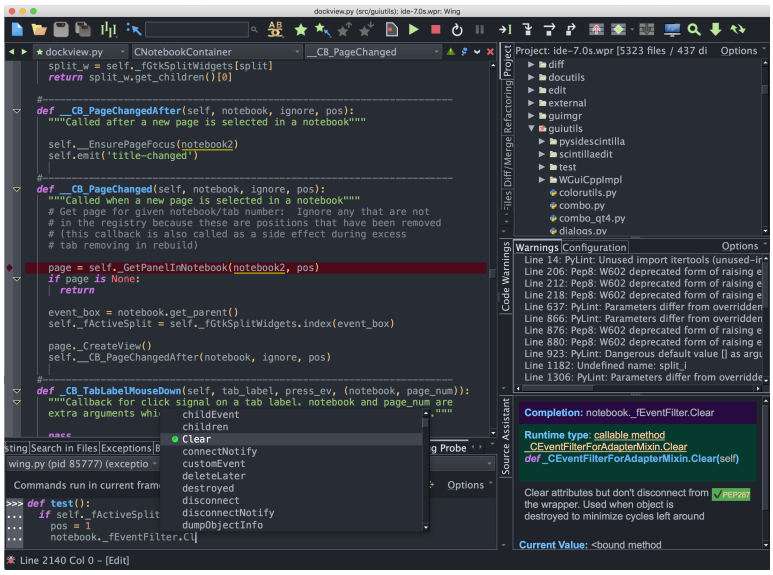
Wing Pro Quick Start Guide



This is a minimalist guide for getting started quickly with Wing Pro. For a more in-depth introduction, try the Tutorial.

Wing Pro is a light-weight yet powerful integrated development environment that was designed from the ground up for Python. Once you're up to speed with Wing you should find that:

- Wing speeds up your development of new code
- Wing makes it easier to understand and work with existing code
- Wing reveals errors earlier in the development process
- Wing makes it easier to find and fix bugs

- Wing adapts to your needs and style

This is made possible through deep code analysis (both static and runtime), a focus on interactive development in the live runtime, high-level editing operations and refactoring, continuous early error detection, support for test-driven development, powerful always-on debugger, seamless support for remote and containerized development, and extreme configurability.

Let's get started with Wing Pro!

Install Python

If you don't already have Python on your system, install it now. Two good options are:

- Obtain the standard Python distribution from python.org
- Use Anaconda for seamless access to many third party Python libraries. See Anaconda package lists for a list of the available libraries.

See Supported Python Versions for other options.

You may need to restart Wing after installing Python, so that it can recognize the new installation.

Set up a Project

After Wing is running, select **New Project** from the **Project** menu to create a new project. This dialog lets you choose or create the source directory and choose or create the Python environment you want to use for your new project. When creating a new source directory, you can optionally pull a revision control repository into it. Wing Pro can also create and install packages into new virtualenv, Poetry env, pipenv, Anaconda env, and Docker container environments.

If you choose **Create Blank Project**, you can configure your project later with the following steps:

1. Use **Add Existing Directory** in the **Project** menu to add your sources to the project. It's best to constrain this to the directories you are actively working with and let Wing find the libraries you use through the **Python Path**.
2. Use **Project Properties** in the **Project** menu to set **Python Executable** to the **python.exe** or other interpreter executable you want to use with your project. If Python is not on the **PATH**, set this to the the full path that is in **sys.executable** in the desired Python installation.
3. If your code alters **sys.path** or loads modules in a non-standard way then you may need to set **Python Path** in **Project Properties** so that Wing can find your modules for auto-completion, refactoring, debugging, testing, and other features.
4. You may want to right-click on your main entry point in the **Project** tool and select **Set As Main Entry Point** so that debugging always starts there.
5. Use **Save Project As** in the **Project** menu to save your project to disk.
6. Use the **Packages** tool in the **Tools** menu to manage Python packages in your selected Python environment.

See Project-Wide Properties and Per-File Properties for a description of all available properties.

Notice that Wing also offers other project types in the **New Project** dialog, including one for connecting to a remote host via SSH, running with Docker or other containers, accessing a Vagrant instance, working with Windows Subsystem for Linux, and a project type for each of the frameworks, tools, and libraries listed in How-Tos.

Wing may consume significant CPU time when it first analyzes your code base. Progress is indicated in the lower left of the IDE window. Once this is done, the results are cached across sessions and Wing should run with a snappy and responsive interface. See Source Code Analysis to learn how Wing's source analysis system works.

Basic Configuration

You are now ready to start working with code, but may want to make a few configuration changes first:

Display Colors - The **User Interface > Display Mode** preference selects whether Wing runs with a light or dark display style. This is also available in the high-level configuration menu in the top right of Wing's window. The specific styles used are selected with the **User Interface > Light Theme** and **User Interface > Dark Theme** preferences. The editor's colors can be configured separately with the **User Interface > Light Editor** and **User Interface > Dark Editor** preference.

Key Bindings - Wing can emulate VI/Vim, Visual Studio, Emacs, Eclipse, XCode, MATLAB, and Brief editors, as selected from **Keyboard Personality** in the **Edit** menu or with the **User Interface > Keyboard > Personality** preference.

