

Introduction for New Users

Wing IDE 101

Wingware
www.wingware.com

Version 3.1.8
March 23, 2009

Thanks for trying [Wing IDE 101!](#) To get started, please **try the tutorial**.

Contents

Wing IDE Tutorial

- 1.1. Tutorial: Getting Started
- 1.2. Tutorial: Getting Around Wing IDE
 - Configuration Options
- 1.3. Tutorial: Check your Python Integration
- 1.4. Tutorial: Setting Python Path
- 1.5. Tutorial: Introduction to the Editor
 - A Note on Proxies
- 1.6. Tutorial: Debugging
 - 1.6.1. Tutorial: Debug I/O
 - 1.6.2. Tutorial: Debug Process Exception Reporting
- 1.7. Tutorial: Searching
 - Toolbar Search
 - Search Tool
 - Replacing
- 1.8. Tutorial: Other IDE Features
- 1.9. Tutorial: Further Reading

Copyright (c) 1999-2008 by Wingware. All rights reserved.:

Wingware
P.O. Box 400527
Cambridge, MA 02140-0006
United States of America

Wing IDE Tutorial

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

To get started, press the **Next** (down arrow) icon in the toolbar immediately above this page.

When using this tutorial with products other than Wing IDE Professional, please note that the screen shots include tools and features only available in Wing IDE Professional. These can safely be ignored and, when working with the tutorial within Wing IDE, those tools will not be discussed in the content that follows.

1.1. Tutorial: Getting Started

In addition to installing Wing IDE, you will also need to install Python. This tutorial will work with Python version 2.0 or later.

To get Python, download it now from python.org or wingware.com.

If the above links don't work or bring up the wrong browser, you may need to define the **BROWSER** environment variable to the name of the browser executable you wish to use (for example: `mozilla`) and restart Wing IDE.

On Linux/Unix, you can also add a browser command line to your **URL Display Commands** preference. This is recommended only if your preferred browser doesn't work when specified with the **BROWSER** environment variable. Setting **BROWSER** will generally do a better job reusing browser instances and creating and raising browser windows as needed.

