Python II:
Строки

Строки представляют собой последовательность символов, заключенных в кавычки (одинарные, двойные или тройные).

```
my_string = "Hello, world!"
```

В Python строки являются неизменяемыми объектами, то есть, если вы хотите изменить строку, то вам нужно создать новую строку.

Вот несколько методов, которые можно использовать со строками:

len(): возвращает длину строки (количество символов):

```
my_string = "Hello, world!"
print(len(my_string))  # 13
```

lower(): возвращает строку в нижнем регистре:

```
my_string = "Hello, world!"
print(my_string.lower())  # hello, world!
```

upper(): возвращает строку в верхнем регистре:

```
my_string = "Hello, world!"
print(my_string.upper())  # HELLO, WORLD!
```

replace(): заменяет подстроку в строке на другую подстроку:

```
my_string = "Hello, world!"
print(my_string.replace("world", "Python"))  # Hello, Python!
```

split(): разделяет строку на список подстрок, используя заданный разделитель:

```
my_string = "Hello, world!"
print(my_string.split(","))  # ['Hello', ' world!']
```

join(): используется для объединения списка строк в одну строку, разделенную определенным разделителем:

```
разделитель.join(список_строк)
```

где *разделитель* - это символ, который будет использоваться для разделения строк, а *список_строк* - это список строк, который нужно объединить в одну строку.

```
# объединение списка строк с пробелом в качестве  разделителя
my_list = ['Hello', 'world!']
my_string = " ".join(my_list)
print(my_string)  # "Hello world!"

# объединение списка строк с запятой в качестве разделителя
my_list = ['apple', 'banana', 'cherry']
my_string = ",".join(my_list)
print(my_string)  # "apple,banana,cherry"

# объединение списка строк с новой строкой в качестве
разделителя
my_list = ['apple', 'banana', 'cherry']
my_string = "\n".join(my_list)
print(my_string)
# "apple
# banana
# cherry"
```

strip(): используется для удаления указанных символов (или пробельных символов) в начале и в конце строки. Если символы не указаны, метод strip() удаляет все пробельные символы (такие как пробел, табуляция и новая строка) в начале и в конце строки:

```
# удаление пробельных символов из начала и конца строки
my_string = "  hello  "
print(my_string.strip())  # "hello"

# удаление указанных символов из начала и конца строки
my_string = "--hello--"
print(my_string.strip("-")) # "hello"

# удаление нескольких указанных символов из начала и конца строки
my_string = "-*-hello-*-"
print(my_string.strip("-*"))  # "hello"
```

Срезы:

В Python вы можете использовать срезы (slicing) для извлечения подстрок из строк, а также для извлечения частей списка, кортежа или других последовательностей.
Срезы используются в следующем формате:

```
sequence[start:end:step]
```

где:

- sequence - последовательность, из которой вы хотите извлечь подстроку или часть
- start - начальный индекс, с которого начинается срез (включительно)
- end - конечный индекс, до которого идет срез (не включительно)
- step - шаг, с которым извлекаются элементы (необязательный аргумент)

Например, если у вас есть строка 'Hello, world!', вы можете извлечь подстроку 'world' с помощью среза:

```
s = 'Hello, world!'
print(s[7:12]) # 'world'
```

Аргумент step позволяет указать шаг, с которым нужно извлекать элементы. Например, чтобы извлечь каждую вторую букву из строки, можно использовать следующий срез:

```
s = 'Hello, world!'
print(s[::2]) # 'Hlo ol!'
```

Также можно использовать отрицательные индексы, чтобы начать срез с конца последовательности. Например, чтобы извлечь последние 3 символа из строки, можно использовать следующий срез:

```
s = 'Hello, world!'
print(s[-3:]) # 'ld!'
```

Существует еще несколько интересных свойств и возможностей срезов. Если в срезе не указать start или end, то Python автоматически подставит соответствующий индекс начала или конца последовательности. Например:

```
s = 'Hello, world!'
print(s[:5]) # 'Hello'
print(s[7:]) # 'world!'
```

Срезы можно использовать для копирования строки. Например:

```
s = 'Hello, world!'
t = s[:] # создание копии строки s
t = t.upper() # преобразование к верхнему регистру
print(s) # 'Hello, world!'
print(t) # 'HELLO, WORLD!'
```

Можно использовать отрицательный step, чтобы инвертировать последовательность. Например:

```
s = 'Hello, world!'
print(s[::-1]) # '!dlrow ,olleH'
```

Дополнительно о строках:

Строки в Python можно объединять с помощью оператора +. Например:

```
s1 = 'Hello, '
s2 = 'world!'
s = s1 + s2
print(s) # 'Hello, world!'
```

Можно умножать строки на число, чтобы повторить строку нужное количество раз. Например:

```
s = 'Hello, world!'
s = s * 3
print(s) # 'Hello, world!Hello, world!Hello, world!'
```

Строки в Python являются неизменяемыми объектами, то есть после создания строки ее нельзя изменить. Например:

```
s = 'Hello, world!'
s[0] = 'h' # ошибка, строку нельзя изменить
```

Строки в Python можно преобразовывать в список символов с помощью функции list(). Например:

```
s = 'Hello, world!'
lst = list(s)
print(lst) # ['H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r',
'l', 'd', '!']
```