help()

A Gtk GUI that provides the REPL help()

Hamilton Python Users Group
Ian Stewart
8 August 2022

help()

REPL - Python interactive interpreter
Read: take user input.
Eval: evaluate the input.
Print: shows the output to the user.
Loop: repeat.

```
$ python
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more info
>>> help()
```

>>> help()
Help categories: Modules, Keywords, Symbols, Topics


```
ian@hp: ~/dev/python-help

File  Edit  View  Search  Terminal  Help

ian@hp:~/dev/python-help$ python
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.8's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.8/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> 
```

Help() utility - Recommends

Tutorial: https://docs.python.org/3.8/tutorial/

Categories from which to obtaining lists of help available:

- Keywords
- Symbols
- Topics
- Modules

To quit this help utility and return to the interpreter, just type "quit".

help("topics")

```
ian@hp: ~/dev/python-help
File  Edit  View  Search  Terminal  Help
>>> help("topics")

Here is a list of available topics.  Enter any topic name to get more help.

ASSERTION             DELETION             LOOPING              SHIFTING
ASSIGNMENT            DICTIONARIES         MAPPINGMETHODS       SLICINGS
ATTRIBUTEMETHODS      DICTIONARYLITERALS   MAPPINGS             SPECIALATTRIBUTES
ATTRIBUTES           DYNAMICFEATURES       METHODS              SPECIALIDENTIFIERS
AUGMENTEDASSIGNMENT   ELLIPSIS             MODULES              SPECIALMETHODS
BASICMETHODS          EXCEPTIONS           NAMESPACES           STRINGMETHODS
BINARY               EXECUTION             NONE                 STRINGS
BITWISE              EXPRESSIONS           NUMBERMETHODS        SUBSCRIPTS
BOOLEAN              FLOAT                 NUMBERS              TRACEBACKS
CALLABLEMETHODS       FORMATTING           OBJECTS              TRUTHVALUE
CALLS                FRAMEOBJECTS          OPERATORS            TUPLELITERALS
CLASSES              FRAMES                PACKAGES             TUPLES
CODEOBJECTS          FUNCTIONS             POWER                TYPEOBJECTS
COMPARISON           IDENTIFIERS           PRECEDENCE           TYPES
COMPLEX              IMPORTING             PRIVATENAMES         UNARY
CONDITIONAL          INTEGER              RETURNING            UNICODE
CONTEXTMANAGERS       LISTLITERALS         SCOPING
CONVERSIONS          LISTS                SEQUENCEMETHODS
DEBUGGING            LITERALS             SEQUENCES
```

help("keywords")

```
ian@hp: ~/dev/python-help                                    ⊖ ⊡ ⊗
File  Edit  View  Search  Terminal  Help
>>> help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not


>>>
```

help("symbols")

```
ian@hp: ~/dev/python-help

File  Edit  View  Search  Terminal  Help

>>> help("symbols")

Here is a list of the punctuation symbols which Python assigns special meaning
to. Enter any symbol to get more help.

!=                       +                       <=                      
"                        +=                      <>                      `
"""                              ,               ==                      b"
%                        -                       >                       b'
%=                       -=                      >=                      f"
&                        .                       >>                      f'
&=                       ...                     >>=                     j
'                        /                       @                       r"
'''                      //                      J                       r'
(                        //=                     [                       u"
)                        /=                      \                       u'
*                        :                       ]                       |
**                       <                       ^                       |=
**=                      <<                      ^=                      ~
*=                       <<=                     
                                                 -

>>>
```

help("modules") – Beginning section...

```
ian@hp: ~/dev/python-help

File  Edit  View  Search  Terminal  Help

>>> help("modules")

Please wait a moment while I gather a list of all available modules...

2dd510b5c3364608e57a__mypyc anyascii              httplib2              re
Appearance               anyio                    httptools             readline
AptUrl                   appdirs                  idna                  redshift_gtk
AtspiStateTracker        apport                   imaplib               regex
AutoHide                 apport_python_hook  imghdr                reportlab
AutoShow                 apt                      imp                   reprlib
ClickSimulator           apt_inst                 importlib             requests
CommandNotFound          apt_pkg                  importlib_metadata    requests_unixsocket
Config                   aptdaemon                importlib_resources resampy
ConfigUtils              aptsources               inflect               resource
Crypto                   argparse                 inspect               retrying
Cython                   array                    io                    rlcompleter
DBusUtils                ast                      ipaddr                rmagic
DistUpgrade              asynchat                 ipaddress             roman
Exceptions               asyncio                  ipykernel             runpy
GlobalKeyListener        asyncore                 ipykernel_launcher  samba
HardwareSensorTracker atexit                       ipython_genutils      scanext
```