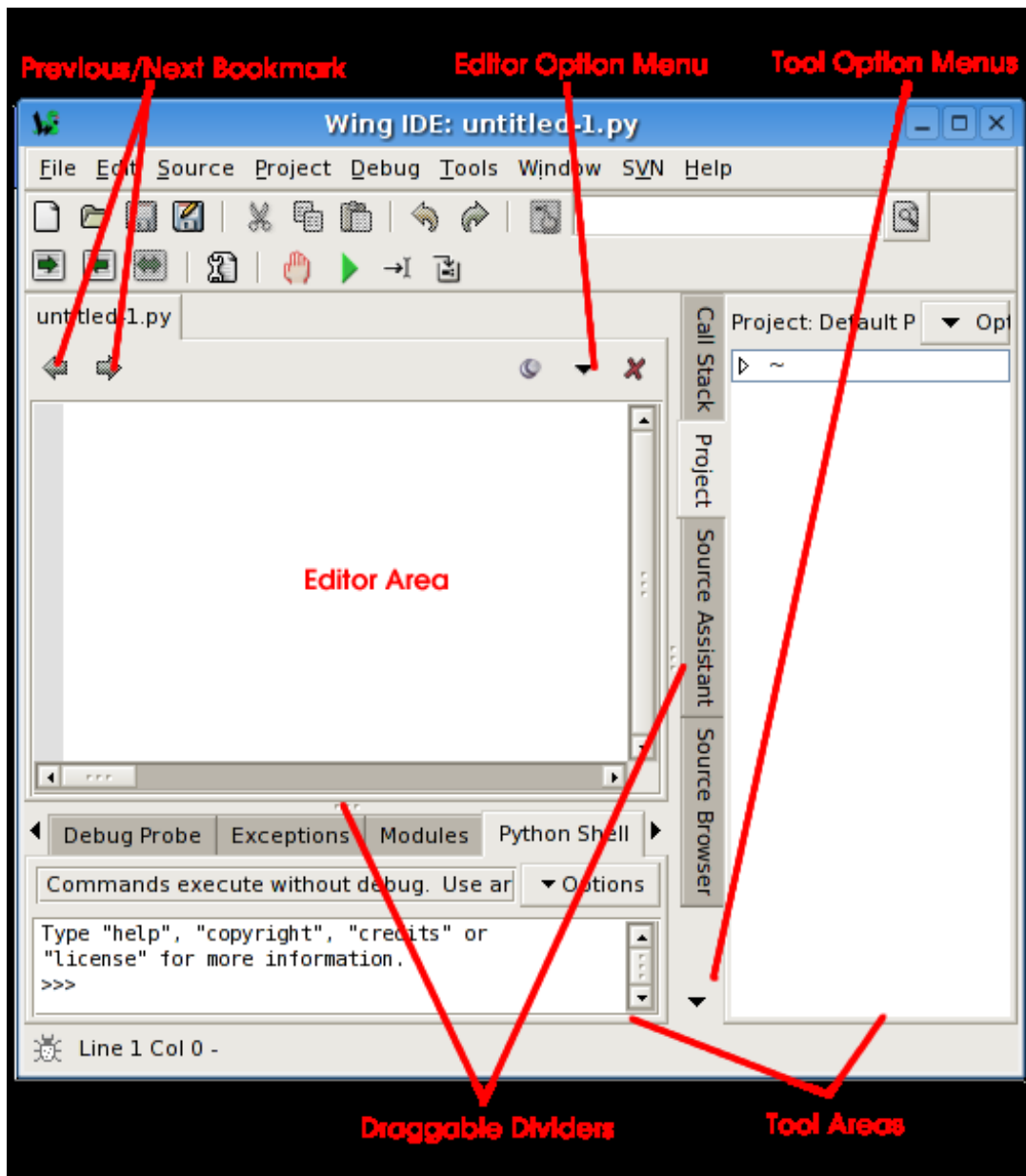
Next, copy the entire `tutorial` directory out of your Wing IDE installation to a location where you will have write access to the files in it. You can do this manually or use the following link to execute a script that will prompt you for a target directory to copy the tutorial info: **Copy Tutorial Now**

We welcome feedback and bug reports, both of which can be submitted directly from Wing IDE using the `Submit Feedback` and `Submit Bug Report` items in the `Help` menu, or by emailing us at [support at wingware.com](mailto:support@wingware.com).

1.2. Tutorial: Getting Around Wing IDE

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

Wing IDE's user interface is divided into an editor area and two tool boxes separated by draggable dividers. Use the option menus in each area to create splits or move tools around. The `Previous/Next Bookmark` buttons and the `Next Document`, `Previous Document` and `Most Recent Document` items in the `Window` menu can be used to switch quickly between documents in the editor area, such as this tutorial and the source files you'll be working with later.



Configuration Options

There are many configuration options available for customizing the user interface. Some of these are described below. Once you make changes to any of these, your settings will be remembered in your project file and preferences.

Editor Personality -- If you are used to another editor such as Visual Studio, VI or Vim,


Emacs, or Brief, you may want to put Wing into a more familiar keyboard mode using the **Personality** preference. Be sure to click **OK** or **Apply** so the changes take effect.

Tab Key Action -- In Python code, the tab key in Wing defaults to indenting a selected region or the current line to the “correct” computed indent level, to the extent that Wing can determine this from context. In non-Python files, the tab key increases indent one level. To change this, use the **Tab Key Action** preference.

Minimizing Tool Boxes -- By clicking on an already-active tool tab in one of the tool boxes, the entire area will be minimized down so that only the tabs for the area are visible. Clicking again on any tab will restore the tool box to its previous size. Or, use F1 and F2 to toggle the state of the two tool boxes. This is a convenient way to increase space available to the editor or other tool box.

Shift-F2 can also be used to maximize the editor area temporarily, hiding the tools and toolbar until Shift-F2 is pressed again.

