(msg[::-2])#HloWrd
```

```
print(msg[:5:2])#Hlo
```

```
print(msg[6::2])#Wrd
```

```
print(msg[1:6:2])#el
```

```
print(msg[::-1])#!dlroW olleH
```

#14 срезы примеры

```
s='0123456789'
```

```
print(s)#0123456789
```

```
print(s[3:])#3456789
```

```
print(s[:4])#0123
```

```
print(s[:])#0123456789
```

```
print(s[::2])#02468
```

```
print(s[1::2])#13579
```

```
print(s[::-1])#9876543210
```

#15 Срезы (slices)

'''

Срез (slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

'''

```
s = 'abcdefg'
print(s[1])
print(s[-1])
print(s[1:3])
print(s[1:-1])
print(s[:3])
print(s[2:])
print(s[:-1])
print(s[::-2])
print(s[1::2])
print(s[::-1])
```

#16 Обратите внимание на то,
#как похож третий параметр среза
#на третий параметр функции range():
s = 'abcdefghijklm'
print(s[0:10:2]) #acegi

```
for i in range(0, 10, 2):  
    print(i, s[i])
```

"""

0 a

2 c

4 e

6 g

8 i

"""

#17Пример срезы

```
s = 'ABCDEFGH'
```

```
print(s[1]) #b
```

```
print(s[-1])#g
```

```
print(s[1:3]) #bc
```

```
print(s[1:-1])#bcdef
```

```
print(s[:3])#abc
```

```
print(s[2:])#cdefg
```

```
print(s[:-1])#abcdef
```

```
print(s[::2])#aceg
```

```
print(s[1::2])#bdf
```

```
print(s[::-1])#gfedcba
```

#17 Обратите внимание на то,
#как похож третий параметр среза
#на третий параметр функции range():
s = 'abcdefghijklm'
print(s[0:10:2]) #acegi

```
for i in range(0, 10, 2):  
    print(i, s[i])
```

"""

0 a

2 c

4 e

6 g

8 i

"""

Преобразования «строка» → «число»

Из строки в число:

```
s = "123"
N = int ( s )          # N = 123
s = "123.456"
X = float ( s )        # X = 123.456
```

Из числа в строку:

```
N = 123
s = str ( N )          # s = "123"
s = "{:5d}".format(N)  # s = " 123"

X = 123.456
s = str ( X )          # s = "123.456"
s = "{:7.2f}".format(X) # s = " 123.46"
s = "{:10.2e}".format(X) # s = " 1.23e+02"
```

#18 Как напечатать первую цифру любого целого числа?

```
x='12345678987654325'
```

```
print(x)
```

```
#print(len(x))#17
```

```
#print('x[0]=' ,x[0]) #x[0]= 1
```

#Как напечатать последнюю цифру любого целого числа?

```
print(x[len(x)-1])#5
```


В Python для этого есть специальная функция `input`:

```
>>> s = input()
```

Земляне, мы прилетели с миром!

```
>>> s
```

```
'Земляне, мы прилетели с миром!'
```

```
>>> type(s)
```

```
<class 'str'>
```

```
>>>
```

ВНИМАНИЕ: не забывайте, что функция `input` возвращает строковый объект!

19

""""

Сравнение строк

Строки можно сравнивать
между собой так же, как числа.

Например, можно проверить
равенство двух строк:

""""

```
password = input( "Введите пароль:" )
```

```
if password == "Sergey":
```

```
    print( "Слушаюсь и повинуюсь!" )
```

```
else:
```

```
    print( "No pasaran!" )
```

20 возвращает True, если строки не
равны

и False в противном случае.

```
psw = "pass"
```

```
in_psw = ""
```

```
while psw != in_psw:
```

```
    in_psw = input("Введите пароль: ")
```

```
print("Вход в систему разрешен")
```

Пример 21

"""

«паровоз» будет «меньше»,

чем слово «пароход»:

они отличаются в пятой букве и «В» < «Х».

"""

```
s1 = "паровоз"
```

```
s2 = "пароход"
```

```
if s1 < s2:
```

```
    print( s1, "<", s2 )
```

```
elif s1 == s2:
```

```
    print( s1, "=", s2 )
```

```
else:
```

```
    print( s1, ">", s2 )
```

Кодировка UNICODE

Юникод (Unicode) — стандарт кодирования символов, позволяющий представить знаки практически всех письменных языков. Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода».

В Unicode используются 16-битовые (2-байтовые) коды, что позволяет представить 65536 символов.

Применение стандарта Unicode позволяет закодировать очень большое число символов из разных письменностей: в документах Unicode могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становится ненужным переключение кодовых страниц.

UTF-8 и Unicode не могут сравниваться. UTF-8 является кодировкой используется для перевода чисел в двоичные данные. Unicode - это набор символов используется для перевода символов в числа.

UTF-8 и Unicode не могут сравниваться. UTF-8 является кодировкой используется для перевода чисел в двоичные данные. Unicode - это набор символов используется для перевода символов в числа.

Python 3: всё на Юникоде

Python 3 полностью реализован на Юникоде, а точнее на UTF-8. Вот что это означает:

По умолчанию предполагается, что исходный код Python 3 написан с помощью UTF-8. Это значит, что вам не нужно использовать определение