help("modules") – Ending section...


```
ian@hp: ~/dev/python-help
File  Edit  View  Search  Terminal  Help
_testcapi              gruut_lang_de        pylab                xapp
_testimportmultiple    gruut_lang_en        pymacaroons          xattr
_testinternalcapi      gruut_lang_es        pynndescent          xdg
_testmultiphase        gruut_lang_fr        pyparsing            xdrlib
_thread                gruut_lang_it        pypinyin             xkit
_threading_local       gruut_lang_nl        pypredict            xml
_tkinter               gruut_lang_pt        pyrsistent           xmlrpc
_tracemalloc           gruut_lang_ru        pysbd                xxlimited
_uuid                  gruut_lang_sv        python-help-gui      xxsubtype
_version               gzip                 python-help-gui-full yaml
_warnings              h11                  python-help-gui_no_module youtube_dl
_weakref               hashlib              python-help-gui_original zipapp
_weakrefset            heapq                pytz                 zipfile
_xxsubinterpreters     hexdump              pytz_deprecation_shim zipimport
_xxtestfuzz            hmac                 pyworld              zipp
_yaml                  hpmudext             pyximport            zlib
abc                    html                 queue                zmq
aifc                   html5lib             quopri
antigravity            http                 random

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose name or summary contain the string "spam".
```

help()

Examples of obtaining Help from the four categories...

```
>>> help("OPERATORS")  # topic from 73 topics

>>> help("+")  # symbol from 59 symbols

>>> help("break")  # keyword from 35 keywords

>>> help("sys")  # module from 500+ modules
```

Example of help("OPERATORS") or help ("+")

```
ian@hp: ~

File  Edit  View  Search  Terminal  Help

Operator precedence
*******************

The following table summarizes the operator precedence in Python, from
lowest precedence (least binding) to highest precedence (most
binding).  Operators in the same box have the same precedence.  Unless
the syntax is explicitly given, operators are binary.  Operators in
the same box group left to right (except for exponentiation, which
groups from right to left).

Note that comparisons, membership tests, and identity tests, all have
the same precedence and have a left-to-right chaining feature as
described in the Comparisons section.

+----------------------------------------------------+----------------------------
----------+
| Operator                                           | Description
          |
|====================================================|============================
==========|
| ":="                                               | Assignment expression
          |
          |
+----------------------------------------------------+----------------------------
:
```

Objective: Write a GUI help() using GTK3
Programming steps:
- Get >>> help() on each of the four categories.
- Output each category with its 4 column lists to a temp file.
- Read back the temp file to a buffer.
- Filter out pre and post amble text.
- Extract the data from the four columns into a single list.
- Sort the data alphabetically
- Store the data in a "help" dictionary.

- Launch a *GTK.Window*
- Add a Scrolled *TextView* widget to display the help information
- Add a Scrolled *TreeView* widget to display the categories.
- Use a *TreeStore* to hold the "help" dictionary.
- Link the *TreeStore* to the *TreeView*.

Writing a category to a temp file using the contextlib module .

```python
import contextlib

def write_help(func, out_file):
    """

    Write the output of help() to a file.
    Usage example: write_help(int, 'integer_info.txt')
    Requires: contextlib
    """

    with open(out_file, 'w') as f:
        with contextlib.redirect_stdout(f):
            help(func)


def create_help_files():
    """

    Use python help to output files that contain the help topics.
    """

    write_help("keywords", 'keywords.txt')
    write_help("topics", 'topics.txt')
    write_help("symbols", 'symbols.txt')
    write_help("modules", 'modules.txt')
```

Avoiding writing to a temp file by using the io.StringIO module.

```python
import contextlib
from io import StringIO

def get_help(func):
    """

    Write the output of help() to a text buffer and return as text string.
    Usage example: get_help("float")
    Requires: contextlib and io.StringIO
    """

    output = StringIO()
    with contextlib.redirect_stdout(output):
        help(func)
    contents = output.getvalue()
    output.close()
    return contents
```

