25.5. IDLE

Source code: Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the `tkinter` GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit window. They currently have the same top menu as Editor windows but a different default title and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)

New File
: Create a new file editing window.

Open…
: Open an existing file with an Open dialog.

Recent Files
: Open a list of recent files. Click one to open it.

Open Module…
: Open an existing module (searches sys.path).

Class Browser

Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.

Path Browser
: Show sys.path directories, modules, functions, classes and methods in a tree structure.

Save
: Save the current window to the associated file, if there is one. Windows that have been changed since being opened or last saved have a * before and after the window title. If there is no associated file, do Save As instead.

Save As…
: Save the current window with a Save As dialog. The file saved becomes the new associated file for the window.

Save Copy As…
: Save the current window to different file without changing the associated file.

Print Window
: Print the current window to the default printer.

Close
: Close the current window (ask to save if unsaved).

Exit
: Close all windows and quit IDLE (ask to save unsaved windows).

25.5.1.2. Edit menu (Shell and Editor)

Undo
: Undo the last change to the current window. A maximum of 1000 changes may be undone.

Redo
: Redo the last undone change to the current window.

Cut
: Copy selection into the system-wide clipboard; then delete the selection.

Copy
: Copy selection into the system-wide clipboard.

Paste
: Insert contents of the system-wide clipboard into the current window.

The clipboard functions are also available in context menus.

Select All
: Select the entire contents of the current window.

Find…

Open a search dialog with many options

Find Again
: Repeat the last search, if there is one.

Find Selection
: Search for the currently selected string, if there is one.

Find in Files…
: Open a file search dialog. Put results in a new output window.

Replace…
: Open a search-and-replace dialog.

Go to Line
: Move cursor to the line number requested and make that line visible.

Show Completions
: Open a scrollable list allowing selection of keywords and attributes. See Completions in the Tips sections below.

Expand Word
: Expand a prefix you have typed to match a full word in the same window; repeat to get a different expansion.

Show call tip
: After an unclosed parenthesis for a function, open a small window with function parameter hints.

Show surrounding parens
: Highlight the surrounding parenthesis.

25.5.1.3. Format menu (Editor window only)

Indent Region
: Shift selected lines right by the indent width (default 4 spaces).

Dedent Region
: Shift selected lines left by the indent width (default 4 spaces).

Comment Out Region
: Insert ## in front of selected lines.

Uncomment Region
: Remove leading # or ## from selected lines.

Tabify Region
: Turn *leading* stretches of spaces into tabs. (Note: We recommend using 4 space blocks to indent Python code.)

Untabify Region
: Turn *all* tabs into the correct number of spaces.

Toggle Tabs
: Open a dialog to switch between indenting with spaces and tabs.

New Indent Width
: Open a dialog to change indent width. The accepted default by the Python community is 4 spaces.

Format Paragraph
: Reformat the current blank-line-delimited paragraph in comment block or multiline string or selected line in a string. All lines in the paragraph will be formatted to less than N columns, where N defaults to 72.

Strip trailing whitespace
: Remove trailing space and other whitespace characters after the last non-whitespace character of a line by applying str.rstrip to each line, including lines within multiline strings.

25.5.1.4. Run menu (Editor window only)

Python Shell
: Open or wake up the Python Shell window.

Check Module
: Check the syntax of the module currently open in the Editor window. If the module has not been saved IDLE will either prompt the user to save or autosave, as selected in the General tab of the Idle Settings dialog. If there is a syntax error, the approximate location is indicated in the Editor window.

Run Module
: Do Check Module (above). If no error, restart the shell to clean the environment, then execute the module. Output is displayed in the Shell window. Note that output requires use of `print` or `write`. When execution is complete, the Shell retains focus and displays a prompt. At this point, one may interactively explore the result of execution. This is similar to executing a file with `python -i file` at a command line.

25.5.1.5. Shell menu (Shell window only)

View Last Restart
: Scroll the shell window to the last Shell restart.

