Introduction for New Users

This tutorial introduces Wing IDE by taking you through its feature set with a small coding example. We strongly recommend using the **Tutorial** in Wing IDE's **Help** menu when actually working through the tutorial, rather than reading the printed form. The integrated form of the tutorial contains links into the IDE's functionality that are not found here.

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

Contents

Wing IDE Tutorial

This tutorial introduces Wing IDE 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 IDE 101 to teach programming with Python.

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 may include tools and features only available in Wing IDE Professional. These can safely be ignored and, when working with the tutorial within Wing IDE's help viewer, those tools will not be discussed in the content that follows.

1.1. Tutorial: Getting Started

To get started, you need to:

(1) Install Python and Wing IDE

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

(2) Start Wing IDE

Wing can be started from a menu, desktop, or tray icon or using the command line executable. For detailed instructions, see Running the IDE.

If you don't have a license, you can obtain a 30-day trial the first time you start Wing.

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 (3) below).

(3) Copy the Tutorial Directory

Next, copy the entire **tutorial** directory out of the top level 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, which will prompt you to select the target directory into which to copy the tutorial: **Copy Tutorial Now**

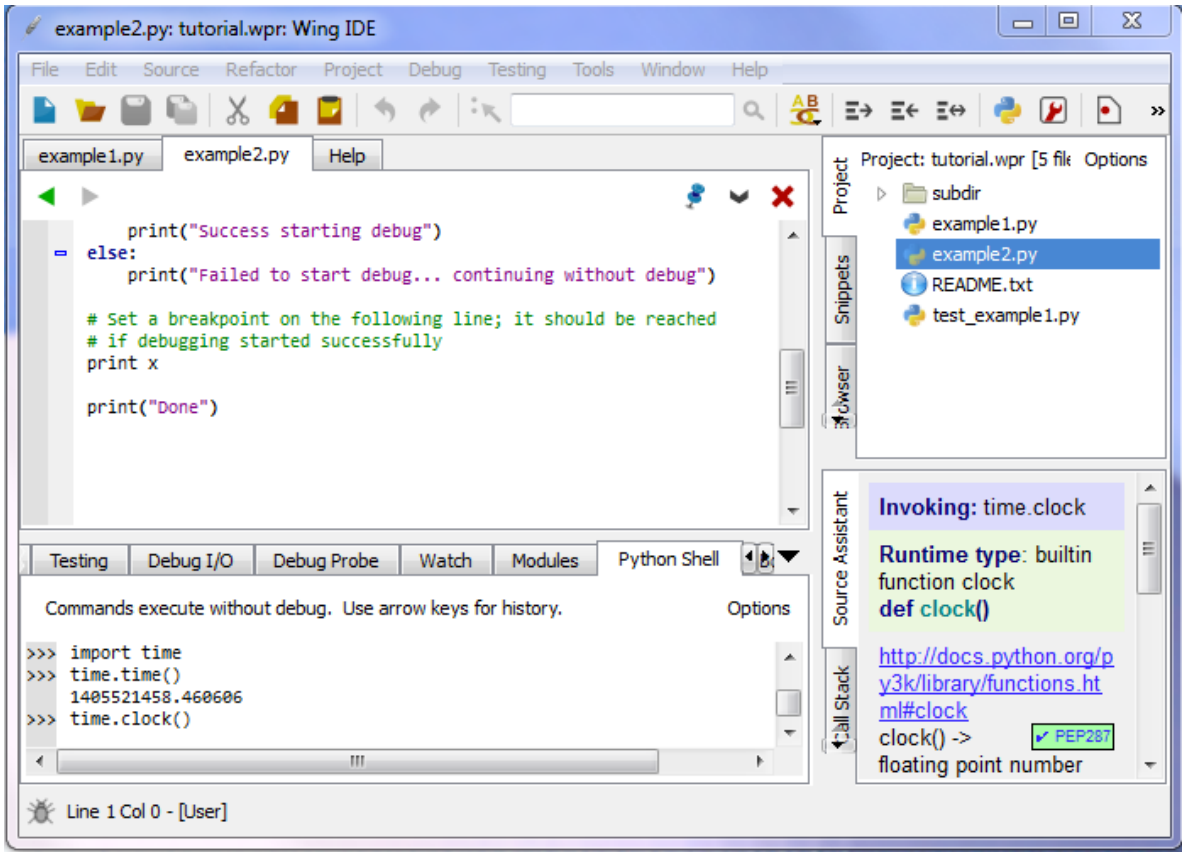
> ***Note***
>
> We welcome feedback, which can be submitted with **Submit Feedback** in Wing's **Help** menu or by emailing support at wingware.com

