Wing Tutorial

This tutorial introduces Wing 101 by taking you through its feature set with a small coding example.

If you are new to programming, you may want to check out the book *Python Programming Fundamentals* and accompanying screen casts, which use Wing 101 to teach programming with Python.

Our *How-Tos* show how to use Wing with 3rd party web development frameworks, GUI toolkits, scientific data visualization tools, Python-based modeling, rendering & compositing systems, and other Python frameworks and toolkits.

To get started, press the Next Page icon in the toolbar immediately above this page.

The screenshots in this tutorial were made with Wing Pro and may contain some tools that are not available in Wing 101. These can be ignored and will not be discussed in the text that follows.

**Tutorial: Getting Started**

To get started using this tutorial, you will need to:

1. **Install Python and Wing**

   If you don’t already have them on your system, install Python and Wing. For detailed instructions, see Installing Wing.

2. **Start Wing**

   Wing can be started from a menu, desktop, or tray icon or using the command line executable. For detailed instructions, see Running Wing. If you don’t have a license, you can obtain a 30-day trial the first time you start Wing.

3. **Switch to the Integrated Tutorial**

   Once Wing is running, you should switch to using the Tutorial listed in Wing’s Help menu because it contains links directly into the IDE’s functionality. This includes step (4) below.

4. **Copy the Tutorial Directory**

   Copy the entire tutorial directory out of the Install Directory listed in Wing’s About box to another location on disk. You can do this manually or use the following link, which will prompt you to select the target directory: Copy Tutorial Now
To get to the next page in the tutorial, use the Next Page icon in the toolbar immediately above this page:

**Tutorial: Getting Around Wing**

Let's start with some basics that will help you get around Wing while working with this tutorial.

Wing's user interface is divided into an editor area and two toolboxes separated by draggable dividers. Try pressing F1 and F2 now to show or hide the two toolboxes. Also try Shift-F2 to maximize the editor area temporarily, hiding both tool areas and toolbar until Shift-F2 is pressed again.
Tool and editor tabs can be dragged to rearrange the user interface, optionally creating a new split. Right click on the tabs for a menu of additional options, such as adding or removing splits or to move the toolbox from right to left. The number of splits shown by default in toolboxes will vary according to the size of your display.

Notice that you can click on an already-active tool tab to minimize that tool area. Click again on any tab to restore the toolbox to its previous size.

See the Use Interface Layout documentation page for details.

**Context Menus**

In general, right-clicking provides a menu for interacting with or configuring a part of the user interface. On some systems you may need to configure your track pad to allow right-clicking, or use a keyboard modifier to emulate a right mouse click.

**Splitting the Editor Area**

Splitting your editor area makes it easier to get around this tutorial. To do this now, right-click on the editor tab area and select Split Side by Side.

By default, the editor shows all open files in all splits, making it easy to work on different parts of a file simultaneously. This can be changed by unchecking Show All Files in All Splits in the right-click context menu on the editor tabs.

**Configuring the Keyboard**
Use the Edit > Keyboard Personality menu or User Interface > Keyboard > Personality preference to tell Wing to emulate another editor, such as Visual Studio, VI/Vim, Emacs, Eclipse, XCode, MATLAB, or Brief.

The Configure Tab Key item in the Edit > Keyboard Personality menu or the User Interface > Keyboard > Tab Key Action preference can be used to select among available behaviors for the Tab key. The default is to match the selected Keyboard Personality.

When the Keyboard Personality is set to Wing, Tab acts differently according to context. For example, if lines are selected, repeated presses of Tab moves the lines among syntactically valid indent positions. And, when the caret is at the end of a line, pressing Tab adds one indent level.

See the Keyboard Personalities documentation section for details.

Other Configuration Options

Wing's cross-platform GUI adjusts to the OS on which you are running it, except on Linux where it cannot use the system-provided UI. You can set the colors used in the editor with the User Interface > Color Palette preference. To apply this palette also to the rest of the UI, outside of editors, enable the Use Color Palette Throughout the UI preference.

To set a dark background display style, select One Dark, Monikai, Black Background, or Solarized - Dark as the Color Palette.
The User Interface > Fonts > Display Font/Size and User Interface > Fonts > Editor Font/Size preferences select the fonts used for the user interface and editor.

The size and type of tools used in the toolbar at the top of Wing’s window can be changed by right-clicking on one of the enabled tools.

For more information on adjusting the user interface to your needs, see the Customization documentation.

**Tutorial: Check your Python Integration**

Before starting with some code, let's make sure that Wing has succeeded in finding your Python installation. Bring up the Python Shell tool now from the Tools menu. If all goes well, it should start up Python and show you the Python command prompt like this:
If this is not working, or the wrong version of Python is being used, you can point Wing in the right direction with Python Executable in the Python Configuration accessed from the Edit menu.