Passing the Help() category to obtain a list.

```python
def get_help_list(data):
    """

    Python help has categories of topics, symbols and keywords.
    These topics are displayed in four columns
    Convert the category into a list and sort alphabetically
    """

    new_data = data.replace("\n", " ")   # change newlines to spaces.
    position = new_data.find("help.")   # Find end of heading
    new_data = new_data[position+5:]   # Start after heading
    data_list = new_data.split(" ")

    # Clear the list of empty fields using pop()
    data_len_index = len(data_list) - 1
    for index, item in enumerate(reversed(data_list)):
        if item == "":
            data_list.pop(data_len_index - index)

    data_list.sort()
    return data_list
```

help()

Having obtained all the recognised help words and symbols, then they may be called. E.g.

help("+")
help("OPERATORS")
help("int")
help("sys")

… and using contextlib and io.StringIO, the information is placed in the text.buffer of *TextView* and is displayed.

Gtk3 code. Importing and launching Window..

```python
import gi
try:
    gi.require_version("Gtk", "3.0")
except ValueError as e:
    print(e)
    sys.exit("Unable to run {} program. Exiting...".format(sys.argv[0]))
from gi.repository import Gtk, Gdk
```

```python
# Start up the window
win = Window()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Gtk3 code. *Gtk.Window* being set up...

```python
class Window(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self)

        # Add the title to the window
        version = sys.version.split(" ")[0]
        self.set_title("Help for Python version {}".format(version))
        # Set default window size
        self.set_default_size(1400, 600)

        self.grid = Gtk.Grid()
        self.add(self.grid)

        self.create_textview()
        self.setup_treeview()
        self.set_style()
        self.textbuffer.set_text(WELCOME)
```

Gtk3 code. In a *Gtk.ScrolledWindow* add *Gtk.TextView*...

```python
def create_textview(self):
    scrolled_window = Gtk.ScrolledWindow()
    scrolled_window.set_hexpand(True)
    scrolled_window.set_vexpand(True)
    scrolled_window.set_border_width(10)
    self.grid.attach(scrolled_window, 0, 0, 7, 1)

    self.textview = Gtk.TextView()
    self.textview.set_name("textview")
    self.textbuffer = self.textview.get_buffer()
    self.textbuffer.set_text("")
    scrolled_window.add(self.textview)
```

Gtk3 code. In a *Gtk.TreeStore* add the help() dictionary to store.
Create *Gtk.TreeView* and get it to use the store...

```python
def setup_treeview(self):
    # create a TreeStore with one string column to use as the model
    store = Gtk.TreeStore(str)

    # From help dictionary add the categories and items to the store
    for category, items in help_dict.items():
        category_key = store.append(None, [category])
        for item in items:
            store.append(category_key,[item])

    # create the TreeView using treestore data
    self.treeview = Gtk.TreeView()
    self.treeview.set_model(store)
```

Gtk3 code. Continue setting up *Gtk.TreeView*

```python
# Set unique name so css can be applied to widget
self.treeview.set_name("treeview")

# Preference for self.treeview. Adjust via contants at start of program
self.treeview.set_activate_on_single_click(SINGLE_CLICK)

treeview_column = Gtk.TreeViewColumn(COLUMN_HEADING)
self.treeview.append_column(treeview_column)

cell = Gtk.CellRendererText()

treeview_column.pack_start(cell, True)
treeview_column.add_attribute(cell, 'text', 0)

# call-back on the self.treeview
self.treeview.connect ("row-activated", self.cb_on_row_activate,)
```

Gtk3 code. Place *Gtk.TreeView* in a *GTK.ScrolledWindow*

```python
# Create a window that can be scrolled
scrolled_window = Gtk.ScrolledWindow(hexpand=True, vexpand=True)
scrolled_window.set_policy(Gtk.PolicyType.NEVER, Gtk.PolicyType.AUTOMATIC)
scrolled_window.set_border_width(10)

scrolled_window.add(self.treeview)