Restart Shell
: Restart the shell to clean the environment.

Interrupt Execution
: Stop a running program.

25.5.1.6. Debug menu (Shell window only)

Go to File/Line
: Look on the current line. with the cursor, and the line above for a filename and line number. If found, open the file if not already open, and show the line. Use this to view source lines referenced in an exception traceback and lines found by Find in Files. Also available in the context menu of the Shell window and Output windows.

Debugger (toggle)
: When activated, code entered in the Shell or run from an Editor will run under the debugger. In the Editor, breakpoints can be set with the context menu. This feature is still incomplete and somewhat experimental.

Stack Viewer
: Show the stack traceback of the last exception in a tree widget, with access to locals and globals.

Auto-open Stack Viewer
: Toggle automatically opening the stack viewer on an unhandled exception.

25.5.1.7. Options menu (Shell and Editor)

Configure IDLE
: Open a configuration dialog and change preferences for the following: fonts, indentation, keybindings, text color themes, startup windows and size, additional help sources, and extensions (see below). On OS X, open the configuration dialog by selecting Preferences in the application menu. To use a new built-in color theme (IDLE Dark) with older IDLEs, save it as a new custom theme.

: Non-default user settings are saved in a .idlerc directory in the user's home directory. Problems caused by bad user configuration files are solved by editing or deleting one or more of the files in .idlerc.

Code Context (toggle)(Editor Window only)
: Open a pane at the top of the edit window which shows the block context of the code which has scrolled above the top of the window. Clicking a line in this pane exposes that line at the top of the editor.

25.5.1.8. Window menu (Shell and Editor)

Zoom Height
: Toggles the window between normal size and maximum height. The initial size defaults to 40 lines by 80 chars unless changed on the General tab of the Configure IDLE dialog.

The rest of this menu lists the names of all open windows; select one to bring it to the foreground (deiconifying it if necessary).

25.5.1.9. Help menu (Shell and Editor)

About IDLE
: Display version, copyright, license, credits, and more.

IDLE Help
: Display a help file for IDLE detailing the menu options, basic editing and navigation, and other tips.

Python Docs
: Access local Python documentation, if installed, or start a web browser and open docs.python.org showing the latest Python documentation.

Turtle Demo
: Run the turtledemo module with example python code and turtle drawings.

Additional help sources may be added here with the Configure IDLE dialog under the General tab.

25.5.1.10. Context Menus

Open a context menu by right-clicking in a window (Control-click on OS X). Context menus have the standard clipboard functions also on the Edit menu.

Cut
: Copy selection into the system-wide clipboard; then delete the selection.

Copy
: Copy selection into the system-wide clipboard.

Paste
: Insert contents of the system-wide clipboard into the current window.

Editor windows also have breakpoint functions. Lines with a breakpoint set are specially marked. Breakpoints only have an effect when running under the debugger. Breakpoints for a file are saved in the user’s .idlerc directory.

Set Breakpoint
: Set a breakpoint on the current line.

Clear Breakpoint
: Clear the breakpoint on that line.

Shell and Output windows have the following.

Go to file/line

Same as in Debug menu.

25.5.2. Editing and navigation

In this section, ‘C’ refers to the `Control` key on Windows and Unix and the `Command` key on Mac OSX.

- `Backspace` deletes to the left; `Del` deletes to the right
- `C-Backspace` delete word left; `C-Del` delete word to the right
- Arrow keys and `Page Up`/`Page Down` to move around
- `C-LeftArrow` and `C-RightArrow` moves by words
- `Home`/`End` go to begin/end of line
- `C-Home`/`C-End` go to begin/end of file
- Some useful Emacs bindings are inherited from Tcl/Tk:
  - `C-a` beginning of line
  - `C-e` end of line
  - `C-k` kill line (but doesn’t put it in clipboard)
  - `C-l` center window around the insertion point
  - `C-b` go backward one character without deleting (usually you can also use the cursor key for this)
  - `C-f` go forward one character without deleting (usually you can also use the cursor key for this)
  - `C-p` go up one line (usually you can also use the cursor key for this)
  - `C-d` delete next character

Standard keybindings (like `C-c` to copy and `C-v` to paste) may work. Keybindings are selected in the Configure IDLE dialog.