Splitting Panels -- The editor area and tool boxes can be split into multiple sub-panels by using the editor and tool box option menus. These can be accessed either by clicking on the dropdown icon or by right-clicking on the notebook tabs. Note that when splitting the editor area, each new split will show the same files as all others; this allows for editing multiple parts of the same file.

 Splitting your editor area or creating a separate **Help** tool window may make it much easier to get around this tutorial.

The number of splits shown by default in tool boxes will vary depending on the size of your monitor.

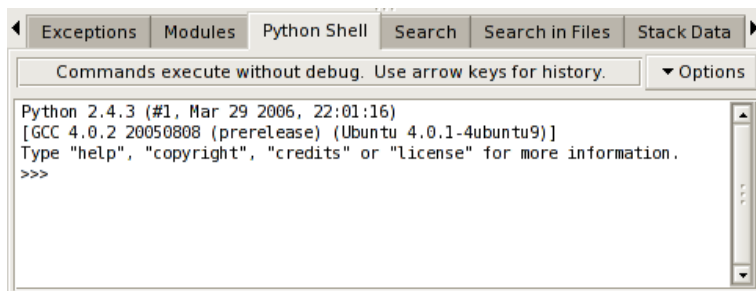
Moving and Adding Tools -- Tools can be moved among the tool boxes by using the tool box option menu. Additional instances of any tool can also be created from the tool box option menu.

Other Options -- **Source Code Font/Size** and **Display Font/Size** can be altered. The toolbar’s appearance can be changed using the **Toolbar Size** and **Toolbar Style** preferences. The tool boxes can be moved from **right to left** or **bottom to top**. The editor option menu allows selecting between using notebook tabs or a popup menu to navigate between open editors.

For more information on adjusting the user interface to your needs, see the **Customization** chapter of the manual.

1.3. Tutorial: Check your Python Integration

Before starting with some code, let's make sure that Wing has succeeded in finding your Python installation (the latest version is preferred if you have multiple versions installed). To check this, bring up the **Python Shell** tool. After a moment, it should show you the Python command prompt like this:



```

Python 2.4.3 (#1, Mar 29 2006, 22:01:16)
[GCC 4.0.2 20050808 (prerelease) (Ubuntu 4.0.1-4ubuntu9)]
Type "help", "copyright", "credits" or "license" for more information.
>>>

```

If this is not working, or the wrong version of Python is being used, you can point Wing in the right direction with the **Python Executable** setting in the **Python Configuration**, available in the **Edit** menu. You will need to **Restart Shell** from **Options** in the Python Shell tool after altering this property.

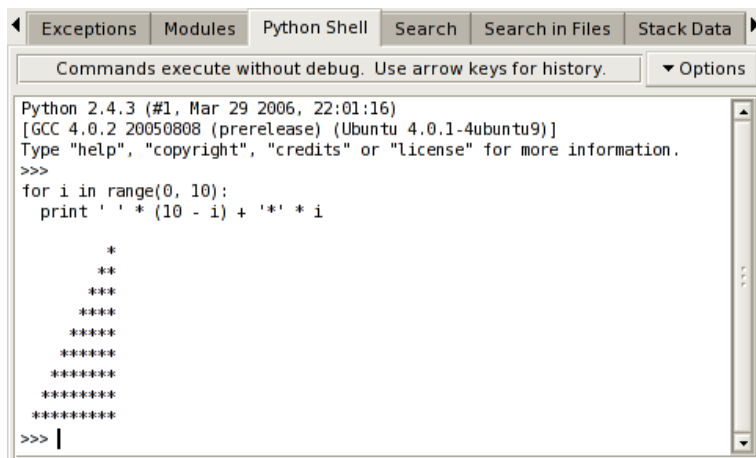
Once the shell works, copy/paste or drag and drop these lines of Python code into it:

```

for i in range(0, 10):
    print ' ' * (10 - i) + '*' * i

```

This should print a triangle as follows:



```

Python 2.4.3 (#1, Mar 29 2006, 22:01:16)
[GCC 4.0.2 20050808 (prerelease) (Ubuntu 4.0.1-4ubuntu9)]
Type "help", "copyright", "credits" or "license" for more information.
>>>
for i in range(0, 10):
    print ' ' * (10 - i) + '*' * i

    *
   **
  ***
 ****
*****
*****
*****
*****
*****
*****
>>> |

