Введение в Python

Лекция 3: Строки и операции над ними

Для работы с текстом в Python предусмотрен специальный строковый тип данных **`str`**

Python создает строковые объекты, если текст поместить в одинарные или двойные кавычки:

```
>>> 'hello'
'hello'
>>> "Hello"
'Hello'
>>>
```

Без кавычек Python расценит текст как переменную и попытается вывести на экран ее содержимое (если такая переменная была создана):

```
>>> hello
Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module>
hello
NameError: name 'hello' is not defined
>>>
```

Можно создать пустую строку:

```
>>> ''
''
>>>
```

Для работы со строками в Python предусмотрено большое число встроенных функций, например, **len**(). Она 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
>>>
```

Например, если мы хотим объединить несколько строк в одну, Python позволяет это сделать с помощью операции конкатенации (обычный символ + для строк):

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

Например, надо объединить строки. Для этого с помощью функции `str()` преобразуем число `5` в строку `'5'` и выполним объединение:

```
>>> 'Марс' + str(5)
'Марс5'
>>>
```

Например, обратное преобразование типов:

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

Повтор строки заданное число раз:

```
>>> "СПАМ" * 10
'СПАМСПАМСПАМСПАМСПАМСПАМСПАМСПАМСПАМСПАМ'
>>>
```

Строки можно присваивать переменным и дальше работать с переменными:

```
>>> s = "Я изучаю программирование"
>>> s
'Я изучаю программирование'
>>> s*4
'Я изучаю программированиеЯ изучаю программированиеЯ
изучаю программированиеЯ изучаю программирование'
>>> s + " на языке Python"
'Я изучаю программирование на языке Python'
>>>
```

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

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

Использование специальных символов (управляющие escape-последовательности), которые записываются, как два символа, но Python видит их как один:

```
>>> len("\'")
1
>>>
```

Управляющие escape - последовательности

```
\n - переход на новую строку
\t - знак табуляции
\\ - наклонная черта влево
\ ' - символ одиночной кавычки
\ " - символ двойной кавычки
```

Многострочная строка (заключается в три одинарные кавычки):

```
>>> '''Это длинная
строка'''
'Это длинная\nстрока'
>>>
```

Строка со специальным симвjлом с функцией **print**

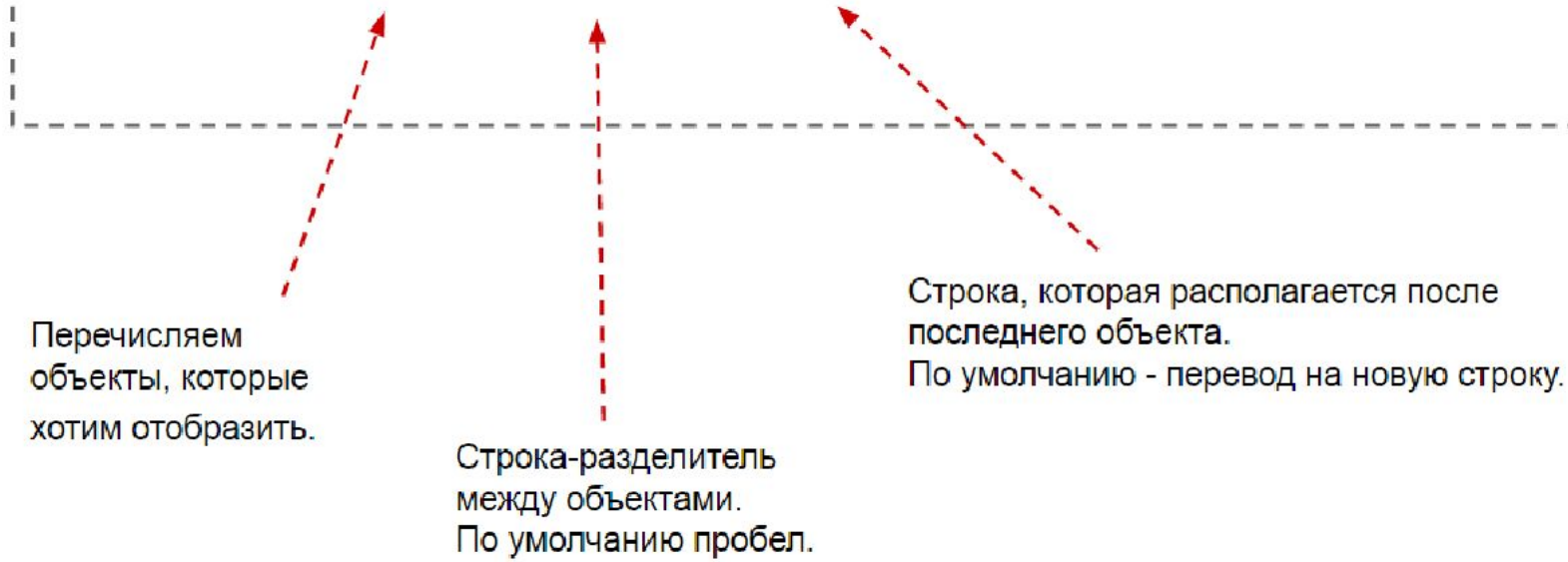
```
>>> print('Это длинная\nстрока')
Это длинная
строка
>>>
```

Примеры