To get to the next page in the tutorial, use the **Next Page** icon shown in the toolbar just above this text:

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 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 monitor.

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.

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.

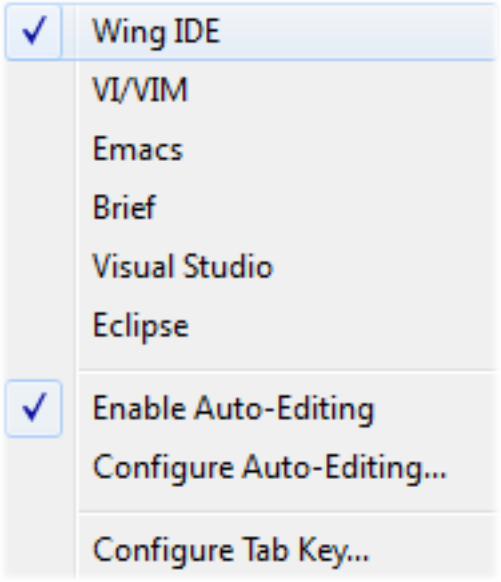
Splitting your editor area makes it easier to get around this tutorial. To do this now, right click on the editor tab area and select **Split Side by Side**. On small monitors and laptops, it may be preferable to create a new window for the tutorial by right clicking on its tab and selected **Move Wing IDE Help to New Window**.

Context Menus

In general, right-clicking provides a menu for interacting with or configuring a part of the user interface. The text that follows refers to these menus as "context menus".

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, or Brief.



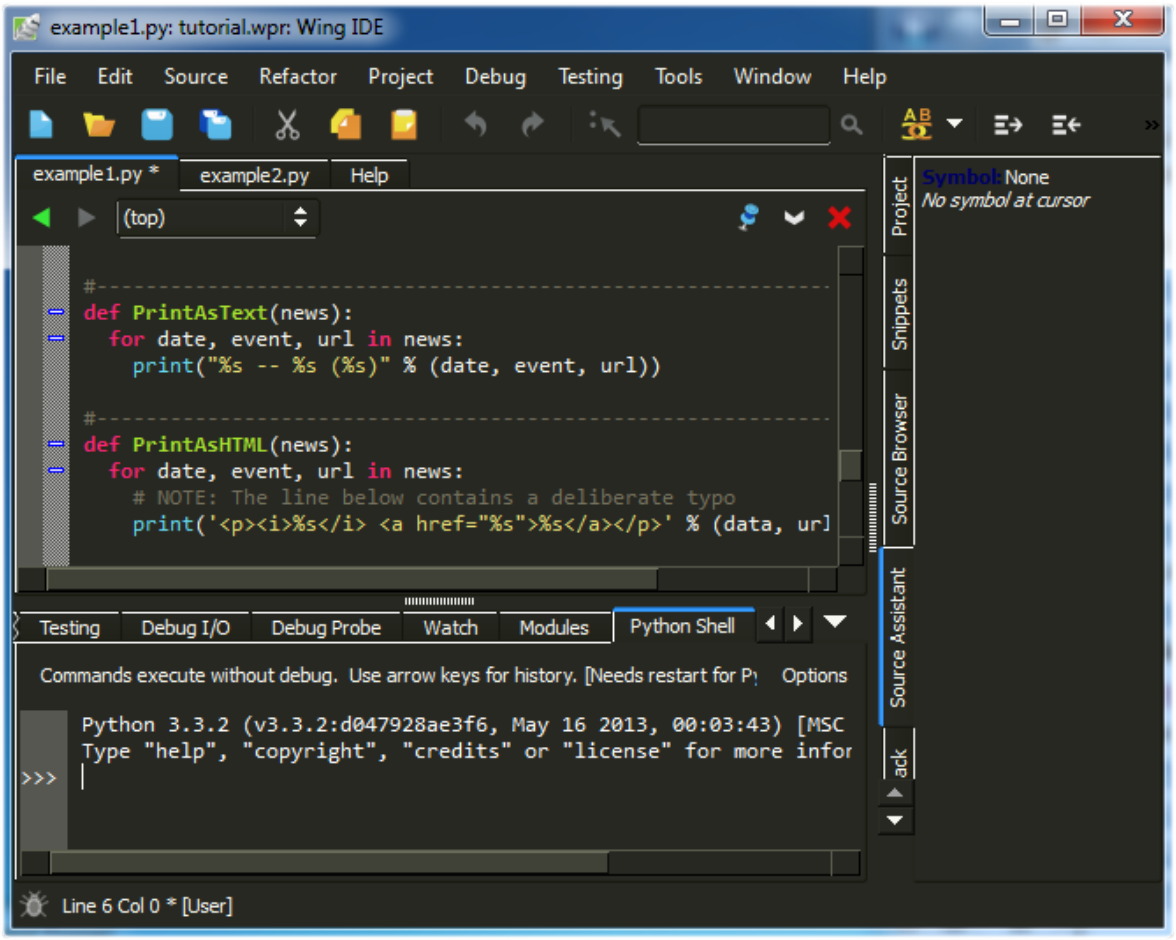
The **Configure Tab Key** item in the **Edit > Keyboard Personality** menu or the **User Interface > Keyboard > Tab Key Action** preference can be used to select among available behaviors for the tab key. The default is to match the selected Keyboard Personality. When the Keyboard Personality is set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