Tab Key - The default tab key action depends on the selected keyboard personality and in some cases file type, context, and whether or not there is a selection in the editor. This can be changed from the **User Interface > Keyboard > Tab Key Action** preference.

Completion Keys - By default, the auto-completer uses the **Tab** key for completion, but other keys can be added using the **Editor > Auto-completion > Completion Keys** preference.

There are many other options in **Preferences**.

Navigating Code

Wing Pro provides a number of different ways to navigate the structure of your code, and several methods for quickly finding symbols or files by name:

Source Index menus at the top of the editor provide quick access to other parts of a source file.

Goto-definition is available from the **Source** menu, and by right-clicking on symbols in the editor, **Python Shell** and **Debug Console**. Use the forward/back history buttons at the top left of the editor to return from the point of definition.

Find Points of Use in Wing Pro's **Source** menu shows where the current symbol is being used. This distinguishes between separate but like-named symbols.

Find Symbol in the **Source** menu in Wing Pro and Wing Personal jumps to a symbol defined in the current file when you type a fragment of its name.

Find Symbol in Project in the **Source** menu in Wing Pro works the same way but searches all files in the project.

Open From Project in the **File** menu in Wing Pro and Wing Personal provides a similar interface for quickly opening project files.

See Navigating Source for details on the above.

Source Browser in the **Tools** menu in Wing Pro and Wing Personal provides module or class oriented display of the structure of your code. Details

Source Assistant in the **Tools** menu shows detailed information about symbols selected in the editor, auto-completer, **Source Browser**, **Python Shell**, **Project**, and other tools. Details

Searching

Wing Pro provides several different interfaces for searching your code. Which you use depends on what you want to search and how you prefer to interact with the search and replace functionality:

Toolbar search is a quick way to search the current file. Details

Search in the **Tools** menu shows the **Search** tool, which provides incremental text, wildcard, and regular expression search and replace in selections and the current file or documentation page. Details

Mini-search in Wing Pro and Wing Personal provides powerful keyboard-driven search and replace. The key bindings listed in the **Mini-search** area of the **Edit** menu display the search entry area at the bottom of the window. Details

Search in Files in the **Tools** menu in Wing Pro and Wing Personal shows the **Search in Files** tool, which provides wildcard and regular expression search and replace in filtered sets of files, directories, named file sets, and within the project and documentation. Details.

Editing Code

Wing Pro's editor is designed to speed up the process of writing and modifying Python code, and to reduce the incidence of coding errors. Its features include:

Auto-completion in Wing's editor, **Python Shell** and **Debug Console** speeds up typing and reduces coding errors. The auto-completer uses **Tab** by default for completion, but this can be changed in the **Editor > Auto-completion > Completion Keys** preference. This feature is disabled by default in Wing 101. Details

Auto-indent in Wing Pro and Wing Personal matches the file's existing indentation. When multiple lines are pasted, they are re-indented according to context. A single **Undo** reverts an unwanted indentation change. A selected range of code may be re-indented as a block using **Indentation** in the **Source** menu or the indentation toolbar group. The **Indentation** tool may be used to convert a whole file's indentation style. Details

Auto-Editing in Wing Pro implements a range of operations such as auto-entering closing parentheses, brackets, braces, and quotes. Among other things, Wing also auto-enters invocation arguments, manages new blocks with the **:** key, and corrects out-of-order typing. Auto-editing operations can be enabled and disabled in the **Editor > Auto-editing** preferences group. The default set includes those operations that don't affect finger memory. The others are well worth learning. Details

Refactoring operations in Wing Pro, accessed from the **Refactoring** menu, implement automated renaming and moving of symbols, creating functions or methods out of existing code, and introducing variables much more quickly than by manually editing code. Details

Multiple Selections can be made with **Multiple Selections** in the **Edit** menu, the multiple selections toolbar item, and by pressing **Ctrl+Alt** (or **Command+Option** on macOS) while making a selection with the mouse. Once multiple selections have been made, edits made will be applied to all the selections at once. Details

Code Warnings are shown in Wing Pro for syntax errors, indentation problems, unreachable code, use of undefined variables and attributes, unresolvable imports, and some other problems. External checkers like ruff, flake8, mypy, pep8, and pylint may also be configured as sources for the code warnings. Warnings are shown on the editor with details shown in a tooltip when the mouse hovers over the warning indicator. Warnings can be navigated from the warnings menu in the top right of the editor and managed from the **Code Warnings** tool. Details

Snippets in Wing Pro are included in Wing's auto-completer as a quick way to enter commonly repeated patterns for coding standards, documentation, testing, and so forth. Data entry for snippet arguments is inline in the editor. Use the **Tab** key to move between the fields. Edit or add snippets in the **Snippets** tool. Details

Turbo Completion in Wing Pro is an optional auto-completion mode for Python, made possible by Wing's powerful source analysis engine. When the **Editor > Auto-completion > Python Turbo Mode** preference is enabled, Wing turns every non-symbol key into a completion key in contexts where a new symbol name is not being typed. Details

Quick Selection operations in the **Edit > Select** menu allow selecting whole statements, blocks, or scopes before copying, editing, or searching through them. Details

Debugging Code

Wing's debugger is a powerful tool for finding and fixing bugs, understanding unfamiliar code, and writing new code interactively. You can launch code from the **Debug** menu or toolbar, from the Python Shell, or from outside of the IDE either on the same machine or on another host. Wing also supports working with code running on containers like those provided by Docker.