```
>>> print(1, 3, 5)
1 3 5
>>> print(1, '2', 'снова строка', 56)
1 2 снова строка 56
>>>
```

```
>>> help (print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```



Примеры

```
>>> print(1, 6, 7, 8, 9)
1 6 7 8 9
>>> print(1, 6, 7, 8, 9, sep=':')
1:6:7:8:9
>>>
```

Операции над строками

Каждый символ строки имеет свой порядковый номер (*индекс*). Нумерация символов начинается с нуля.

Пример:

```
>>> s = 'Я люблю писать программы!'
>>> s[0]
'Я'
>>> s[-1]
'!'
>>>
```

Отрицательный индекс: `длина строки + отрицательный индекс`. Например, для `-1` это будет: `len(s)-1`, т.е. `24`.

```
>>> len(s)-1
24
>>> s[24]
'!'
>>>
```

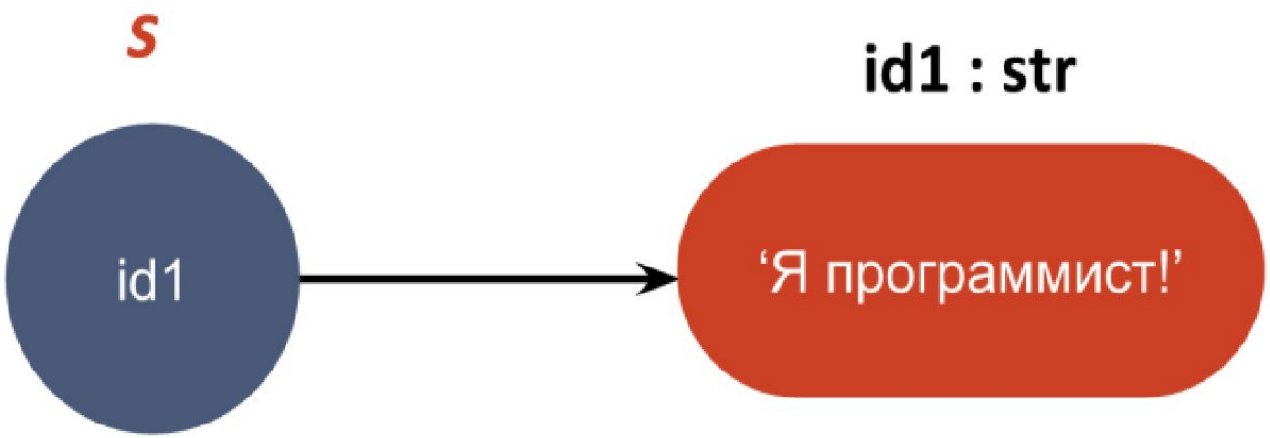
Пример:

```
>>> 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, так как в Python строки, как и числа, являются неизменяемыми.

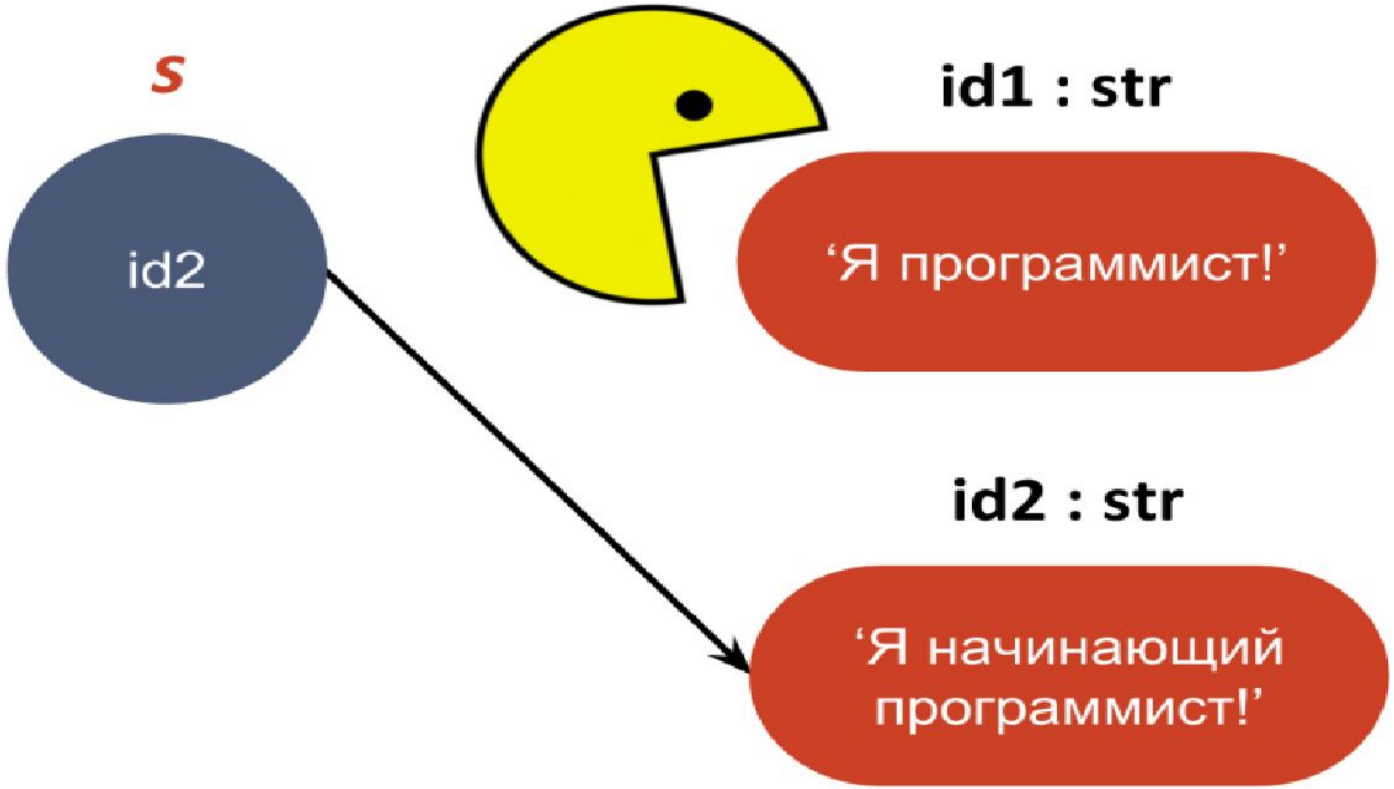
Работа со строковыми объектами для Python не отличается от работы с числовыми объектами:

```
>>> s = ‘Я программист!’
```



Изменяем значение переменной `s`. Создается новый строковый объект (а не изменяется предыдущий) по адресу `id2` и этот адрес записывается в переменную `s`.

```
>>> s = ‘Я программист!’
>>> s = ‘Я начинающий программист!’
```

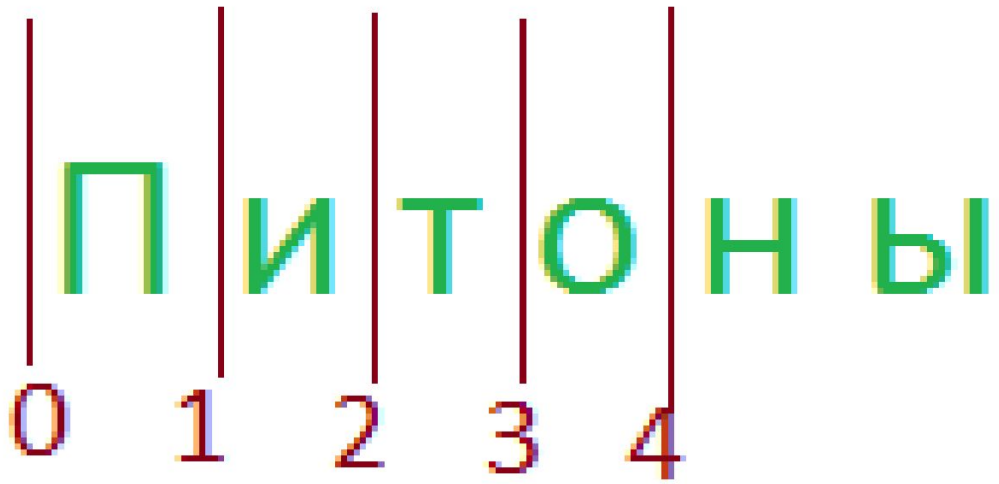


Прежде чем мы поймем, как строки можно изменять, познакомимся со срезами:

```
>>> s = 'Питоны водятся в Африке'
>>> s[1:3]
'ит'
>>>
```

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

Это легко запомнить, если индексы представить в виде смещений:



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 ястядов ынотиП'
>>> s[:-1] # вспомним, как мы находили отрицательный индекс
'Питоны водятся в Африк'
>>> s[-1:] # снова отрицательный индекс
'е'
>>>
```

Вернемся к вопросу, как изменить первый символ в строке?

```
>>> s = 'Я люблю писать программы!'
>>> 'J' + s[1:]
'J люблю писать программы!'
>>>
```