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

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 > Editor Color Palette** preference. To apply this palette also to the rest of the UI (outside of editors), enable the **Use Editor Palette Throughout the UI** preference.

To set a dark background display style select one of **One Dark**, **Monikai**, **Black Background**, or **Solarized - Dark** as the **Editor Color Palette**.



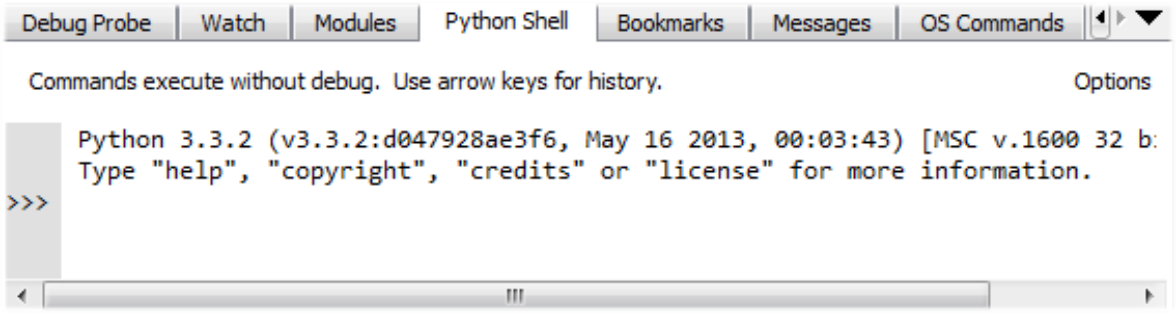
The **User Interface > Fonts > Display Font/Size** and **User Interface > Fonts > Editor Font/Size** preferences select fonts for the user interface and editor.

The size and type of tools used in the toolbar at the top of Wing IDE's main 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 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. 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 the **Python Executable** setting in the **Python Configuration**, available in the **Edit** menu.