An easy way to determine the value to use for Python Executable is to start the Python you wish to use with Wing and type the following at Python's >>> prompt:

```python
import sys
sys.executable
```

You will need to Restart Shell from Options in the Python Shell tool after altering Python Executable.

Once the shell works, copy/paste or drag and drop these lines of Python code into it:

```python
for i in range(0, 10):
    print('*' * i)
```

This should print a triangle as follows:

```
    *
   **
  ***
 ****
*****
******
*******
********
*********
```

Notice that the shell removes common leading white space when blocks of code are copied into it. This is useful when trying out code from source files.

Now type something into the shell, such as:
You can create as many instances of the Python Shell tool as you wish. Each one runs in its own process space that is kept totally separate from Wing and any debug processes.

**Tutorial: Setting Python Path**

Python uses a search path referred to as the Python Path to find modules that are imported into code with the `import` statement. Most code only imports modules that are already on the default path, for example modules in the Python standard library, or modules installed into Python by pip, conda, or some other package manager.

However, in some cases code will depend on a different path provided either by setting the environment variable `PYTHONPATH` before starting Python, or by modifying `sys.path` at runtime before importing modules.

If the Python Path is changed by one of these methods, you may also need to tell Wing about this change. This is done with Python Path in the Configure Python dialog, accessed from the Edit menu:

For this tutorial, you need to add the `subdir` sub-directory of your `tutorials` directory to Python Path, as shown above. This directory contains a module used as part of the first coding example.
Note that the full path to the directory `subdir` is used. This is strongly recommended because it avoids potential problems finding source code during debugging, if the starting directory is ambiguous or changes over time.

The configuration used here is for illustrative purposes only. You could run the example code without altering the `Python Path` by moving the `path_example.py` file to the same location as the example scripts, or by placing it into your Python installation’s `site-packages` directory, which is in the default Python Path.

**Startup Environment**

Wing uses the environment it is started with as the default environment for your Python code. As a result, if `PYTHONPATH` is set when you start Wing, it will also be used with your code. If this inherited path matches the needs of your code, then you don’t need to set `Python Path` in Wing.

**Virtualenv**

If you are using `virtualenv` to set up your Python environment, you don’t need to set `Python Path`. Instead, set `Python Executable`, as described in Check Your Python Integration, to the Python executable set up by `virtualenv`. This causes Wing to pick up the correct path and other environment needed to run code in the virtual environment.

**Tutorial: Introduction to the Editor**

Now we’re ready to get started with some coding. Open up the file `example1.py` within Wing by selecting `Open` from the `File` menu.

Scroll down to the bottom of `example1.py` and enter the following code:

```python
news = ReadPythonNews(GetItemCount())
```

Press enter a few times. Note that Wing auto-indents the subsequent lines and adds red error indicators under them shortly after you stop typing. This indicates that there is a syntax error in your code:

```python
# Enter code according to the tutorial here:

news = ReadPythonNews(GetItemCount())
```
Once you correct the line and complete it by typing the final ), the error indicators will be removed. You should now have this complete line of code in your file:

```python
news = ReadPythonNews(GetItemCount())
```

Then enter the following additional lines of code:

```python
PrintAsText(news)
PromptToContinue()
PrintAsHTML(news)
```

At this point you have a complete program that can be run in the debugger.

**Tutorial: Debugging**

The `example1.py` program you have just created connects to `python.org` via HTTP, reads and parses the Python-related news feed in RDF format, and then prints the most recent five items as text and HTML. Don't worry if you are working offline. The script has canned data it will use when it cannot connect to `python.org`.

To start debugging, set a breakpoint on the line that reads `return 5` in the `GetItemCount` function. This can be done by clicking on the line and selecting the Break toolbar item, or by clicking on the left-most margin to the left of the line. The breakpoint should appear as a filled red circle:
Next start the debugger with `Debug` in the toolbar or `Start/Continue` in the `Debug` menu.

Wing will run to the breakpoint and stop, placing a red indicator on the line. Notice that the toolbar changes to include additional debug tools, as shown below:

Your display may vary depending on the size of your screen, or if you have altered the toolbar’s configuration. Wing displays tooltips explaining what the items do when you hover the mouse over them.

Now you can inspect the program state at that point with the `Stack Data` tool and by going up and down the stack with `Up Stack` and `Down Stack` in the toolbar or from the `Debug` menu. The stack can also be viewed as a list using the `Call Stack` tool:
Notice that the debug status indicator in the lower left of Wing’s main window changes color depending on the state of the debug process. Hover the mouse over the indicator to see detailed status in a tooltip:

Next, try stepping out to the enclosing call to `ReadPythonNews`. In this particular context, you can achieve this in a single click with the `Step Out` in the toolbar or Debug menu item. Two clicks on `Step Over` also work. `ReadPythonNews` is a good function to step through in order to try out the basic debugger features described above.

