Сообщить об ошибке.

# Модуль readline в Python, автодополнение табуляцией

study.skysmart.ru

РЕКЛАМА

**Учим детей программированию на основе реальных заданий**

Бесплатный вводный урок • Обучение от 7 до 18 лет • Разработчик сайтов • Python-разработчик • Разработчик игр

Узнать больше



## Автодополнение текста в командной строке табуляцией

Модуль readline может быть использован для улучшения интерактивных программ командной строки, чтобы упростить их использование. Он в основном используется для обеспечения завершения текста в командной строке или "автодополнения табуляцией". Настройки, сделанные с помощью этого модуля, влияют на поведение как интерактивной подсказки переводчика, так и подсказок, предлагаемых встроенной функцией input().

**Внимание**! Ссылка для тех, кто ищет метод файлового объекта построчного чтения из файла file.readline().

Примечание: так как модуль readline взаимодействует с содержимым консоли, печать отладочных сообщений затрудняет просмотр того, что происходит в примере кода.

## Содержание:



---

## Конфигурирование.

Существует два способа настройки базовой библиотеки readline, используя файл конфигурации или функцию parse_and_bind(). Опции конфигурации включают связывание клавиш для вызова завершения, режимы редактирования, например vi или emacs и многие другие значения. За подробностями обращайтесь к документации библиотеки readline GNU.

Самый простой способ включить завершение табуляции - через вызов функции модуля parse_and_bind(). Другие параметры могут быть установлены одновременно. В этом примере изменяются элементы управления редактированием для использования режима vi вместо значения по умолчанию emacs. Чтобы редактировать текущую строку ввода, нажмите ESC , затем используйте обычные клавиши навигации vi, такие как j , k , l и h .

---

```
import readline

readline.parse_and_bind('tab: complete')
readline.parse_and_bind('set editing-mode vi')

while True:
    line = input('Prompt ("stop" to quit): ')
    if line == 'stop':
        break
    print (f'ENTERED: "{line}"')
```

Та же самая конфигурация может быть сохранена как инструкции в файле, читаемом библиотекой за один вызов. Если myreadline.rc содержит:

```
tab: complete
set editing-mode vi
```

то файл можно прочитать с помощью функции модуля read_init_file():

```
import readline

re Вверх read_init_file('myreadline.rc')
```

```
while True:
    line = input('!("stop" to quit) Ввод текста: =>')
    if line == 'stop':
        break
    print (f'Отправлено: "{line}"')
```

## Автодополнение текста.

В качестве примера того, как построить завершение командной строки, можно взглянуть на программу, которая имеет встроенный набор возможных команд и использует завершение табуляции, когда пользователь вводит инструкции.

```
import readline

class SimpleCompleter():
    def __init__(self, options):
        self.options = sorted(options)
        return

    def complete(self, text, state):
        response = None
        if state == 0:
            # Создание списка соответствий.
            if text:
                self.matches = [s
                                for s in self.options
                                if s and s.startswith(text)]
            else:
                self.matches = self.options[:]
        # Вернуть элемент состояния из списка совпадений,
        # если их много.
        try:
            response = self.matches[state]
        except IndexError:
            response = None
        return response

def inputing():
    line = ''
    while line != 'stop':
        line = input('!("stop" to quit) Ввод текста: => ')
        print (f'Отправка: {line}')

# Регистрация класса 'SimpleCompleter'
readline.set_completer(SimpleCompleter(['start', 'stop', 'list', 'print']).complete)

# Регистрация клавиши `tab` для автодополнения
readline.parse_and_bind('tab: complete')

# Запрос текста
inputing()
```

Класс SimpleCompleter хранит список параметров, которые являются кандидатами на автозаполнение. Метод complete() предназначен для регистрации в функции set_completer() модуля readline в качестве источника дополнений. Аргументами являются строка text для завершения и значение state, указывающее, сколько раз функция была вызвана с одним и тем же текстом. Функция вызывается многократно, состояние увеличивается каждый раз. Класс SimpleCompleter должен вернуть строку, если есть кандидат для этого значения состояния, или None, если нет кандидатов. Реализация метода complete() ищет набор совпадений, когда состояние равно 0, а затем возвращает все совпадения кандидатов по одному при последующих вызовах.