An easy way to determine the path to use here is to start the Python you wish to use with Wing and type the following at Python's **>>>** prompt:

```
import sys
sys.executable
```

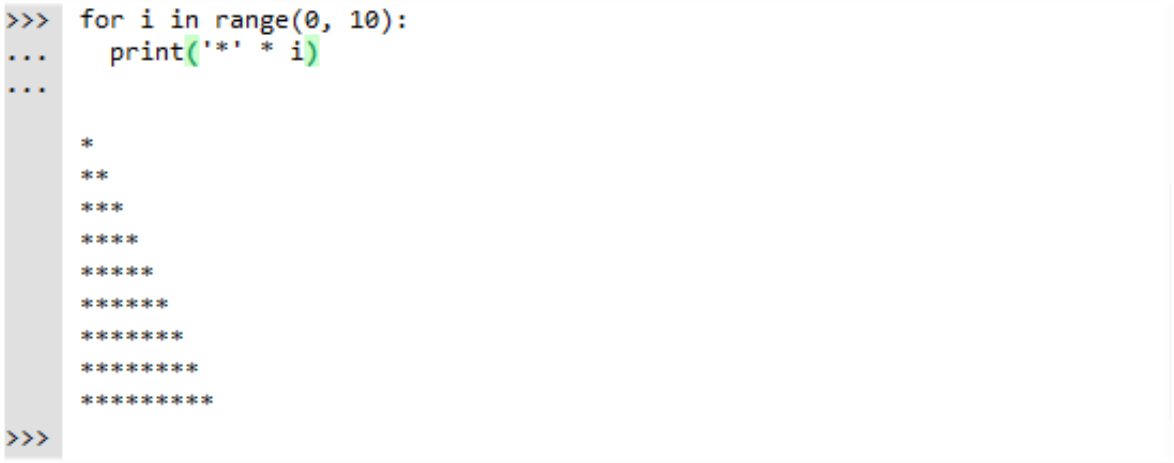
This can also be typed into the IDLE that is associated with your Python install, if IDLE is installed. On OS X this is generally the easiest way to find the correct executable to use.