Try stepping or running to a breakpoint on the last line of this function, which reads `return news[:count]`. In this context, right-clicking on `news` under locals in Stack Data allows viewing the value in textual form or as an array. The latter loads data incrementally for only the visible portion of the value, which is useful with numpy arrays, pandas DataFrames, sqlite query results, and other larger data sets.

Data can also be viewed in tooltips on the editor by hovering the mouse over a value. Try this with `count` to see the value 5.

Finally, try `Step Out` to reach the RETURN event in `ReadPythonNews`. Notice that hovering the mouse over `return` in the editor displays the value that is being returned from the function call. Similarly, `<return value>` is added to the locals shown in the Stack Data tool.
6.1. Tutorial: Debug I/O

Before continuing any further in the debugger, bring up the Debug I/O tool so you can watch the subsequent output from the program. This is also where keyboard input takes place in debug code that requests it.

Once you step over the line `PrintAsText(news)` you should see output similar to the following:

```
For code that reads from stdin, uses input(), or calls Python 2.x's raw_input(), the Debug I/O tool is where you would type input to your program. Try this now by stepping over the PromptToContinue call with Step Over in the toolbar. You will see the prompt "Press Enter to Continue" appear in the Debug I/O tool and the debugger will not complete the Step Over operation until you press Enter while keyboard focus is in the Debug I/O tool.

You can also configure Wing to use an external console from the Options menu in the Debug I/O tool. This is useful for programs that requires a more complete console implementation to run correctly, for example those that use the curses module.

See the Debug Process I/O documentation page for details.
```

6.2. Tutorial: Debug Process Exception Reporting

Wing's debugger reports any exceptions that would be printed when the code outside of the debugger.

Try this out by continuing execution of the debug process with the Debug toolbar icon or Start / Continue in the Debug menu. Wing will stop on an incorrect line of code in PrintAsHTML and report the problem in the Exceptions tool:
The Exceptions tool highlights the current stack frame as you move up and down the stack. You can click on frames to navigate the exception backtrace, showing the source code for each frame.

Whenever you are stopped on an exception, the debugger status indicator in the lower left of Wing's main window turns red.

After reaching an exception in the debugger, you can correct your code, stop the debugger with the Stop icon in the toolbar, and then start debugging again.

### 6.3. Tutorial: Debugging from the Python Shell

In addition to launching code to debug from Wing's menu bar and Debug menu, it is also possible to debug code that is entered into the Python Shell.

Enable this now by clicking on the bug icon in the top right of the Python Shell. Once this is done, the status message at the top of the Python Shell will change to include Commands will be debugged and an extra margin is shown in which you can set breakpoints. Wing will reach those breakpoints, as well as any breakpoints in editors for code that is invoked. Any exceptions will be reported in the debugger.

Let's try this out. First stop any running debug process with the Stop icon in the toolbar. Then paste the following into the Python Shell and press Enter so that you are returned to the >>> prompt:
def test_function():
    x = 10
    print(x)
    x += 5
    y = 20
    print(x+y)

Next place a breakpoint on the line that reads `print(x)` by clicking in the breakpoint margin to the left of the prompt on that line.

Then type this into the Python Shell and press Enter:

test_function()

Wing should reach the breakpoint on `print(x)`.

You can now work with the debugger in the same way that you would if you had launched code from the toolbar or Debug menu. Try stepping and viewing the values of `x` and `y` as they change, either in the Stack Data tool or by hovering the mouse over the variable names.

Take a look at the stack in the Call Stack or Stack Data tool to see how stack frames that occur within the Python Shell are listed. You can move up and down the stack just as you would if your stack frames were in an editor.

Notice that if you step off the end of the call, you will return to the shell prompt. If you press the Stop item in the toolbar or select Stop Debugging from the Debug menu, Wing will complete execution of the code without debug and return you to the `>>>` prompt. Note that the code is still executed to completion in this case because there is no way to simply abandon a number of stack frames in the Python interpreter.

See the Debugging Code in the Python Shell documentation page for details.

**Tutorial: Indentation Features**

Since indentation is syntactically significant in Python, Wing provides a number of features to make working with indentation easier.

**Auto-Indentation**

By now you will have noticed that Wing auto-indents lines as you type, according to context. This can be disabled with the Editor > Indentation > Auto-Indent preference.

Wing also adjusts the indentation of blocks of code that are pasted into the editor. If the indentation change is not what you wanted, a single Undo removes the indentation adjustment, if there was one.

See the Auto-indent documentation page for details.

**Block Indentation**

In Wing's default keyboard personality, the Tab key is defined to indent the current line or blocks of lines, rather than entering a tab character (which can be done with Ctrl-T). As noted earlier, the User Interface > Keyboard > Tab Key Action preference can be used to customize how the tab key behaves. See The Tab Key for details.