25.5.2.1. Automatic indentation

After a block-opening statement, the next line is indented by 4 spaces (in the Python Shell window by one tab). After certain keywords (break, return etc.) the next line is dedented. In leading indentation, `Backspace` deletes up to 4 spaces if they are there. `Tab` inserts spaces (in the Python Shell window one tab), number depends on Indent width. Currently, tabs are restricted to four spaces due to Tcl/Tk limitations.

See also the indent/dedent region commands in the edit menu.

25.5.2.2. Completions

Completions are supplied for functions, classes, and attributes of classes, both built-in and user-defined. Completions are also provided for filenames.

The AutoCompleteWindow (ACW) will open after a predefined delay (default is two seconds) after a ‘.’ or (in a string) an os.sep is typed. If after one of those characters (plus zero or more other characters) a tab is typed the ACW will open immediately if a possible continuation is found.

If there is only one possible completion for the characters entered, a `Tab` will supply that completion without opening the ACW.

‘Show Completions’ will force open a completions window, by default the `C-space` will open a completions window. In an empty string, this will contain the files in the current directory. On a blank line, it will contain the built-in and user-defined functions and classes in the current namespaces, plus any modules imported. If some characters have been entered, the ACW will attempt to be more specific.

If a string of characters is typed, the ACW selection will jump to the entry most closely matching those characters. Entering a `tab` will cause the longest non-ambiguous match to be entered in the Editor window or Shell. Two `tab` in a row will supply the current ACW selection, as will return or a double click. Cursor keys, Page Up/Down, mouse selection, and the scroll wheel all operate on the ACW.

“Hidden” attributes can be accessed by typing the beginning of hidden name after a ‘.’, e.g. ‘_’. This allows access to modules with `__all__` set, or to class-private attributes.

Completions and the ‘Expand Word’ facility can save a lot of typing!

Completions are currently limited to those in the namespaces. Names in an Editor window which are not via `__main__` and `sys.modules` will not be found. Run the module once with your imports to correct this situation. Note that IDLE itself places quite a few modules in sys.modules, so much can be found by default, e.g. the re module.

If you don’t like the ACW popping up unbidden, simply make the delay longer or disable the extension.

25.5.2.3. Calltips

A calltip is shown when one types `(` after the name of an *accessible* function. A name expression may include dots and subscripts. A calltip remains until it is clicked, the cursor is moved out of the argument area, or `)` is typed. When the cursor is in the argument part of a definition, the menu or shortcut display a calltip.

A calltip consists of the function signature and the first line of the docstring. For builtins without an accessible signature, the calltip consists of all lines up the fifth line or the first blank line. These details may change.

The set of *accessible* functions depends on what modules have been imported into the user process, including those imported by Idle itself, and what definitions have been run, all since the last restart.

For example, restart the Shell and enter `itertools.count(`. A calltip appears because Idle imports itertools into the user process for its own use. (This could change.) Enter `turtle.write(` and nothing appears. Idle does not import turtle. The menu or shortcut do nothing either. Enter `import turtle` and then `turtle.write(` will work.

In an editor, import statements have no effect until one runs the file. One might want to run a file after writing the import statements at the top, or immediately run an existing file before editing.