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:

```
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 in the shell, such as:

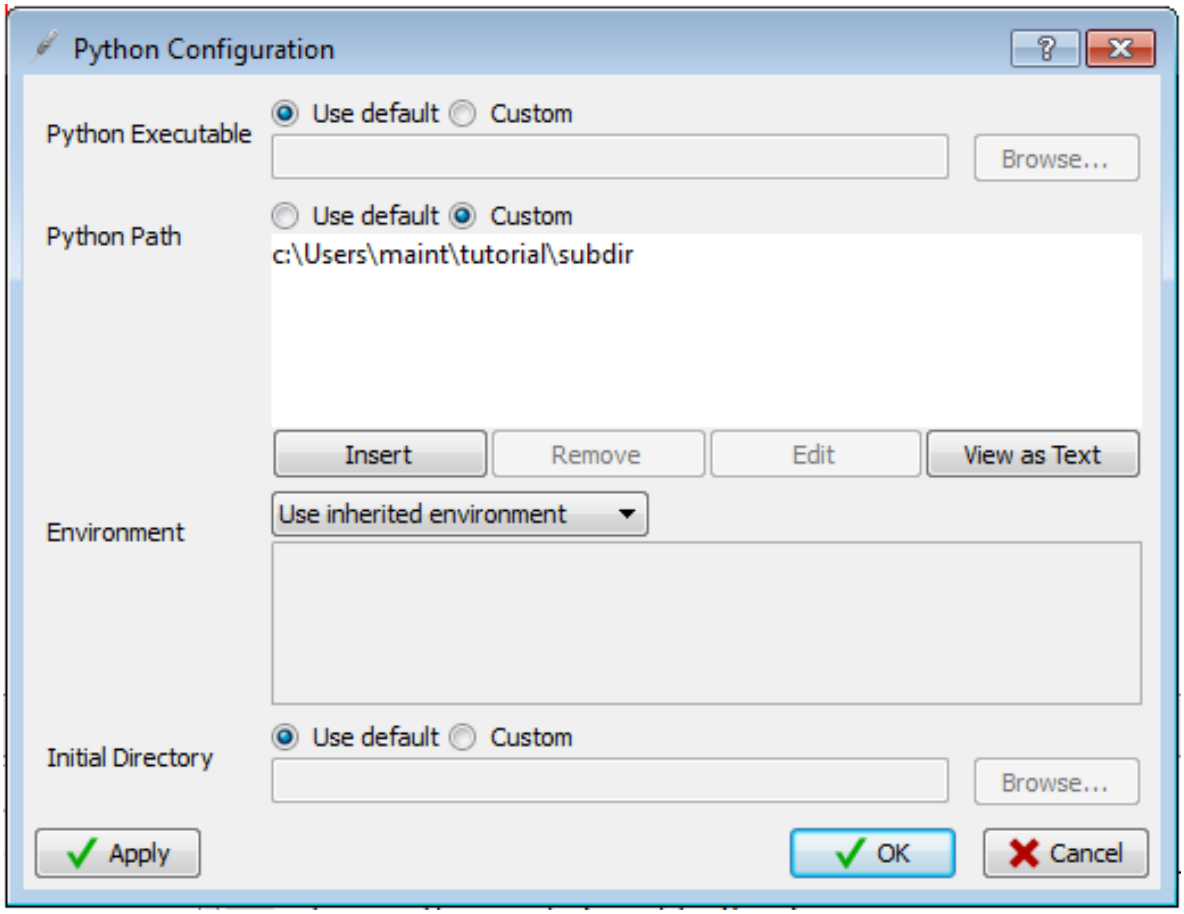
```
import sys
sys.getrefcount(i)
```

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** at runtime, you will also need to tell Wing about your path.

This value can be entered in **Python Path** in the **Configure Python** dialog, which is 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 when 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 **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, which is in the default **PYTHONPATH**.

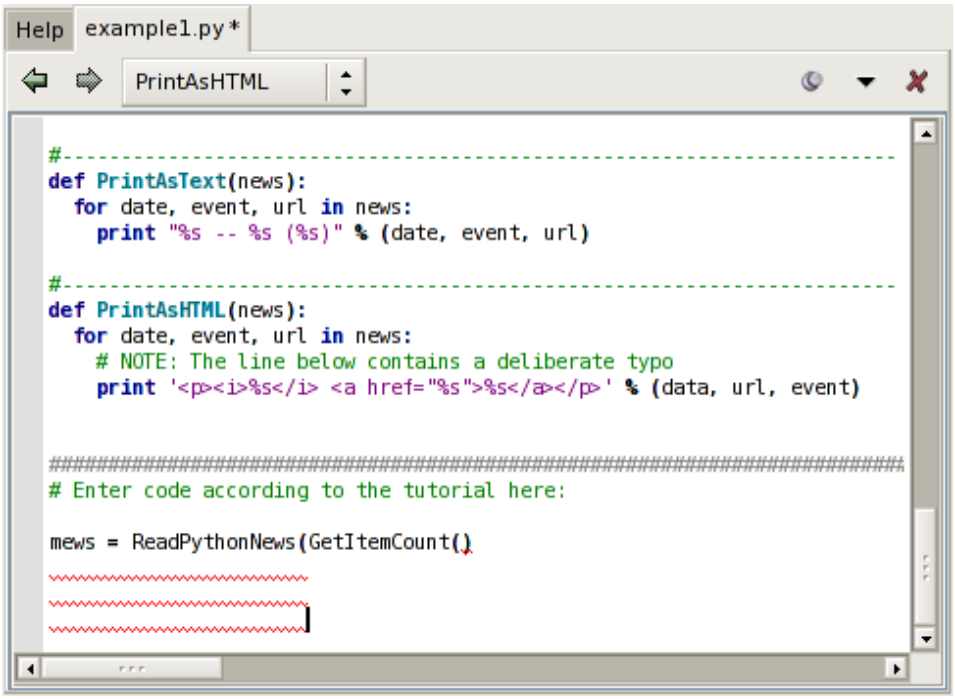
1.5. Tutorial: Introduction to the Editor

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-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:



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 additional lines of code:

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

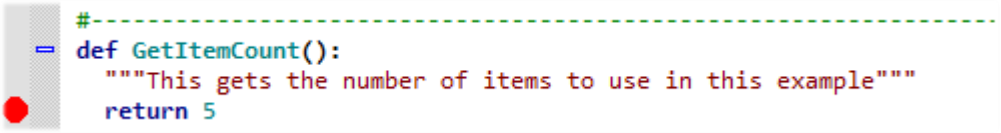
At this point you have a complete program that can be run in the debugger.

