|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Пример** |
| **len(строка)** | Возвращает длину строки | print(len(**"my\_string"**))  -> 9 |
| **строка.split(<разделитель>)** | Разбить строку по разделителю | print(**"раз два три"**.split())  -> ['раз', 'два', 'три'] print(**"четыре\_пять\_шесть"**.split(**'\_'**))  -> ['четыре', 'пять', 'шесть'] |
| **<разделитель>.join(список)** | Собрать строку из списка с указанным разделителем | print(**'\_'**.join([**'раз'**, **'два'**, **'три'**]))   -> раз\_два\_три  print(**''**.join([**'раз'**, **'два'**, **'три'**]))  -> раздватри |
| **строка.title()** | Перевести первую букву каждого слова в верхний регистр, остальные - в нижний | print(**"ехал грека через реку"**.title())  -> Ехал Грека Через Реку |
| **строка.upper()** | Преобразовать строку к верхнему регистру | print(**'простая строка'**.upper())  -> ПРОСТАЯ СТРОКА |
| **строка.lower()** | Преобразовать строку к нижнему регистру | print(**'ПРОСТАЯ СТРОКА'**.lower())  -> простая строка |
| **строка.istitle()** | Проверить, начинаются ли слова строки с буквы в верхнем регистре | print(**'Ехал Грека Через Реку'**.istitle()) -> True  print(**'Ехал Грека Через реку'**.istitle()) -> False |
| **строка.isupper()** | Проверить, состоит ли строка из символов в верхнем регистре | print(**'ПРОСТАЯ СТРОКА'**.isupper())   -> True  print(**'простая строка'**.isupper())   -> False |
| **строка.islower()** | Проверить, состоит ли строка из символов в нижнем регистре | print(**'простая строка'**.islower())  -> True  print(**'ПРОСТАЯ СТРОКА'**.islower())  -> False |
| **ord(символ)** | Получить ASCII-код для символа | print(ord(**'b'**))  -> 98 |
| **chr(код)** | Получить символ по ASCII-коду | print(chr(98))  -> **'b'** |
| **строка.count(подстрока, [начало], [конец])** | Вернуть количество вхождений подстроки в строку | print(**'рарара'**.count(**'ра'**))  -> 3  print(**'рарара'**.count(**'ра'**, 2, 4))  -> 1 |
| **строка.capitalize()** | Перевести первый символ строки в верхний регистр, остальные - в нижний | print(**'cТРОКА'**.capitalize())  -> Cтрока |
| **строка.startswith(шаблон)** | Проверить, начинается ли строка с шаблона | print(**'рарара'**.startswith(**'ра'**))  -> True  print(**'рарара'**.startswith(**'не'**))  -> False |
| **строка.endswith(шаблон)** | Проверить, заканчивается ли строка шаблоном | print(**'рарара'**.endswith(**'ра'**))  -> True  print(**'рарара'**.endswith(**'не'**))  -> False |
| **строка.replace(шаблон, замена)** | Заменить в строке шаблон на указанную подстроку | print(**'рарара'**.replace(**'ра'**, **'не'**))  -> **'ненене'** |
| **строка.index(подстрока, [начало], [конец])** | Найти подстроку в строке. Получить позицию первого вхождения или получить ValueError | print(**'рарарара'**.index(**'ра'**))  -> 0  print(**'рарарара'**.index(**'ра'**, 4, 6))  -> 4  print(**'рарарара'**.index(**'ра'**, 10, 20))  -> ValueError: substring not found |