```

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 in the shell, such as:

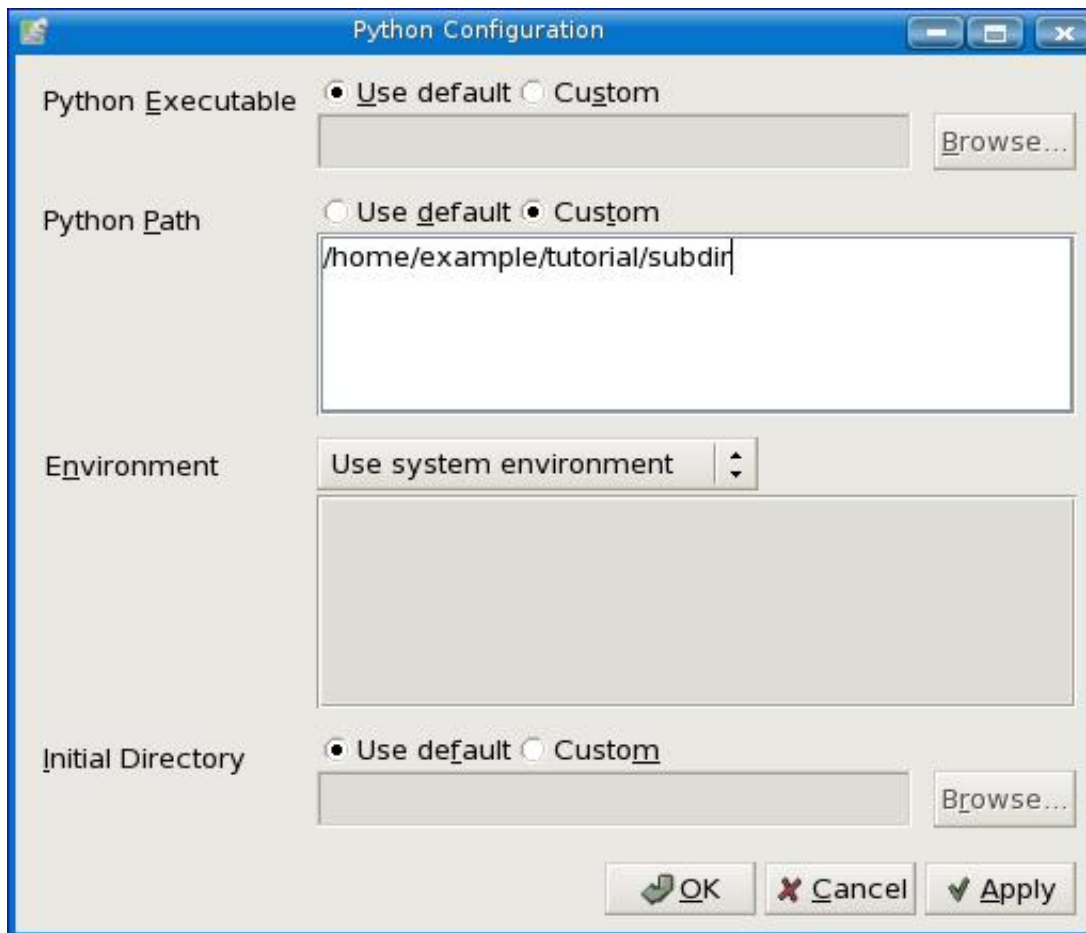
```
import sys
sys.getrefcount(i)
```

You can create as many instances of the Python Shell tool as you wish; each one runs in its own private process space that is kept totally separate from Wing IDE and your debug process.

1.4. Tutorial: Setting Python Path

Whenever your Python source depends on `PYTHONPATH` (either set externally or by altering `sys.path` internally), you will also need to tell Wing about your path.