1.6. 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**.

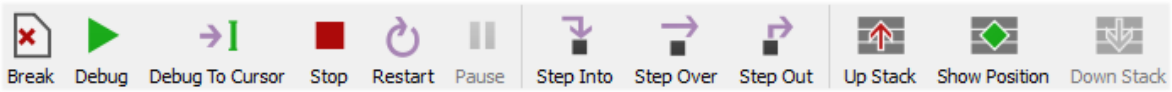
To start debugging, set a breakpoint on the line that reads **return 5** in the **GetItemCount** function. This can be done by clicking on the line and selecting the

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

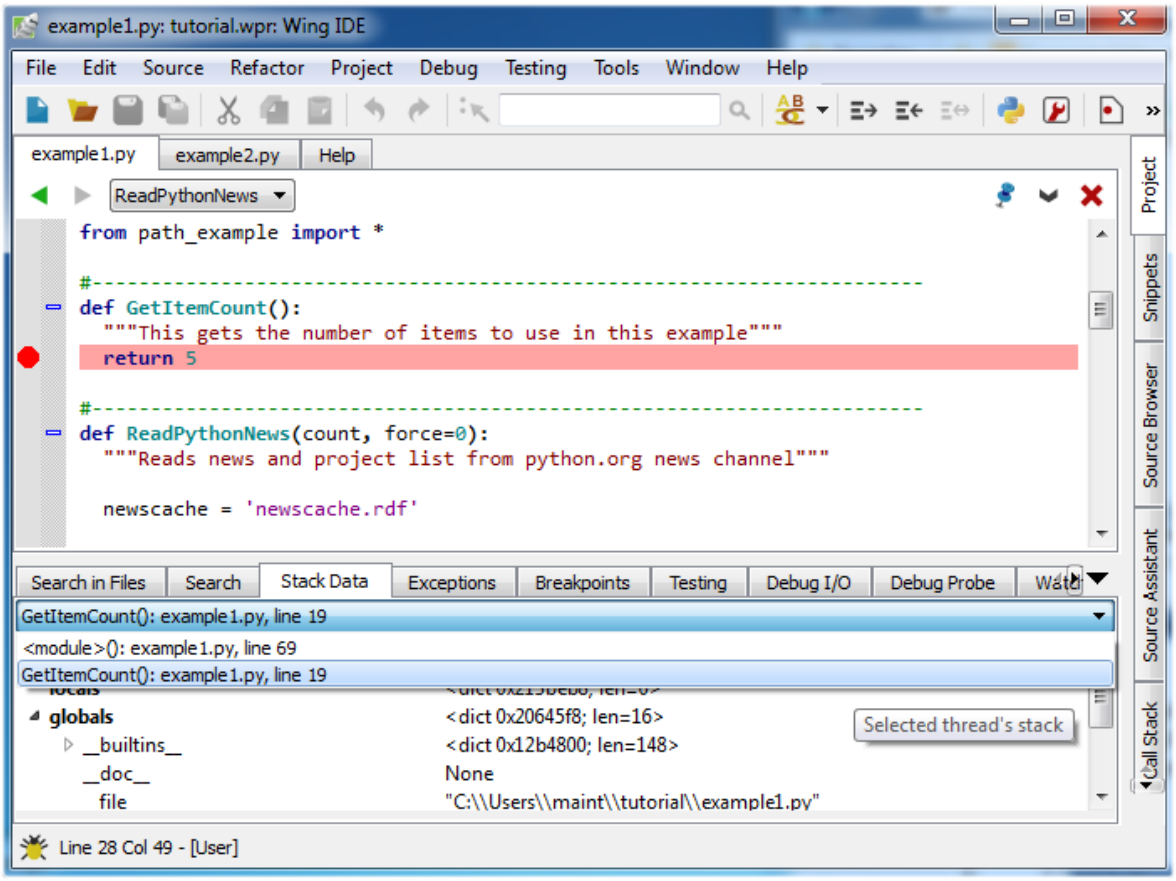
Wing will run to the breakpoint and stop, placing a red indicator on the line. Notice that the toolbar changes to include additional debug tools, as shown below:



Your display may vary depending on the size of your screen or if you have altered the toolbar's configuration. 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:

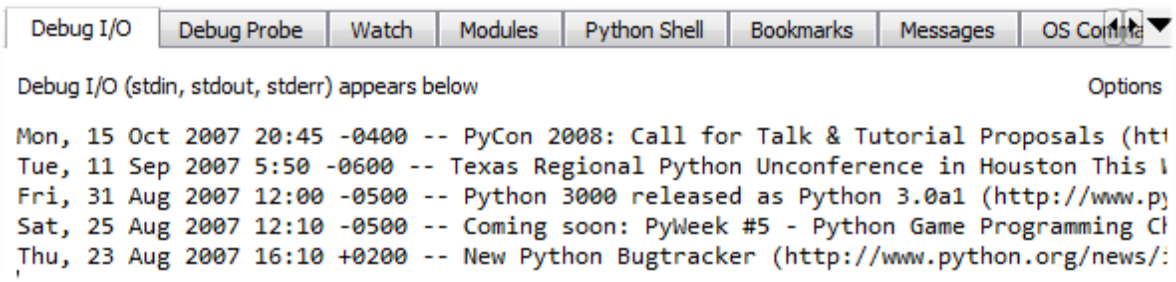


Next, try stepping out to the enclosing call to **ReadPythonNews**. In this particular context, you can achieve this in a single click with the **Step Out** toolbar icon or **Debug** menu item. Two clicks on **Step Over** also work. **ReadPythonNews** is a good function to step through in order to try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



