Краткая Документация по open (Мой ручной перевод)

**open**(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Открывает файл и возвращает соответствуйщий объект файла. Если файл не может быть открыт, появляется ошибка OSError

File – указывается путь (абсолютный или относительный) открываемого файла.

Mode – необязательная строка определяющая режим открытия файла

| **Character** | **Meaning** |
| --- | --- |
| 'r' | Открыт для чтения (по умолчанию) |
| 'w' | открыть для записи, сначала обрезав файл, если файла нет - исключение |
| 'x' | открыть для записи, сначала обрезав файл, если файла нет – создание нового |
| 'a' | Продолжение записи |
| 'b' | Двоичный режим(слишком сложно) |
| 't' | Текстовый режим (открывает в формате текста) (по умолчанию) |
| '+' | Чтение и запись (слишком сложно, не рекомендую) |

По умолчанию стоит “rt” – открытие в текстовом режиме для чтения. Далее объяснять не буду, объясню в других сериях.

buffering - необязательное целое число, используемое для установки политики буферизации. Пропустите 0, чтобы выключить буферизацию (разрешено только в двоичном режиме), 1, чтобы выбрать буферизацию строки (можно использовать только в текстовом режиме), и целое число> 1, чтобы указать размер в байтах буфера фрагмента фиксированного размера. Если аргумент буферизации не задан, политика буферизации по умолчанию работает следующим образом:

* Двоичные файлы буферизуются кусками фиксированного размера; размер буфера выбирается с помощью эвристики, пытающейся определить «размер блока» базового устройства и возвращающейся к нему [io.DEFAULT\_BUFFER\_SIZE](https://docs.python.org/3/library/io.html" \l "io.DEFAULT_BUFFER_SIZE" \o "io.DEFAULT_BUFFER_SIZE). Во многих системах размер буфера обычно составляет 4096 или 8192 байта.
* «Интерактивные» текстовые файлы (файлы, для которых [isatty()](https://docs.python.org/3/library/io.html" \l "io.IOBase.isatty" \o "io.IOBase.isatty) возвращается True) используют строковую буферизацию. Другие текстовые файлы используют политику, описанную выше для двоичных файлов

Encoding – это имя кодировки, используемой для декодирования или кодирования файла. Это следует использовать только в текстовом режиме. Кодировка по умолчанию зависит от платформы (независимо от [locale.getpreferredencoding()](https://docs.python.org/3/library/locale.html#locale.getpreferredencoding) возврата), но может использоваться любая [кодировка текста,](https://docs.python.org/3/glossary.html#term-text-encoding) поддерживаемая Python.

error является необязательной строкой, которая определяет, как должны обрабатываться ошибки кодирования и декодирования - это нельзя использовать в двоичном режиме. Доступно множество стандартных обработчиков ошибок (перечисленных в разделе « [Обработчики ошибок»](https://docs.python.org/3/library/codecs.html#error-handlers) ), хотя любое зарегистрированное имя обработки ошибок [codecs.register\_error()](https://docs.python.org/3/library/codecs.html" \l "codecs.register_error" \o "codecs.register_error)также допустимо. Стандартные имена включают в себя:

* 'strict'вызвать [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError" \o "ValueError)исключение, если есть ошибка кодирования. Значение по умолчанию Noneимеет тот же эффект.
* 'ignore'игнорирует ошибки. Обратите внимание, что игнорирование ошибок кодирования может привести к потере данных.
* 'replace'вызывает вставку замещающего маркера (такого как '?') в случае искаженных данных.
* 'surrogateescape'любые некорректные байты будут представлены как кодовые точки в Частной области использования Юникода в диапазоне от U + DC80 до U + DCFF. Эти частные кодовые точки будут затем возвращаться в те же байты, когда surrogateescapeпри записи данных используется обработчик ошибок. Это полезно для обработки файлов в неизвестной кодировке.
* 'xmlcharrefreplace'поддерживается только при записи в файл. Символы, не поддерживаемые кодировкой, заменяются соответствующей ссылкой на символ XML &#nnn;.
* 'backslashreplace' заменяет искаженные данные на escape-последовательности Python с обратной косой чертой.
* 'namereplace'(также поддерживается только при записи) заменяет неподдерживаемые символы на \N{...}escape-последовательности.

newline контролирует, как работает [универсальный](https://docs.python.org/3/glossary.html#term-universal-newlines) режим [новой строки](https://docs.python.org/3/glossary.html#term-universal-newlines) (это относится только к текстовому режиму). Это может быть None, '', '\n', '\r', и '\r\n'. Это работает следующим образом:

* При чтении ввода из потока, если новая строка является Noneрежим универсальной новой строки включен. Строки на входе могут оканчиваться на '\n', '\r'или '\r\n', и они преобразуются в '\n'перед возвратом вызывающей стороне. Если это так, включается ''универсальный режим перевода строки, но окончания строк возвращаются вызывающей стороне без перевода. Если он имеет какие-либо другие допустимые значения, входные строки заканчиваются только данной строкой, а окончание строки возвращается вызывающей стороне без перевода.
* При записи выходных данных в потоке, если новая строка является None, любые '\n' символы , написанные переведены в сепаратор линии системы по умолчанию, [os.linesep](https://docs.python.org/3/library/os.html" \l "os.linesep" \o "os.linesep). Если перевод строки - ''или '\n', перевод не выполняется. Если символ новой строки является любым из других допустимых значений, любые '\n'написанные символы переводятся в данную строку.