This value can be entered using `Configure Python` in the `Edit` menu, in the `Python Path` field:



For this tutorial, you will at least need a `PYTHONPATH` that includes the `subdir` sub-directory of your `tutorials` directory as shown in the figure above. This contains a module used as part of the first coding example.

Note that in the screen shot above the `PYTHONPATH` has been set with the full path to the directory `subdir`. This is strongly recommended because it avoids potential problems finding source code when the starting directory is ambiguous, both for source code analysis purposes and in Wing's debugger. A partial path can be specified, but Wing will issue a warning explaining why this is a bad idea.

The configuration is used here for illustration purposes. You could easily run the example code without a `PYTHONPATH` 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. Either of these allows Python to find the modules without altered `PYTHONPATH`.

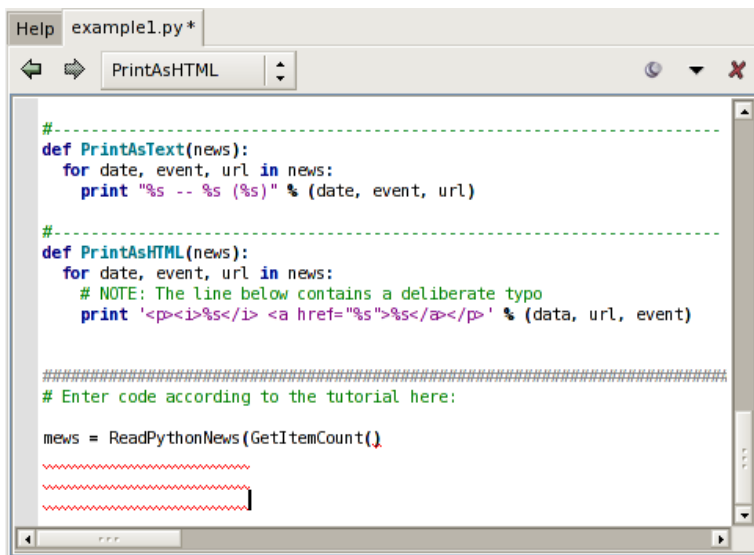
1.5. Tutorial: Introduction to the Editor

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:

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

Press enter a few times. Note that Wing IDE auto-indent 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:



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:

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

Then enter the following two additional lines of code:

```
PrintAsText(news)
PrintAsHTML(news)
```

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

A Note on Proxies

If you are behind a firewall and use a web proxy, you may need to alter the tutorial before it will work. In particular, set up a proxy mapping like this:

```
proxies = {'http': 'http://192.168.3.7:3128'}
```

And then add a second parameter to the urllib call that obtains the news in `ReadPythonNews` so it looks like this:

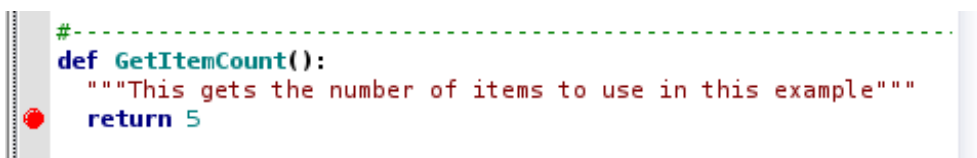
```
svc = urllib.urlopen("http://www.python.org/channews.dat", proxies=proxies)
```

You will of course need to set the http proxy url according to your local network's configuration.