25.5.2.4. Python Shell window

- `C-c` interrupts executing command
- `C-d` sends end-of-file; closes window if typed at a `>>>` prompt
- `Alt-/` (Expand word) is also useful to reduce typing

  Command history

  - `Alt-p` retrieves previous command matching what you have typed. On OS X use `C-p`.
  - `Alt-n` retrieves next. On OS X use `C-n`.
  - `Return` while on any previous command retrieves that command

25.5.2.5. Text colors

Idle defaults to black on white text, but colors text with special meanings. For the shell, these are shell output, shell error, user output, and user error. For Python code, at the shell prompt or in an editor, these are keywords, builtin class and function names, names following `class` and `def`, strings, and comments. For any text window, these are the cursor (when present), found text (when possible), and selected text.

Text coloring is done in the background, so uncolorized text is occasionally visible. To change the color scheme, use the Configure IDLE dialog Highlighting tab. The marking of debugger breakpoint lines in the editor and text in popups and dialogs is not user-configurable.

25.5.3. Startup and code execution

Upon startup with the `-s` option, IDLE will execute the file referenced by the environment variables `IDLESTARTUP` or `PYTHONSTARTUP`. IDLE first checks for `IDLESTARTUP`; if `IDLESTARTUP` is present the file referenced is run. If `IDLESTARTUP` is not present, IDLE checks for `PYTHONSTARTUP`. Files referenced by these environment variables are convenient places to store functions that are used frequently from the IDLE shell, or for executing import statements to import common modules.

In addition, `Tk` also loads a startup file if it is present. Note that the Tk file is loaded unconditionally. This additional file is `.Idle.py` and is looked for in the user's home directory. Statements in this file will be executed in the Tk namespace, so this file is not useful for importing functions to be used from IDLE's Python shell.

25.5.3.1. Command line usage

```
idle.py [-c command] [-d] [-e] [-h] [-i] [-r file] [-s] [-t title] [-

-c command  run command in the shell window
-d          enable debugger and open shell window
-e          open editor window
-h          print help message with legal combinations and exit
-i          open shell window
-r file     run file in shell window
-s          run $IDLESTARTUP or $PYTHONSTARTUP first, in shell window
-t title    set title of shell window
-           run stdin in shell (- must be last option before args)
```

If there are arguments:

- If `-`, `-c`, or `r` is used, all arguments are placed in `sys.argv[1:...]` and `sys.argv[0]` is set to `''`, `'-c'`, or `'-r'`. No editor window is opened, even if that is the default set in the Options dialog.
- Otherwise, arguments are files opened for editing and `sys.argv` reflects the arguments passed to IDLE itself.

25.5.3.2. Startup failure

IDLE uses a socket to communicate between the IDLE GUI process and the user code execution process. A connection must be established whenever the Shell starts or restarts. (The latter is indicated by a divider line that says 'RESTART'). If the user process fails to connect to the GUI process, it displays a `Tk` error box with a 'cannot connect' message that directs the user here. It then exits.

A common cause of failure is a user-written file with the same name as a standard library module, such as *random.py* and *tkinter.py*. When such a file is located in the same directory as a file that is about to be run, IDLE cannot import the stdlib file. The current fix is to rename the user file.

Though less common than in the past, an antivirus or firewall program may stop the connection. If the program cannot be taught to allow the connection, then it must be turned off for IDLE to work. It is safe to allow this internal connection because no data is visible on external ports. A similar problem is a network mis-configuration that blocks connections.

Python installation issues occasionally stop IDLE: multiple versions can clash, or a single installation might need admin access. If one undo the clash, or cannot or does not want to run as admin, it might be easiest to completely remove Python and start over.

A zombie pythonw.exe process could be a problem. On Windows, use Task Manager to detect and stop one. Sometimes a restart initiated by a program crash or Keyboard Interrupt (control-C) may fail to connect. Dismissing the error box or Restart Shell on the Shell menu may fix a temporary problem.

When IDLE first starts, it attempts to read user configuration files in ~/.idlerc/ (~ is one's home directory). If there is a problem, an error message should be displayed. Leaving aside random disk glitches, this can be prevented by never editing the files by hand, using the configuration dialog, under Options, instead Options. Once it happens, the solution may be to delete one or more of the configuration files.