|  |  |  |
| --- | --- | --- |
| **строка.find(подстрока, [начало], [конец])** | Найти подстроку в строке. Получить позицию первого вхождения или получить -1 | print(**'рарарара'**.find(**'ра'**))  -> 0  print(**'рарарара'**.find(**'ра'**,4,6))   -> 4  print(**'рарарара'**.find(**'ра'**,10,20))  -> -1 |
| **str.format(\*args, \*\*kwargs)** | Данный метод позволяет вставить в указанные места указанные аргументы, с выполнением их предварительного форматирования. | >>> s = 'карма: {0}, рейтинг: {1}, имя - {name}, ник - {nik\_name}'  >>> s.format(711, 1546, name = 'Sam', nik\_name = 'SamProfi')  'карма: 711, рейтинг: 1546, имя - Sam, ник - SamProfi' |
| **str.center(width[, fillchar])** | Возвращает новую строку длинной width в которой исходная строка находится в центре, а справа и слева от нее находятся символы указанные в fillchar (по умолчанию пробел). | >>> 'A'.center(11)  ' A '  >>> 'A'.center(11, 'a')  'aaaaaAaaaaa' |
| **str.isdigit()** | Возвращает True если строка не пустая и состоит только из десятичных цифр и символов, которые так же отосятся к цифрам. | >>> ''.isdigit()  False  >>> '0123456789'.isdigit()  True  >>> '0.1'.isdigit()  False  >>> 'a'.isdigit()  False |
| **str.rindex(sub[, start[, end]])** | Выполняет тоже самое, что и метод rfind(), но в случае если не найдено ни одного вхождения подстроки str он вызывает исключение ValueError, а не возвращает -1. | >>> s = 'ABC\_ABC\_ABC\_ABC'  >>> s.rfind('AD')  -1  >>> s.rindex('AD')  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  ValueError: substring not found |
| **str.rfind(sub[, start[, end]])** | Возвращает индекс последнего вхождения подстроки sub в строке str. | >>> s = 'ABC\_ABC\_ABC\_ABC'  >>> s.rfind('A')  12  >>> s[12]  'A' |
| **str.strip([chars])** | Возвращает копию исходной строки в начале и конце которой удалены указанные символы chars. Если параметр chars не указан, то по умолчанию удаляются пробельные символы | >>> ' xxx '.rstrip()  ' xxx'  >>> ' xxx '.lstrip()  'xxx '  >>> 'abcxxxabc'.rstrip('cba')  'abcxxx'  >>> 'abcxxxabc'.lstrip('cba')  'xxxabc' |
| **str.lstrip([chars])** | Возвращает копию исходной строки в которой слева удалены указанные символы (по умолчанию удаляются пробельные символы). | >>> ' xxx'.lstrip()  'xxx'  >>> 'xxxyyyzzz'.lstrip('x')  'yyyzzz'  >>> 'xxxyyyzzz'.lstrip('xy')  'zzz'  >>> 'xxxyyyzzz'.lstrip('xyz')  '' |
| **str.rstrip([chars])** | Аналогичен методу lstrip(), но удаление удаление пробелов или указанных символов chars происходит справа а не слева. | >>> ' xxx '.rstrip()  ' xxx'  >>> ' xxx '.lstrip()  'xxx '  >>> 'abcxxxabc'.rstrip('cba')  'abcxxx'  >>> 'abcxxxabc'.lstrip('cba')  'xxxabc' |
| **str.rsplit(sep=None, maxsplit=-1)** | Аналогичен методу split() с тем лишь отличием, что если параметр maxsplit указан, то разбиение выполняется справа налево. | >>> '1 2 3 4 5 6 7'.rsplit(maxsplit = 3)  ['1 2 3 4', '5', '6', '7']  >>> '1->2->3->4->5->6'.rsplit('->', maxsplit = 3)  ['1->2->3', '4', '5', '6'] |
| **str.partition(sep)** | Разбивает строку по указанному разделителю и возвращает кортеж из трех элементов: строка до разделителя, сам разделитель и строка после разделителя. | >>> 'xxxxYzzzz'.partition('Y')  ('xxxx', 'Y', 'zzzz') |