Если closefd - это Falseи был задан дескриптор файла, а не имя файла, базовый дескриптор файла будет оставаться открытым при закрытии файла. Если задано имя файла, closefd должно быть True(по умолчанию), иначе возникнет ошибка.

Пользовательский open er может быть использован, передавая вызываемый как *open* er . Базовый дескриптор файла для объекта file затем получается путем вызова *open* er с помощью ( file , flags ). *open* er должен возвращать дескриптор открытого файла (передача по мере открытия приводит к функциональности, аналогичной передаче ).[os.open](https://docs.python.org/3/library/os.html" \l "os.open" \o "os.open)None

Readline(автоперевод)

Модуль определяет ряд функций для облегчения завершения и чтения / записи файлов история из интерпретатора Python. Этот модуль можно использовать напрямую или через модуль, который поддерживает заполнение идентификаторов Python в интерактивном режиме. Настройки, сделанные с помощью этого модуля, влияют на поведение как интерактивной подсказки переводчика, так и подсказок, предлагаемых встроенной функцией.[readline](https://docs.python.org/3/library/readline.html?highlight=readline#module-readline)[rlcompleter](https://docs.python.org/3/library/rlcompleter.html#module-rlcompleter)[input()](https://docs.python.org/3/library/functions.html#input)

Связывание клавиш Readline может быть настроено через файл инициализации, обычно .inputrcв вашем домашнем каталоге. См. [Файл](https://tiswww.cwru.edu/php/chet/readline/rluserman.html#SEC9) инициализации Readline в руководстве GNU Readline для получения информации о формате и допустимых конструкциях этого файла, а также о возможностях библиотеки Readline в целом.

Исходный файл

Следующие функции относятся к файлу инициализации и пользовательской конфигурации:

readline.**parse\_and\_bind**( *строка*)

Выполните строку инициализации, указанную в *строковом* аргументе. Это вызывает rl\_parse\_and\_bind()в базовой библиотеке.

readline.**read\_init\_file**( [ *имя файла*] )

Выполнить файл инициализации readline . Имя файла по умолчанию - это последнее использованное имя файла. Это вызывает rl\_read\_init\_file()в базовой библиотеке.

Строка буфера

Следующие функции работают с буфером строки:

readline.**get\_line\_buffer**( )

Возвращает текущее содержимое буфера строки ( rl\_line\_buffer в базовой библиотеке).

readline.**insert\_text**( *строка*)

Вставьте текст в буфер строки в позиции курсора. Это вызывает rl\_insert\_text()в базовой библиотеке, но игнорирует возвращаемое значение.

readline.**redisplay**( )

Измените то, что отображается на экране, чтобы отразить текущее содержимое буфера строки. Это вызывает rl\_redisplay()в базовой библиотеке.

Файл истории

Следующие функции работают с файлом истории:

readline.**read\_history\_file**( [ *имя файла*] )

Загрузите файл истории readline и добавьте его в список истории. Имя файла по умолчанию ~/.history. Это вызывает read\_history()в базовой библиотеке.

readline.**write\_history\_file**( [ *имя файла*] )

Сохраните список истории в файл истории чтения строки, перезаписывая любой существующий файл. Имя файла по умолчанию ~/.history. Это вызывает write\_history()в базовой библиотеке.

readline.**append\_history\_file**( *nelements*[ , *имя файла*] )

Append последних *nelements* элементов истории в файл. Имя файла по умолчанию ~/.history. Файл должен уже существовать. Это вызывает append\_history()в базовой библиотеке. Эта функция существует, только если Python был скомпилирован для версии библиотеки, которая ее поддерживает.