One or more selected lines can be increased or reduced in indentation, or adjusted to match indentation according to context, from the Indentation toolbar group:
Repeated presses of the **Match Indent** tool will move the selected lines among the possible correct indent levels for that context.

These indentation features are also available in the **Source** menu, where their key bindings are listed.

**Tutorial: Other Editor Features**

There are a number of other editor features that are worth knowing about:

**Goto-Line**

Navigate quickly to a numbered source line with the **Goto Line** item in the **Edit** menu, or with the key binding displayed there. In some keyboard personalities, the line number is typed into the data entry area that appears at the bottom of the window. Press **Enter** to complete the action.

**Selecting Code**

Wing supports character, line, and block mode selection with **Selection Mode** in the **Edit** menu.

In Python code, the **Select sub-menu** in the **Edit** menu can be used to easily select and traverse logical blocks of code. The **Select More** and **Select Less** operations are particularly useful when preparing to type over or copy/paste ranges of text. Try these out now on `urllib` in `ReadPythonNews` in `example1.py`. Each repeated press of **Ctrl-Up** will select more code in logical units. Press **Ctrl-Down** to select less code.

The other operations in the **Select sub-menu** can be used for selecting and moving forward or backward over whole statements, blocks, or scopes. If you plan to use these and your selected **User Interface > Keyboard > Personality** preference does not support them, then you will want to define key bindings for them using the **User Interface > Keyboard > Custom Key Bindings** preference. The command names are `select-x`, `next-x`, and `previous-x` where `x` is either `statement`, `block`, or `scope`.

See the **Selecting Text** documentation section for details.

**Code Warnings**

As you probably noticed while working through the tutorial, Wing flags some types of incorrect code by drawing an underline under the code. This is done for syntax errors, indentation errors, and code that can't be reached. Hovering the mouse cursor over an indicator on the editor displays details for that warning or error in a tooltip.

**Block Commenting**

Lines of code can be commented out or un-commented quickly from the **Source** menu. In Python code, the **Editor > Block Commenting Style** preference controls the type of commenting that is used. The default is to use indented single `#` characters since this works better with some of Wing's other features.

**Brace Matching**

Wing highlights brace matches as you type, unless this is disabled from the **Editor > Brace Matching > Brace Highlighting** preference. The **Match Braces** item in the **Source** menu causes Wing to select all the code that is contained in the nearest matching braces, as found from the current insertion point on the editor. Repeated invocations of the command will traverse outward or forward in the file.

**Text Reformatting**
Code can be re-wrapped to the column configured in the preference Editor > Line Wrapping > Reformatting Wrap Column with the Justify Rewrap item in the Source menu. This will limit wrapping to a single logical line of code, so it can be used to reformat an argument list or long list or tuple without altering surrounding code.

**Tutorial: Searching**

Wing 101 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.

**9.1. Tutorial: Toolbar Search**

A quick way to search through the current editor or documentation page is to enter your search string into the search area provided by the toolbar:

If you enter only lower case letters, then the search will be case-insensitive. Entering one or more upper-case letters causes the search to become case-sensitive.

Try this now in example1.py: Type GetItem into the toolbar search area. Wing will search incrementally, starting when the first letter is typed. Press the Enter key to move on to the next match, wrapping around to the top of the file if necessary.

Toolbar-based searches always go forward in the file from the current editor caret position.

See the Toolbar Quick Search documentation page for details.

**9.2. Tutorial: Search Tool**

The Search tool provides a simple search and replace tool for operating on the current editor or documentation page. Key bindings for operations on this tool are given in the Search and Replace group in the Edit menu.

Searches may span the whole file or be constrained to the current selection, can be case sensitive or insensitive, and may optionally be constrained to matching only whole words.

By default, searching is incremental while you type your search string. To disable this, uncheck Incremental in the Options menu.

**Replacing**

When the tool is displayed with Replace, or when the Show Replace item in the Options menu is activated, Wing will show an area for entering a replace string and add Replace and Replace All buttons to the Search tool:
Try replacing `example1.py` with search string `PrintAs` and replace string `OutputAs`.

Select the first result match and then Replace repeatedly. One search match will be replaced at a time. Search will occur again after each replace automatically unless you turn off the Find After Replace option. Changes can be undone in the editor, one at a time. Do this now to avoid saving this replace operation.

Next, try Replace All instead. Wing will simply replace all occurrences in the file at the same time. When this is done, a single undo in the editor will cancel the entire replace operation.

See the Search Tool documentation page for more information on the Search tool.

**Tutorial: Further Reading**

Congratulations, you've finished the tutorial!

As you work with Wing 101 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.
- Wing displays a useful tip at startup, to help you continue learning about the feature set over time. These are also accessible from Wing Tips in the Help menu.

Thanks for using Wing 101!