Если дважды нажать клавишу TAB , список параметров будет напечатан.

```
# Дважды нажата клавиша 'TAB'
!("stop" to quit) Ввод текста: =>
# list   print  start  stop

# Введите 'li' и дважды нажмите 'TAB'
!("stop" to quit) Ввод текста: => list

# Вверх 'pr' и дважды нажмите 'TAB'
```

```
!("stop" to quit) Ввод текста: => print
# Отправка: print
```

## Доступ к буферу автодополнения текста.

Функции модуля readline можно использовать для манипулирования текстом входного буфера. В примере выполняются команды с вложенными опциями. Метод complete() должен посмотреть на положение завершения в буфере ввода, чтобы определить, является ли оно частью первого слова или более позднего слова. Если целью является первое слово, то в качестве кандидатов используются ключи словаря опций. Если это не первое слово, то первое слово используется для поиска кандидатов из словаря опций.

Будем обрабатывать три команды верхнего уровня, две из которых имеют подкоманды:

- list
  - files
  - directories
- print
  - byname
  - bysize
- stop

```
import readline

class BufferAwareCompleter():
    def __init__(self, options):
        self.options = options
        self.current_candidates = []
        return

    def complete(self, text, state):
        response = None
        if state == 0:
            # Создание списка соответствий.
            origline = readline.get_line_buffer()
            begin = readline.get_begidx()
            end = readline.get_endidx()
            being_completed = origline[begin:end]
            words = origline.split()

            if not words:
                self.current_candidates = sorted(self.options.keys())
            else:
                try:
                    if begin == 0:
                        # Первое слово
                        candidates = self.options.keys()
                    else:
                        # Последующие слова
                        first = words[0]
                        candidates = self.options[first]

                    if being_completed:
                        # параметры сопоставления с частью ввода
                        self.current_candidates = [ w for w in candidates
                                                    if w.startswith(being_completed) ]
                    else:
                        # соответствие пустой строке, используются все кандидаты
                        self.current_candidates = candidates
                except IndexError as err:
                    self.current_candidates = []
                except KeyError as err:
                    self.current_candidates = []
        try:
            response = self.current_candidates[state]
        except IndexError:
            response = None
        return response

def inputing():
```

Вверх

```
    line = ''
    while line != 'stop':
        line = input('!("stop" to quit) Ввод текста: => ')
        print (f'Отправка: {line}')

# Регистрация класса 'BufferAwareCompleter'
readline.set_completer(BufferAwareCompleter(
    {'list':['files', 'directories'],
     'print':['byname', 'bysize'],
     'stop':[],
    }).complete)

# Регистрация клавиши `tab` для автодополнения
readline.parse_and_bind('tab: complete')

# Запрос текста
inputing()
```

Если запустить программу то можно увидеть что-то подобное:

```
# Дважды нажата клавиша 'TAB'
!("stop" to quit) Ввод текста: =>
list   print  stop

# Введите 'li' и дважды нажмите 'TAB'
!("stop" to quit) Ввод текста: => list

# Дважды нажата клавиша 'TAB'
!("stop" to quit) Ввод текста: => list
directories  files

# Введите 'di' и дважды нажмите 'TAB'
!("stop" to quit) Ввод текста: => list directories
Отправка: list directories
```

## История пользовательского ввода.

Модуль readline автоматически отслеживает историю ввода. Есть два разных набора функций для работы с историей. Доступ к истории текущего сеанса можно получить с помощью функций модуля get_current_history_length() и get_history_item() . Эта же история может быть сохранена в файл для последующей загрузки с использованием write_history_file() и read_history_file() . По умолчанию вся история сохраняется, но максимальная длина файла может быть установлена с помощью set_history_length() . Длина -1 означает отсутствие ограничений.

```
import readline
import os

HISTORY_FILENAME = 'completer.hist'

def get_history_items():
    return [ readline.get_history_item(i)
             for i in range(1, readline.get_current_history_length() + 1)
             ]

class HistoryCompleter(object):

    def __init__(self):
        self.matches = []
        return

    def complete(self, text, state):
        response = None
        if state == 0:
            history_values = get_history_items()
            if text:
                self.matches = sorted(h
                                      for h in history_values
                                      if h and h.startswith(text))
            else:
                self.matches = []
        try:
            response = self.matches[state]
```