```
# -*- coding: UTF-8 -*-
```

- в начале файлов .py в этой версии языка.

Пример 22 Коды символов

""""

Функция `ord()` позволяет получить номер символа по таблице Unicode. Соответственно, принимает она в качестве аргумента одиночный символ, заключенный в кавычки:

""""

```
print("ord(a)=",ord("a"))#ord(a)= 97
print("ord(A)=",ord("A"))#ord(A)= 65
print("ord(z)=",ord("z"))#ord(z)= 122
print("ord(ϕ)=",ord("ϕ"))#ord(ϕ)= 1092
print("ord(Φ)=",ord("Φ"))#ord(Φ)= 1060
print("ord(В)=",ord("В"))#ord(В)= 1074
print("ord(х)=",ord("х"))#ord(х)= 1093
```

#Пример 23. Вывод таблицы СИМВОЛОВ

```
for i in range(32, 128):  
    print("i=",i, chr(i), end=' ')
```

```
    if (i - 1) % 10 == 0:  
        print()
```

```
print()
```


i= 32 i= 33 ! i= 34 " i= 35 # i= 36 \$ i= 37 % i= 38 & i= 39 ' i= 40 (i= 41)
i= 42 * i= 43 + i= 44 , i= 45 - i= 46 . i= 47 / i= 48 0 i= 49 1 i= 50 2 i= 51 3
i= 52 4 i= 53 5 i= 54 6 i= 55 7 i= 56 8 i= 57 9 i= 58 : i= 59 ; i= 60 < i= 61 =
i= 62 > i= 63 ? i= 64 @ i= 65 A i= 66 B i= 67 C i= 68 D i= 69 E i= 70 F i= 71 G
i= 72 H i= 73 I i= 74 J i= 75 K i= 76 L i= 77 M i= 78 N i= 79 O i= 80 P i= 81 Q
i= 82 R i= 83 S i= 84 T i= 85 U i= 86 V i= 87 W i= 88 X i= 89 Y i= 90 Z i= 91 [
i= 92 \ i= 93] i= 94 ^ i= 95 _ i= 96 ` i= 97 a i= 98 b i= 99 c i= 100 d i= 101 e
i= 102 f i= 103 g i= 104 h i= 105 i i= 106 j i= 107 k i= 108 l i= 109 m i= 110 n i= 111 o
i= 112 p i= 113 q i= 114 r i= 115 s i= 116 t i= 117 u i= 118 v i= 119 w i= 120 x i= 121 y
i= 122 z i= 123 { i= 124 | i= 125 } i= 126 ~ i= 127 □

Функция chr()

позволяет получить символ по его
номеру

"""

```
print(87, chr(87)) #87 W
```

Выводы:

- Символьная строка – это последовательность символов.
- Длина строки – это количество символов в строке.
- Подстрока – это часть символьной строки.
- При обращении к отдельному символу строки его номер записывают в квадратных скобках.
- Нумерация символов в строке в языке Python начинается с нуля.
- Знак «+» при работе со строками означает объединение (кон-катенацию) строк.

- Для обработки символьных строк используют встроенные функции стандартной библиотеки.
- Функция поиска подстроки возвращает номер символа, с которого начинается подстрока, или -1 в случае неудачи.
- Строку можно преобразовать в число для того, чтобы затем выполнять с ним вычисления.
- Число можно преобразовать в символьную строку.

Если хотим поместить разные виды кавычек в строку, то сделать это можно несколькими способами:

```
>>> "Hello's"
```

```
"Hello's"
```

```
>>> 'Hello\'s'
```

```
"Hello's"
```

```
>>>
```

Полезно знать об этих символах, т.к. они часто используются при работе со строками:

`\n` - перевод на новую строку

`\t` - знак табуляции

`\\` - наклонная черта влево

`\'` - символ одиночной кавычки

`\"` - символ двойной кавычки

Передадим на вход функции print строку со специальным символом:

```
>>> print('Это длинная\nстрока')
```

Это длинная
строка

```
>>>
```

```
>>> print(1, 3, 5)
```

1 3 5

```
>>> print(1, '2', 'снова строка', 56)
```

1 2 снова строка 56

```
>>>
```