If IDLE quits with no message, and it was not started from a console, try starting from a console (`python -m idlelib`) and see if a message appears.

25.5.3.3. IDLE-console differences

With rare exceptions, the result of executing Python code with IDLE is intended to be the same as executing the same code in a console window. However, the different interface and operation occasionally affect visible results. For instance, `sys.modules` starts with more entries.

IDLE also replaces `sys.stdin`, `sys.stdout`, and `sys.stderr` with objects that get input from and send output to the Shell window. When Shell has the focus, it controls the keyboard and screen. This is normally transparent, but functions that directly access the keyboard and screen will not work. If `sys` is reset with `importlib.reload(sys)`, IDLE's changes are lost and things like `input`, `raw_input`, and `print` will not work correctly.

With IDLE's Shell, one enters, edits, and recalls complete statements. Some consoles only work with a single physical line at a time. IDLE uses `exec` to run each statement. As a result, `'__builtins__'` is always defined for each statement.

25.5.3.4. Developing tkinter applications

IDLE is intentionally different from standard Python in order to facilitate development of tkinter programs. Enter `import tkinter as tk; root = tk.Tk()` in standard Python and nothing appears. Enter the same in IDLE and a tk window appears. In standard Python, one must also enter `root.update()` to see the window. IDLE does the equivalent in the background, about 20 times a second, which is about every 50 milleseconds. Next enter `b = tk.Button(root, text='button'); b.pack()`. Again, nothing visibly changes in standard Python until one enters `root.update()`.

Most tkinter programs run `root.mainloop()`, which usually does not return until the tk app is destroyed. If the program is run with `python -i` or from an IDLE editor, a `>>>` shell prompt does not appear until `mainloop()` returns, at which time there is nothing left to interact with.

When running a tkinter program from an IDLE editor, one can comment out the mainloop call. One then gets a shell prompt immediately and can interact with the live application. One just has to remember to re-enable the mainloop call when running in standard Python.

25.5.3.5. Running without a subprocess

By default, IDLE executes user code in a separate subprocess via a socket, which uses the internal loopback interface. This connection is not externally visible and no data is sent to or received from the Internet. If firewall software complains anyway, you can ignore it.

If the attempt to make the socket connection fails, Idle will notify you. Such failures are sometimes transient, but if persistent, the problem may be either a firewall blocking the connection or misconfiguration of a particular system. Until the problem is fixed, one can run Idle with the -n command line switch.

If IDLE is started with the -n command line switch it will run in a single process and will not create the subprocess which runs the RPC Python execution server. This can be useful if Python cannot create the subprocess or the RPC socket interface on your platform. However, in this mode user code is not isolated from IDLE itself. Also, the environment is not restarted when Run/Run Module (F5) is selected. If your code has been modified, you must reload() the affected modules and re-import any specific items (e.g. from foo import baz) if the changes are to take effect. For these reasons, it is preferable to run IDLE with the default subprocess if at all possible.

Deprecated since version 3.4.

25.5.4. Help and preferences

25.5.4.1. Additional help sources

IDLE includes a help menu entry called “Python Docs” that will open the extensive sources of help, including tutorials, available at docs.python.org. Selected URLs can be added or removed from the help menu at any time using the Configure IDLE dialog. See the IDLE help option in the help menu of IDLE for more information.

25.5.4.2. Setting preferences

The font preferences, highlighting, keys, and general preferences can be changed via Configure IDLE on the Option menu. Keys can be user defined; IDLE ships with four built-in key sets. In addition, a user can create a custom key set in the Configure IDLE dialog under the keys tab.

25.5.4.3. Extensions

IDLE contains an extension facility. Preferences for extensions can be changed with the Extensions tab of the preferences dialog. See the beginning of config-extensions.def in the idlelib directory for further information. The only current default extension is zzdummy, an example also used for testing.