Breakpoints can be set by clicking on the breakpoint margin to the left of the editor. Stepping operations are in the **Debug** menu and toolbar.

The **Stack Data** tool is used to inspect or change program data. Right-click on items to display the item as an array or in textual form. Hovering the mouse over a symbol in the editor shows the value for that

symbol in a tooltip, if available on the active debug stack. Pressing **Shift-Space** shows tooltips for all symbols visible in the editor.

Debug process I/O is shown in the **Debug I/O** tool, or optionally in an external console.

Other debugger features include:

Interactive Debugging is supported by Wing Pro's **Debug Console**, which provides a Python prompt that executes code in the current debug stack frame. When the debugger is paused, Wing also uses the live runtime state to populate the auto-completer in the editor, **Source Assistant**, goto-definition, and other tools. Details

Conditional Breakpoints can be used in Wing Pro to isolate and understand complex bugs by stopping before they occur. Using a conditional breakpoint to isolate a broken case and the **Debug Console** to design a fix is far more productive than relaunching code repeatedly. Details

Move Program Counter is supported by Wing Pro, by right-clicking in the editor and selecting **Move Program Counter Here**. Because of how Python is implemented, this feature works only in the innermost stack frame and it does not work when the debugger is stopped on an exception.

Watching Values in Wing Pro by right-clicking on the editor or any of the data views tracks values over time by symbolic name or object reference in the **Watch** tool. Expressions can be also be watched. Details

Launch Configurations in the **Project** menu in Wing Pro and Wing Personal define different runtime environments for debugging, executing, and unit testing your code. Details.

Named Entry Points in the **Debug** menu in Wing Pro and Wing Personal provide a way to launch the same file with different debug environments. Details

Other Features

Wing Pro includes a number of other features designed to make Python coding easier and more productive:

Python Shell -- Wing's **Python Shell** lets you try out code in an independent sandbox process. To enable debugging, click the bug icon in the top right of the **Python Shell**. In Wing Pro and Wing Personal, the shell provides auto-completion, goto-definition, and is integrated with the **Source Assistant**. Details

Unit Testing in Wing Pro's **Testing** tool works with unittest, doctest, pytest, nose, and Django unit tests. You can run tests suites, view the results, and debug tests. Details

Version Control in Wing Pro supports revision control with Mercurial, Git, Subversion, Perforce, Bazaar, and CVS. Wing auto-detects which systems are used in your project and shows the appropriate additional menus and tools in the **Tools** menu. Details.

Difference and Merge in the **Source** menu can be used to compare and merge files and directories on disk, files open in the IDE, an unsaved buffer with disk, and a working copy with its revision control repository. Details

Remote Development is easy in Wing Pro, to remote hosts, virtual machines, or containers that are accessible via SSH. In this model of remote development, Wing works seamlessly and securely with files stored entirely on the remote host. Use the **Remote Hosts** item in the **Project** menu to configure a remote host, then set the **Python Executable** in **Project Properties** to that remote host, and use **Add Existing Directory** in the **Project** menu to add your remote directories to the project. Wing can edit, debug, test, search, inspect, and manage files, run the **Python Shell**, and execute **OS Commands** on the remote host in the same way as it does when working locally. Details

Package management is available for virtualenv, Poetry, pipenv, and Anaconda environments, using the **Packages** tool in the **Tools** menu. This can be used to install, remove, and manage the packages that are installed into your project's Python environment. Details.

Containers like those provided by Docker are also supported. In this development model, files are stored locally but code is run inside a containerized environment. Details **OS Commands** in the **Tools** menu in Wing Pro and Wing Personal's displays the **OS Commands** tool, which execute external tools for build, code generation, and other purposes. Details.

Preferences in the **Edit** menu (or **Wing Pro** menu on macOS) gives you control of the overall layout and color of the IDE, among many other options. Right click on tool and editor tabs for layout options, or drag tabs to move them or create new splits. Right-click on the toolbar to configure which tools are visible or to add your own. See Customization for details.

Perspectives in Wing Pro and Wing Personal let you save named tool panel layouts. Details.

Other Features like bookmarks, code folding, keyboard macros are also available, and you can extend Wing by writing Python scripts.

Further Reading

As you work with Wing Pro on your own software development projects, the following resources may be useful:

- Wing Support Website which includes a Q&A support forum, mailing lists, documentation, links to social media, and other information for Wing users.
- Wing Reference Manual which documents all the features in detail.
- How-Tos with instructions for using Wing with third party frameworks, applications, and tool, like Django, Jupyter, matplotlib, Autodesk Maya, Raspberry Pi, pygame, and many others.
- A collection of Wing Tips, available on our website and by weekly email subscription, provides additional tips and tricks for using Wing productively.

Thanks for using Wing Pro!