Список истории

Следующие функции работают с глобальным списком истории:

readline.**clear\_history**( )

Очистить текущую историю. Это вызывает clear\_history()в базовой библиотеке. Функция Python существует только в том случае, если Python был скомпилирован для версии библиотеки, которая ее поддерживает.

readline.**get\_current\_history\_length**( )

Вернуть количество предметов, которые в данный момент находятся в истории. (Это отличается от того [get\_history\_length()](https://docs.python.org/3/library/readline.html?highlight=readline" \l "readline.get_history_length" \o "readline.get_history_length), которое возвращает максимальное количество строк, которые будут записаны в файл истории.)

readline.**get\_history\_item**( *индекс*)

Вернуть текущее содержимое элемента истории по *индексу* . Предметный указатель является единичным. Это вызывает history\_get()в базовой библиотеке.

readline.**remove\_history\_item**( *поз*)

Удалить элемент истории, определенный его положением, из истории. Позиция начинается с нуля. Это вызывает remove\_history()в базовой библиотеке.

readline.**replace\_history\_item**( *поз* , *строка*)

Заменить элемент истории, заданный его положением, *строкой* . Позиция начинается с нуля. Это вызывает replace\_history\_entry() в базовой библиотеке.

readline.**add\_history**( *линия*)

Добавьте *строку* в буфер истории, как если бы это была последняя напечатанная строка. Это вызывает add\_history()в базовой библиотеке.

readline.**set\_auto\_history**( *включено*)

Включить или отключить автоматические вызовы add\_history()при чтении ввода через readline . *Включен* аргумент должен быть логическим значением, когда истинные, позволяет автоматически историю, и что , когда ложные, отключает авто истории.

*Новое в версии 3.6.*

**Детали реализации CPython:** Автоматическая история включена по умолчанию, и изменения в ней не сохраняются в течение нескольких сеансов.

Пусковые крючки

readline.**set\_startup\_hook**( [ *функция*] )

Установите или удалите функцию, вызванную rl\_startup\_hook обратным вызовом базовой библиотеки. Если указана *функция* , она будет использоваться в качестве новой функции подключения; если опущен или Noneлюбая установленная функция удаляется. Хук вызывается без аргументов непосредственно перед тем, как readline напечатает первое приглашение.

readline.**set\_pre\_input\_hook**( [ *функция*] )

Установите или удалите функцию, вызванную rl\_pre\_input\_hook обратным вызовом базовой библиотеки. Если указана *функция* , она будет использоваться в качестве новой функции подключения; если опущен или Noneлюбая установленная функция удаляется. Хук вызывается без аргументов после того, как напечатано первое приглашение, и непосредственно перед тем, как readline начнет читать вводимые символы. Эта функция существует, только если Python был скомпилирован для версии библиотеки, которая ее поддерживает.

Завершение

Следующие функции относятся к реализации пользовательской функции завершения слова. Обычно это происходит с помощью клавиши Tab и может предложить и автоматически завершить вводимое слово. По умолчанию Readline настроен для использования [rlcompleter](https://docs.python.org/3/library/rlcompleter.html" \l "module-rlcompleter" \o "rlcompleter: завершение идентификатора Python, подходит для библиотеки readline GNU.)для завершения идентификаторов Python для интерактивного интерпретатора. Если [readline](https://docs.python.org/3/library/readline.html?highlight=readline" \l "module-readline" \o "readline: поддержка GNU readline для Python.  (Unix))модуль должен использоваться с пользовательским завершителем, должен быть установлен другой набор разделителей слов.

readline.**set\_completer**( [ *функция*] )

Установите или удалите завершающую функцию. Если указана *функция* , она будет использоваться как новая завершающая функция; если опущен или Noneлюбая установленная завершающая функция удаляется. Функция завершившая называются , для *государства* в , , , ..., пока она не возвращает значение нестрокового. Он должен вернуть следующее возможное завершение, начиная с *текста* .function(text, state)012

Установленная завершающая функция вызывается *обратным* вызовом *entry\_func,* переданным rl\_completion\_matches()в базовую библиотеку. *Текст* строка происходит от первого параметра rl\_attempted\_completion\_functionобратного вызова базовой библиотеки.

readline.**get\_completer**( )

Получить функцию завершения, или Noneесли функция завершения не была установлена.

readline.**get\_completion\_type**( )

Получите тип завершения попытки. Это возвращает rl\_completion\_typeпеременную в базовой библиотеке как целое число.

readline.**get\_begidx**( )

readline.**get\_endidx**( )

Получить начальный или конечный индекс области завершения. Эти индексы являются *начальным* и *конечным* аргументами, передаваемыми rl\_attempted\_completion\_functionобратному вызову базовой библиотеки.

readline.**set\_completer\_delims**( *строка*)

readline.**get\_completer\_delims**( )

Установите или получите разделители слов для завершения. Они определяют начало слова, которое должно рассматриваться для завершения (область завершения). Эти функции обращаются к rl\_completer\_word\_break\_characters переменной в базовой библиотеке.

readline.**set\_completion\_display\_matches\_hook**( [ *функция*] )

Установите или удалите функцию отображения завершения. Если указана *функция* , она будет использоваться как функция отображения нового завершения; если опущен или Noneлюбая установленная функция отображения завершения удаляется. Это устанавливает или очищает rl\_completion\_display\_matches\_hookобратный вызов в базовой библиотеке. Функция отображения завершения вызывается так, как только каждый раз должны отображаться совпадения.function(substitution, [matches], longest\_match\_length)

На write документация не найдена.