1.6. Tutorial: Debugging

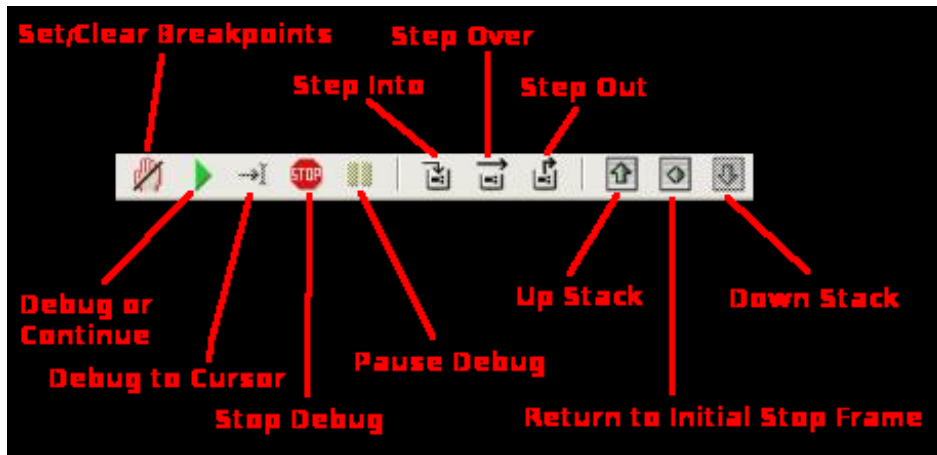
In case you haven't already figured it out, the `example1.py` program you have created connects to `python.org` via HTTP, reads and parses the Python-related news feed that is hosted there, and then prints the most recent five items as text and HTML. Don't worry if you don't have an internet connection on your machine; 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 dark margin to the left of the line. The breakpoint should appear as a filled red circle:



Next start the debugger from the bug icon in the toolbar or the **Start/Continue** item in the **Debug** menu.

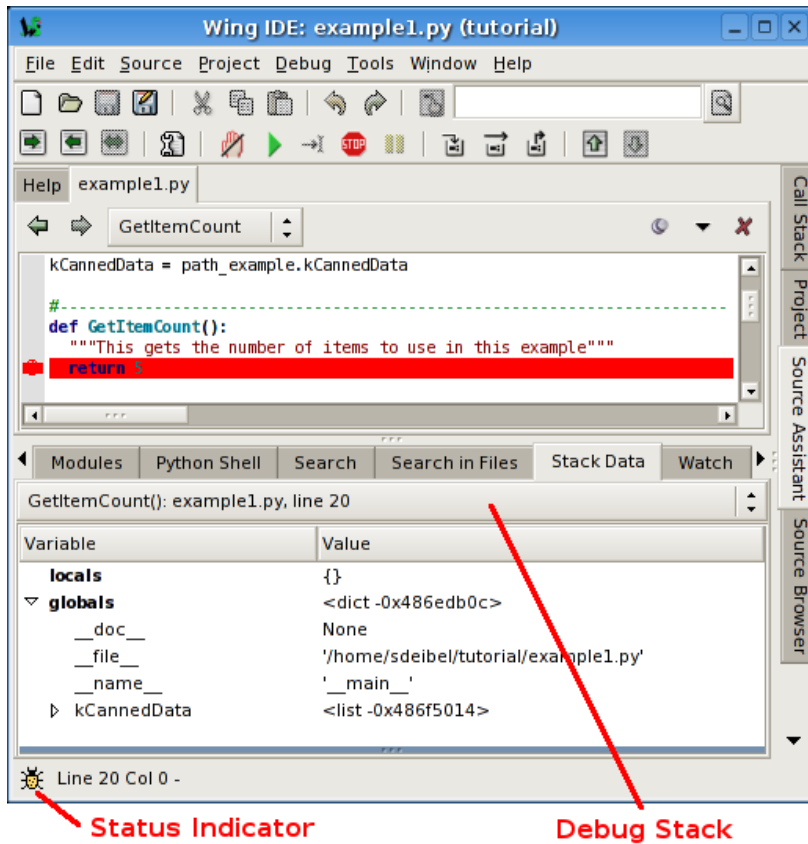
Wing will run to the breakpoint and stop, placing a red indicator on the line. Notice that the toolbar changes to include additional debug tools, as shown below:



Your display may vary depending on how you have configured the **Toolbar Size** and **Toolbar Style** preferences. Note that Wing displays tooltips explaining what the tools do when you 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 from the toolbar or 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. 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** toolbar icon or **Debug** menu item (two clicks on **Step Over** also work). This is a good function to step through in order to familiarize yourself with the basic debugger features covered above.

1.6.1. 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 for it.