For code that tries to read from **stdin** or uses **input** (or in Python 2.x **raw_input**), the **Debug I/O** tool is where you would type your input to your program. Try this now by stepping over the **PromptToContinue** call. 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 focus is in the **Debug I/O** tool.

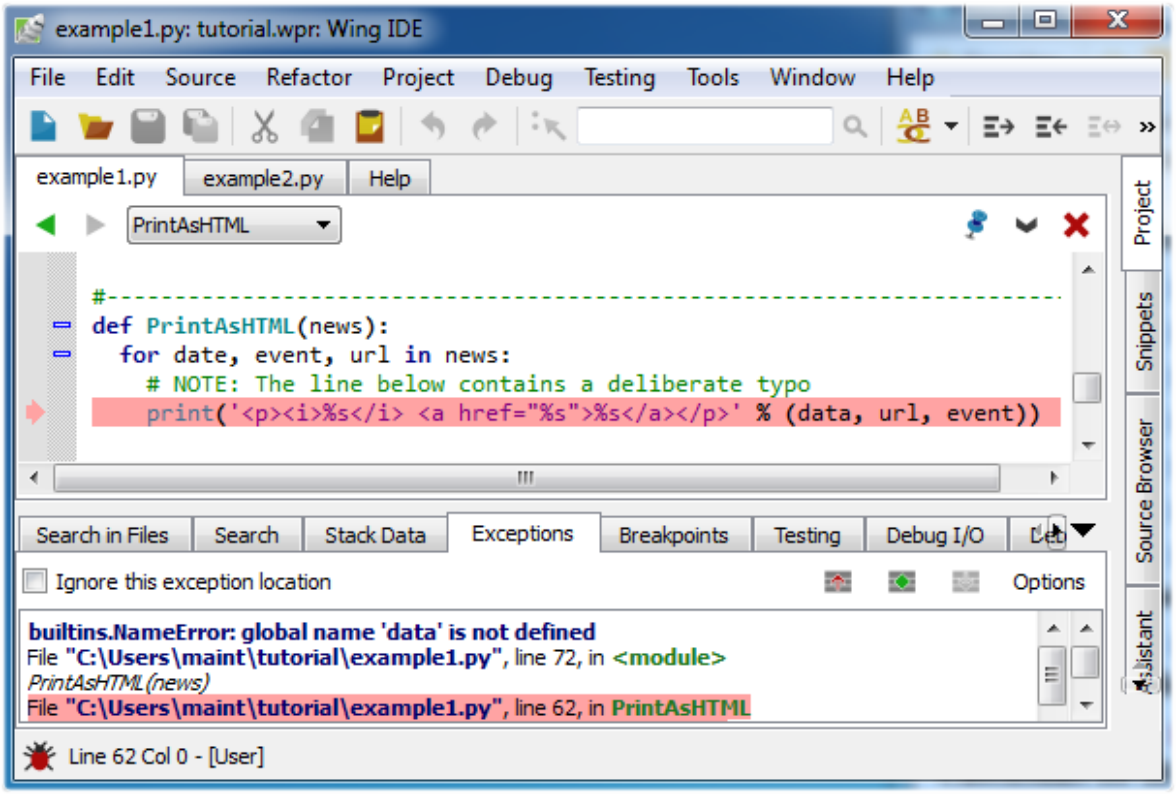
Note that you can also configure Wing to use an external console from the **Options** menu in the **Debug I/O** tool. This is useful for code that depends on details of the **Debug I/O** environment (such as cursor control with special output characters).

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.

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

1.6.3. 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** (and in Wing Pro the **Debug Probe**).

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** should 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 also be reported in the debugger.

Let's try this out. 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)
```