self.grid.attach(scrolled_window, 8, 0, 1, 1)
```
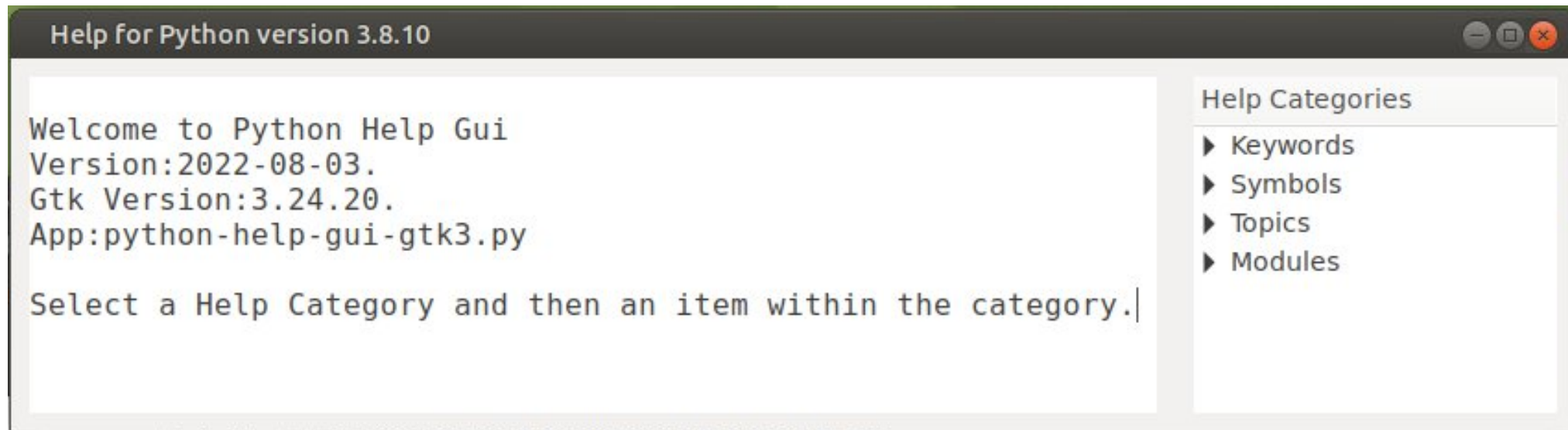
Gtk3 code. Call-back when "click" on *Gtk.TreeView*

```python
def cb_on_row_activate (self, treeview, path, column,):
    ' Call back when click on a row in the treeview. Select station'
    model = treeview.get_model()
    iter  = model.get_iter (path)
    #print(model[iter][0]) # Whatever item is clicked on

    # Don't want the categories only the items which have two fields
    pointer_list = path.to_string().split(":")
    if len(pointer_list) == 2:
        self.item_selected = model[iter][0]
        # Update the title to reflect selected item
        version = sys.version.split(" ")[0]
        self.set_title("Help for Python version {} - Selection: {}"
                .format(version, self.item_selected))
        # Retrieve the help information and display
        info = get_help(self.item_selected)
        self.textbuffer.set_text(info)
```

Gtk3 code. Apply CSS for fonts in *Gtk.TextView* and *Gtk.TreeView*...

```python
def set_style(self):
    """ Loads custom CSS for textview and treeview"""
    style_provider = Gtk.CssProvider()
    style_provider.load_from_data(b"""
#textview {
    font: 16px "Monospace";
    /*color: #000000;  Black will conflict with dark themed display */
}
#treeview {
    font: 14px Sans;
    /* border-width: 10px; Around the column header */
}
""")
    Gtk.StyleContext.add_provider_for_screen(Gdk.Screen.get_default(),
            style_provider,
            Gtk.STYLE_PROVIDER_PRIORITY_APPLICATION)
```

Run the program...

Help for Python version 3.8.10

Welcome to Python Help Gui
Version:2022-08-03.
Gtk Version:3.24.20.
App:python-help-gui-gtk3.py

Select a Help Category and then an item within the category.

Help Categories

▶ Keywords
▶ Symbols
▶ Topics
▶ Modules

Run the program. In help category "Topics" clicked on "OPERATORS"

Darkened View: In help category "Topics" clicked on "OPERATORS"

```
Operator precedence
*******************

The following table summarizes the operator precedence in Python, from
lowest precedence (least binding) to highest precedence (most
binding).  Operators in the same box have the same precedence.  Unless
the syntax is explicitly given, operators are binary.  Operators in
the same box group left to right (except for exponentiation, which
groups from right to left).

Note that comparisons, membership tests, and identity tests, all have
the same precedence and have a left-to-right chaining feature as
described in the Comparisons section.