Once you step over the line `PrintAsText(news)` you should see output appear as follows:



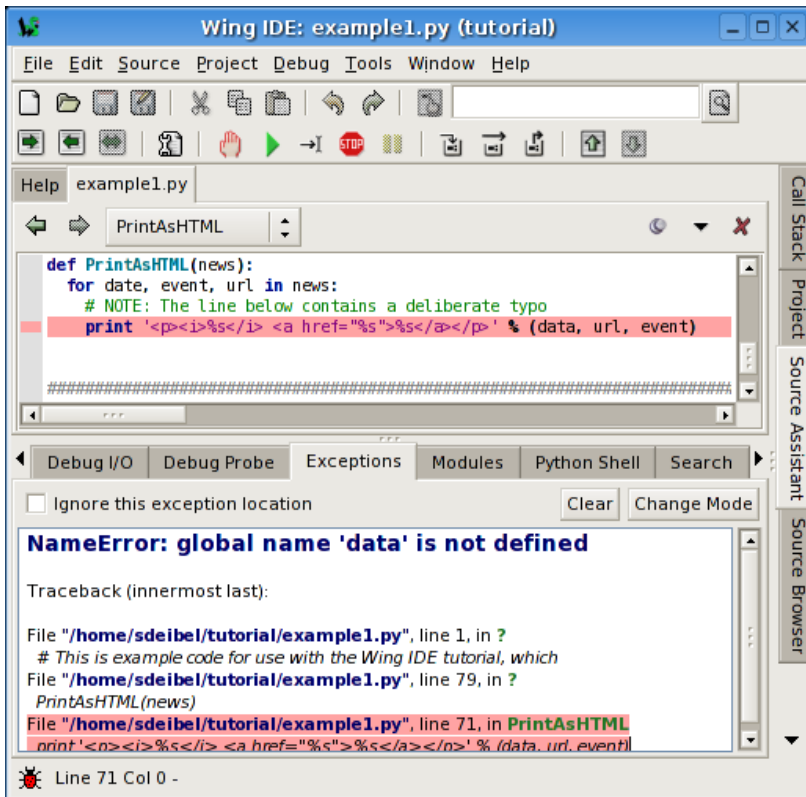
Note that you can also configure Wing to use an external console from the **Options** menu in the Debug I/O tool. This is useful for code that depends on details of the Debug I/O environment (such as cursor control with special output characters).

1.6.2. Tutorial: Debug Process Exception Reporting

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

Try this out by continuing execution of the debug process with the **Debug** toolbar item or **Start / Continue** item in the **Debug** menu.

Wing will stop on an incorrect line of code in `PrintAsHTML` and will report the error in the **Exceptions** tool:



Notice that this tool highlights the current stack frame and that you can click on frames to navigate the exception backtrace. Whenever you are stopped on an exception, the Debugger Status indicator in the lower left of Wing's main window turns red.

1.7. Tutorial: Searching

Wing IDE provides several different interfaces for searching your code. Which you use depends on your task.

Toolbar Search

A quick way to search through the current editor is to enter your search string in the area provided in the toolbar:



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

Try this now in `example1.py`: Type `GetItem` in the toolbar search area and Wing will immediately, starting with the first letter typed, search for matching text in the editor. Notice that if you press the `Enter` key, Wing will move on to the next match, wrapping around to the top of the file if necessary.

Toolbar-based searches always go forward (downward) in the file from the current cursor position.

Search Tool

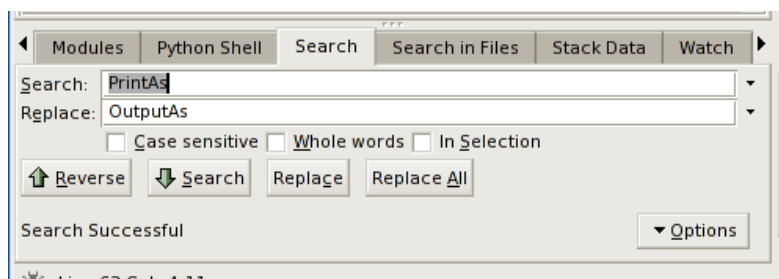
The **Search** tool provides a familiar GUI-based search and replace tool for operating on the current editor. Key bindings for operations on this tool are given in the **Search and Replace** group in the **Edit** menu.