Then place a breakpoint on the line that reads **print(x)**.

Next type this into the **Python Shell** and press **Enter**:

test_function()

Wing should reach the breakpoint on the **print(x)** line.

You can now work with the debugger in the same way that you would if you have 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 withing 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 red **Stop** button 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 becaused there is no way to simply abandon a number of stack frames in the Python interpreter.

1.7. Tutorial: Indentation Features

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

Auto-Indentation

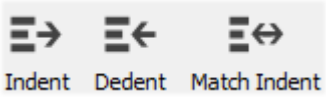
By now you will have noticed that Wing auto-indents lines as you type, according to context. This can be disabled with the **Auto-Indent** preference.

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.

Block Indentation

In Wing's default keyboard personality, the **Tab** key is defined to indent the current line or blocks of lines, rather then entering a tab character (which can be done with **Ctrl-Tab**). As noted earlier, 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, or matche 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.

1.8. 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 from the **Selection Mode** item 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 > 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**.

Block Commenting

Lines of code can be commented out or un-commented quickly from the **Source** menu . In Python code, the **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 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 invocations of the command will traverse outward or forward in the file.

Text Reformatting

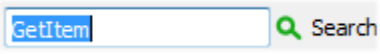
Code can be re-wrapped to the configured **Reformatting Wrap Column** with the **Justify Rewrap** item in the **Source** menu. This will limit wrapping to a single logical line of code, so it can be used to reformat an argument list or long list or tuple without altering surrounding code.

1.9. 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:



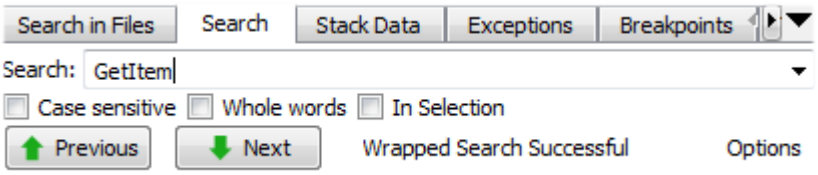
If you enter only lower case the search will be case-insensitive. Entering one or more upper-case letter causes the search to become case-sensitive.

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. 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 (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.

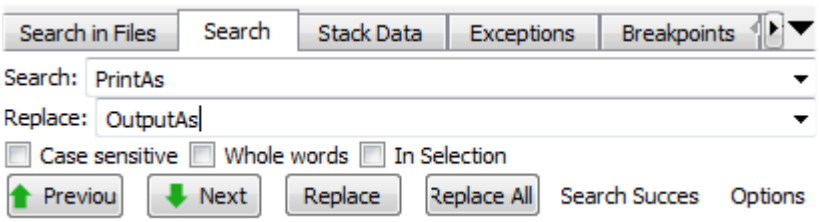


Searches may span the whole file or be constrained to the current selection, can be case sensitive or insensitive, and may optionally be constrained to matching only whole words.

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 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.

1.10. Tutorial: Further Reading

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

- Wing IDE Support Website which includes our mailing lists and other information for Wing IDE users.
- Wing IDE Reference Manual which documents the features in detail.