Вверх

```
        except IndexError:
            response = None
        return response

def inputing():
    if os.path.exists(HISTORY_FILENAME):
        readline.read_history_file(HISTORY_FILENAME)
    print(f'Максимальная длина файла истории: {readline.get_history_length()}')
    print(f'История запуска:{get_history_items()}')
    try:
        while True:
            line = input('!("stop" to quit) Ввод текста: => ')
            if line == 'stop':
                break
            if line:
                print(f'Добавление "{line}" в файл истории.')
    finally:
        print(f'Конец записи истории: {get_history_items()}')
        readline.write_history_file(HISTORY_FILENAME)

# Регистрация класса 'HistoryCompleter'
readline.set_completer(HistoryCompleter().complete)

# Регистрация клавиши `tab` для автодополнения
readline.parse_and_bind('tab: complete')

# Запрос текста
inputing()
```

Запустим программу:

```
Максимальная длина файла истории: -1
История запуска:[]
!("stop" to quit) Ввод текста: => print
Добавление "print" в файл истории.
!("stop" to quit) Ввод текста: => sort
Добавление "sort" в файл истории.

# Введите 'pr' и дважды нажмите 'TAB'
!("stop" to quit) Ввод текста: => print

# Введите 'so' и дважды нажмите 'TAB'
!("stop" to quit) Ввод текста: => sort

!("stop" to quit) Ввод текста: => stop
Конец записи истории: ['print', 'sort', 'stop']
```

Когда скрипт запускается во второй раз, вся история читается из файла.

```
Максимальная длина файла истории: -1
История запуска:['print', 'sort', 'stop']
# Введите 's' и дважды нажмите 'TAB'
!("stop" to quit) Ввод текста: => s
sort  stop
```

Функции для удаления отдельных элементов истории, а также очистки всей истории смотрите в разделе "Список истории".

## Hooks (Хуки).

Есть несколько доступных уловок/хуков для запуска действий как часть последовательности взаимодействия. Хук немедленно вызывается перед печатью подсказки, после приглашения перед чтением текста от пользователя.

```
import readline

def startup_hook():
    readline.insert_text('from startup_hook')

def pre_input_hook():
    readline.insert_text(' from pre_input_hook')
    ine.redisplay()
```

Вверх

```
readline.set_startup_hook(startup_hook)
readline.set_pre_input_hook(pre_input_hook)
readline.parse_and_bind('tab: complete')

line = ''
while line != 'stop':
    line = input('!("stop" to quit) Ввод текста: => ')
    print (f'Отправка: {line}')
```

Любой хук является потенциально хорошим местом для использования функцию модуля insert_text() для изменения входного буфера.

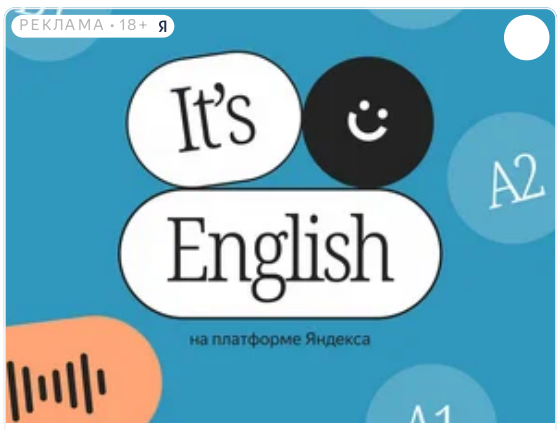
```
!("stop" to quit) Ввод текста: => from startup_hook from pre_input_hook
```

Если буфер изменен внутри хука предварительного ввода, вам нужно вызвать функцию модуля redisplay() для обновления экрана.

Содержание раздела:




ХОЧУ ПОМОЧЬ ПРОЕКТУ




practicum.yandex.ru

**Курсы Английского от Яндекс Практикума**

Яндекс бесплатно определит уровень языка и подберёт правильную программу обучения

Займёт 25 минут

Ответим на все вопросы

Бесплатно

LMS Практикум

Узнать больше


DOCS-Python.ru™, 2023 г. (Внимание! При копировании материала ссылка на источник обязательна) @docs_python_ru

Вверх