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 **Show Replace** item in **Options** is activated, Wing will show an area for entering a replace string and adds **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.

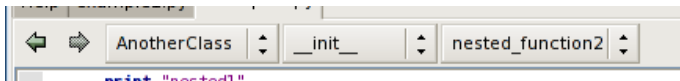
Next, try **Replace All** instead. Wing will simply replace all occurrences in the file at the same time. When this is done, a single undo in the editor will cancel the entire replace operation.

1.8. Tutorial: Other IDE Features

There are a number of other features available in the IDE that are worth noting:

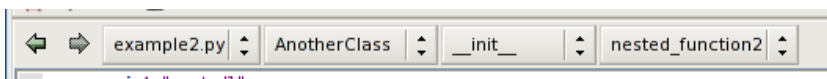
Source Index -- The top of the editor area displays a series of popup menus that act as an index into Python source files. Select from them to navigate around your source file.

Try this out by opening `example2.py` from your `tutorial` directory. If you place the cursor on the line that reads `print "nested2"`, you should see the following in the source index area:




Each subsequent menu lists the symbols available within the preceding nested context.

If you have turned off **Show Notebook Tabs** in the Editor Options Menu, the file selector menu will be prepended as follows:



Goto-Definition -- There are a number of ways to navigate to the point of definition of symbols in your source code. One is to right-click on the symbol and select **Goto Definition**. Another is to move the insertion cursor to the symbol and select **Goto Selected Symbol Defn** from the **Source Menu** (or press F4). The Source Assistant in Wing IDE Pro also contains links to points of definition.

Try this from `example2.py` with some of the symbols imported from `htmllib`, such as `HTMLParser` in the class definition for `MyHTMLParser`. Remember that the file `htmllib.py` is opened in non-sticky mode and will auto-close unless you toggle the stick pin icon to 

or edit the file.

Goto-Line -- Navigate quickly to a numbered source line with the **Goto Line** item in the **Edit** menu. In emacs mode, the line number is typed into the data entry area that appears at the bottom of the window. Press **Enter** to complete the action.

Auto-Indentation -- Wing auto-indent lines as you type according to its static analysis of your code. This can be disabled with the **Auto-Indent** preference.

Another way in which Wing uses code analysis is in auto-indentation as you type, and for altering indentation or wrapping of code. For example, when you select a block of code and press the tab key, the entire block is re-indented according to the correct position of its first line relative to the preceding non-blank line of code. The **Justify Text** option in the **Source** menu also uses the source analyser to constrain re-wrapping to a single logical line of Python code.

Block Indentation -- 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-Tab**). The **Tab Key Action** preference can be used to customize how the tab key behaves.

One or more selected lines can be increased or reduced in indentation from the Indentation toolbar group, which contains the following icons for this purpose:



Single lines or whole blocks can also be indented automatically to their appropriate position, as determined by analysis of the preceding line. If a range of lines is selected, the whole block is indented or outdented without changing the relative indents within the block. This is done from the following toolbar icon:



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

Block Commenting -- Units of code can be commented out or un-commented quickly from the **Source** menu.

Brace Matching -- Wing highlights brace matching as you type unless disabled from the **Auto Brace Match** preference. The **Match Braces** item in the **Source** menu causes

Wing to select all the code that is contained in the nearest matching braces found from the current insertion point on the editor. Repeated application of the command will traverse outward and forward in the file.

Text Reformatting -- Code can be re-wrapped with the **Justify Text** item in the **Source** menu. This will limit wrapping to a single logical line of code, so it can be used for wrapping an argument list or long list or tuple without altering surrounding code.

1.9. Tutorial: Further Reading

Congratulations! You've finished the tutorial. As you work with Wing IDE on your own software development project, the following resources may be useful:

- [Wing IDE Support Website](#)
- **Wing IDE Reference Manual**
- **OS X Quickstart**