+---------------------------------------------------+----------------------------------------------+
| Operator                                          | Description                                  |
|===================================================|==============================================|
| ":="                                              | Assignment expression                        |
+---------------------------------------------------+----------------------------------------------+
| "lambda"                                          | Lambda expression                            |
+---------------------------------------------------+----------------------------------------------+
| "if" – "else"                                     | Conditional expression                       |
+---------------------------------------------------+----------------------------------------------+
| "or"                                              | Boolean OR                                   |
+---------------------------------------------------+----------------------------------------------+
| "and"                                             | Boolean AND                                  |
+---------------------------------------------------+----------------------------------------------+
| "not" "x"                                         | Boolean NOT                                  |
+---------------------------------------------------+----------------------------------------------+
| "in", "not in", "is", "is not", "<", "<=", ">",   | Comparisons, including membership            |
| ">=", "!=", "=="                                  | tests and identity tests                     |
+---------------------------------------------------+----------------------------------------------+
| "|"                                               | Bitwise OR                                   |
```

Help Categories

- NUMBERMETHODS
- NUMBERS
- OBJECTS
- **OPERATORS**
- PACKAGES
- POWER
- PRECEDENCE
- PRIVATENAMES
- RETURNING
- SCOPING
- SEQUENCEMETHODS
- SEQUENCES
- SHIFTING
- SLICINGS
- SPECIALATTRIBUTES
- SPECIALIDENTIFIERS
- SPECIALMETHODS
- STRINGMETHODS
- STRINGS
- SUBSCRIPTS
- TRACEBACKS
- TRUTHVALUE
- TUPLELITERALS
- TUPLES
- TYPEOBJECTS
- TYPES
- UNARY
- UNICODE
- ▸ Modules

help()

Demo Program.

help()

Python Observations

Get rid of the empty fields in the list (1/2)...

```
>>> a_str = "abc\nd  e"

>>> a_str
'abc\nd  e'

>>> print(a_str)
abc
d  e

>>> a_str = a_str.replace("\n", " ")

>>> print(a_str)
abc d  e

>>> a_str
'a b c  d   e'

>>> a_list = a_str.split(" ")

>>> a_list
['a', 'b', 'c', '', 'd', '', '','e']
```

Get rid of the empty fields in the list (2/2)...

```
>>> DATA_LEN_INDEX = len(a_list) - 1
>>> for index, item in enumerate(reversed(a_list)):
...     print(index, DATA_LEN_INDEX, item)
...     if item == "":
...         a_list.pop(DATA_LEN_INDEX - index)
...
0 7 e
1 7
''
2 7
''
3 7 d
4 7
''
5 7 c
6 7 b
7 7 a

>>> a_list
['a', 'b', 'c', 'd', 'e']
```

Fails when length is not held at initial value...

```
>>>> a_str = "a b c  d    e"
>>> a_list = a_str.split(" ")
>>> a_list
['a', 'b', 'c', '', 'd', '', '', '', 'e']
>>> for index, item in enumerate(reversed(a_list)):
...     print(len(a_list), index, item)
...     if item == "":
...         a_list.pop((len(a_list) - 1) - index)
...
9 0 e
9 1
''
8 2
''
7 3
''
6 4
'b'
5 5
'e'
4 6 d
4 7 c
4 8 a
>>> a_list
['a', 'c', 'd', '']
>>>
```

Iterate with list.remove() until ValueError...

```
>>> a_str = "a b c  d    e"
>>> a_list = a_str.split(" ")
>>> a_list
['a', 'b', 'c', '', 'd', '', '', '', 'e']

>>> try:
...     while True:
...         a_list.remove("")
... except ValueError as e:
...     print(e)
...
list.remove(x): x not in list

>>> a_list
['a', 'b', 'c', 'd', 'e']



>>> # Same as remove:
>>> while True:
...     x = a_list.index("")
...     a_list.pop(x)
```

How a list in a dictionary is preserved as a list...

```
>>> a = [1,2,3,4]
>>> a
[1, 2, 3, 4]

>>> d = {}

>>> d["a-list"]=a

>>> d
{'a-list': [1, 2, 3, 4]}

>>> a
[1, 2, 3, 4]

>>> a.pop()
4

>>> a
[1, 2, 3]

>>> d
{'a-list': [1, 2, 3]}
```

List is part of dictionary...

```
>>> a_str = "a b c d e"
>>> a_list = a_str.split(" ")
>>> a_list
['a', 'b', 'c', 'd', 'e']

>>> a_dict = {}
>>> a_dict["a_list_key"] = a_list

>>> a_dict
{'a_list_key': ['a', 'b', 'c', 'd', 'e']}

>>> a_list.reverse()

>>> a_list
['e', 'd', 'c', 'b', 'a']

>>> a_dict
{'a_list_key': ['e', 'd', 'c', 'b', 'a']}
>>>
```