|  |  |  |
| --- | --- | --- |
| **str.casefold()** | Возвращает копию строки в которой все символы приводятся к нижнему регистру в соответствии с разделом 3.13 стандарта Юникода. |  |
| **str.encode(encoding="utf-8", errors="strict")** | Декодирует строку str в указанную в параметре encoding кодировку и возвращает результат в виде строки типа bytes. |  |
| **str.expandtabs(tabsize=8)** | Разбивает строку по символам \t на столбцы указанной ширины. |  |
| **str.format\_map(mapping)** | Тоже что и str.format(\*\*mapping), но позволяет использовать собственные отображения (объекты типа "ключ - значение"), которые используются напряму, без копирования. |  |
| **str.isalnum()** | Возвращает True если строка не пустая и состоит только из букв и цифр. |  |
| **str.isalpha()** | Возвращает True если строка не пустая и состоит только из букв. |  |
| **str.isdecimal()** | Возвращает True если строка не пустая и состоит только из десятичных цифр. |  |
| **str.isidentifier()** | Возвращает True если строка является допустимым идентификатором языка Python. |  |
| **str.isnumeric()** | Возвращает True если строка не пустая и состоит только из десятичных цифр и символов, которые так же отосятся к цифрам. |  |
| **str.isprintable()** | Возвращает True если все символы в строке являются печатаемыми или если строка является пустой. |  |
| **str.isspace()** | Возвращает True если строка не пуста и состоит только из пробельных символов, в противном случае возвращается False. |  |
| **str.ljust(width[, fillchar])** | Возвращает новую строку в которой исходная строка str дополнена справа символами fillchar (по умолчанию это пробел ASCII) до указанной длины width. |  |
| **static str.maketrans(x[, y[, z]])** | Возвращает таблицу преобразования символов, используемую методом translate(). Содержание строки, относительно которой вызывается метод maketrans() не имеет никакого значения. |  |
| **str.rjust(width[, fillchar])** | Возвращает новую строку в которой исходная строка дополнена слева указанным в параметре width количеством символов fillchar (по умолчанию – это пробел). |  |
| **str.rpartition(sep)** | Разбивает строку по последнему встреченному разделителю sep и возвращает кортеж, который состоит из трех элементов: строки до разделителя, самого разделителя и строки после разделителя. |  |
| **str.splitlines([keepends])** | Разбивает строку на подстроки по фиксированному набору разделителей и возвращает их в списке. |  |
| **str.swapcase()** | Возвращает копию исходной строки в которой все заглавные буквы преобразованы в строчные, строчные – в заглавные. |  |
| **str.translate(table)** | Преобразует символы исходной строки в соответствии с правилами преобразования в объекте table. |  |
| **str.zfill(width)** | Возвращает новую строку, в которой исходная строка str дополнена нулями слева, так что бы длина новой строки стала равна width. |  |
| **ИСТОЧНИК: pypprog.pro** |

**Методы списков**

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| **result\_list.append(el)** | Добавить элемент el в конец списка result\_list |
| **result\_list.extend(my\_list)** | Расширить список result\_list —  добавить в конец элементы списка my\_list |
| **result\_list.insert(pos, el)** | Разместить на позиции pos (индекс элемента списка) элемент el |
| **result\_list.remove(el)** | Удалить из списка первый элемент со значением el |
| **result\_list.pop(pos)** | Удалить элемент с индексом pos |
| **result\_list.index(el)** | Получить позицию (индекс) первого элемента со значением el |
| **result\_list.count(el)** | Возвращает количество элементов списка со значением el |
| **result\_list.sort([key=функция])** | Выполнить сортировку списка на основе указанной функции |
| **result\_list.reverse()** | Выполнить реверс списка (развернуть список) |
| **result\_list.copy()** | Вернуть копию списка |
| **result\_list.clear()** | Очистить список |

**Методы словарей**

|  |  |
| --- | --- |
| **.keys()** | Возвращает список ключей словаря |
| **.values()** | Возвращает список значений |
| **.items()** | Возвращает список кортежей (ключ, значение) |
| **.get(key)** | Возвращает значение, соответствующее ключу key. Если ключ отсутствует, возвращает значение None |
| **.popitem()** | Удаляет элемент словаря и возвращает пару (ключ, значение). Если элементы отсутствуют, возникает исключение KeyError |
| **.setdefault(key)** | Возвращает значение, соответствующее ключу. Если ключ отсутствует, создаётся элемент с указанным ключом и значением None |
| **.pop(key)** | Удаляет ключ и возвращает значение, соответствующее ключу |
| **.update(new\_dict)** | Добавляет пары (ключ, значение) в текущий словарь из словаря new\_dict. Имеющиеся ключи перезаписываются |
| **.copy()** | Возвращает копию словаря |
| **.clear()** | Очищает словарь |

**Методы множеств**

|  |  |
| --- | --- |
| **.add(el)** | Добавить элемент в множество |
| **.remove(el)** | Удалить элемент из множества. Если элемент отсутствует — ошибка KeyError |
| **.discard(el)** | Удалить элемент из множества |
| **.pop()** | Удалить первый элемент из множества. Множества не упорядочены, поэтому первый элемент множества заранее не определён |
| **.copy()** | Создать копию множества |
| **.clear()** | Очистить множество |