Getting Started With Python IDLE

Table of Contents

If you’ve recently downloaded Python onto your computer, then you may have noticed a new program on your machine called **IDLE**. You might be wondering, “What is this program doing on my computer? I didn’t download that!” While you may not have downloaded this program on your own, IDLE comes bundled with every Python installation. It’s there to help you get started with the language right out of the box. In this tutorial, you’ll learn how to work in Python IDLE and a few cool tricks you can use on your Python journey!

In this tutorial, you’ll learn:

- What Python IDLE is
- How to interact with Python directly using IDLE
- How to edit, execute, and debug Python files with IDLE
- How to customize Python IDLE to your liking

What Is Python IDLE?

Every Python installation comes with an **Integrated Development and Learning Environment**, which you’ll see shortened to IDLE or even IDE. These are a class of applications that help you write code more efficiently. While there are many IDEs for you to choose from, Python IDLE is very bare-bones, which makes it the perfect tool for a beginning programmer.

Python IDLE comes included in Python installations on Windows and Mac. If you're a Linux user, then you should be able to find and download Python IDLE using your package manager. Once you've installed it, you can then use Python IDLE as an interactive interpreter or as a file editor.

An Interactive Interpreter

The best place to experiment with Python code is in the interactive interpreter, otherwise known as a **shell**. The shell is a basic Read-Eval-Print Loop (REPL). It reads a Python statement, evaluates the result of that statement, and then prints the result on the screen. Then, it loops back to read the next statement.

The Python shell is an excellent place to experiment with small code snippets. You can access it through the terminal or command line app on your machine. You can simplify your workflow with Python IDLE, which will immediately start a Python shell when you open it.

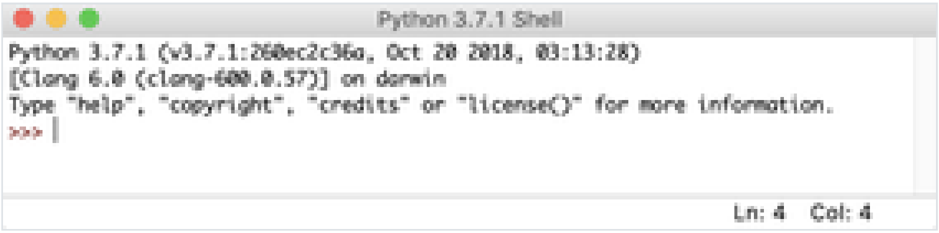
A File Editor

Every programmer needs to be able to edit and save text files. Python programs are files with the `.py` extension that contain lines of Python code. Python IDLE gives you the ability to create and edit these files with ease.

Python IDLE also provides several useful features that you'll see in professional IDEs, like basic syntax highlighting, code completion, and auto-indentation. Professional IDEs are more robust pieces of software and they have a steep learning curve. If you're just beginning your Python programming journey, then Python IDLE is a great alternative!

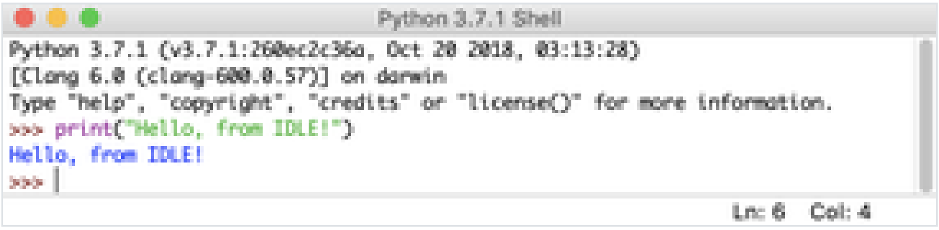
How to Use the Python IDLE Shell

The shell is the default mode of operation for Python IDLE. When you click on the icon to open the program, the shell is the first thing that you see:



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

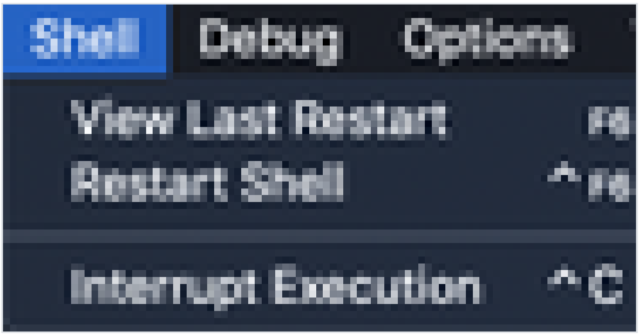
This is a blank Python interpreter window. You can use it to start interacting with Python immediately. You can test it out with a short line of code:



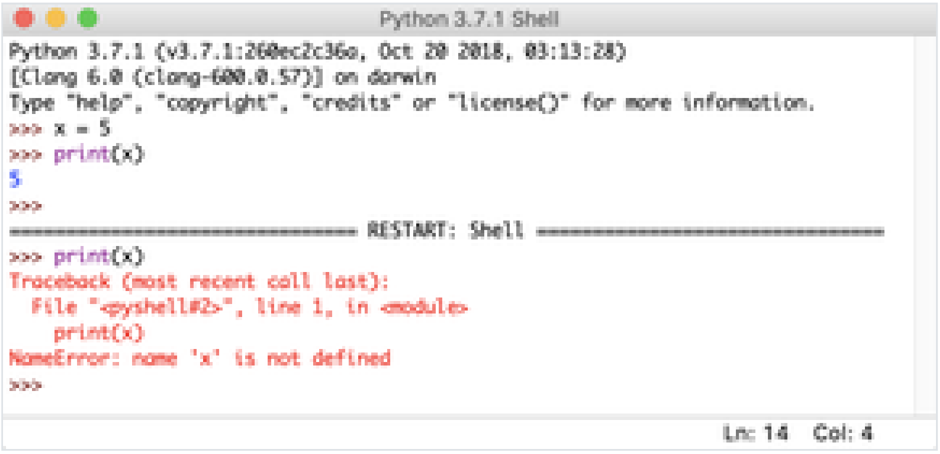
```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, from IDLE!")
Hello, from IDLE!
>>> |
```

Here, you used `print()` to output the string `"Hello, from IDLE!"` to your screen. This is the most basic way to interact with Python IDLE. You type in commands one at a time and Python responds with the result of each command.

Next, take a look at the menu bar. You'll see a few options for using the shell:



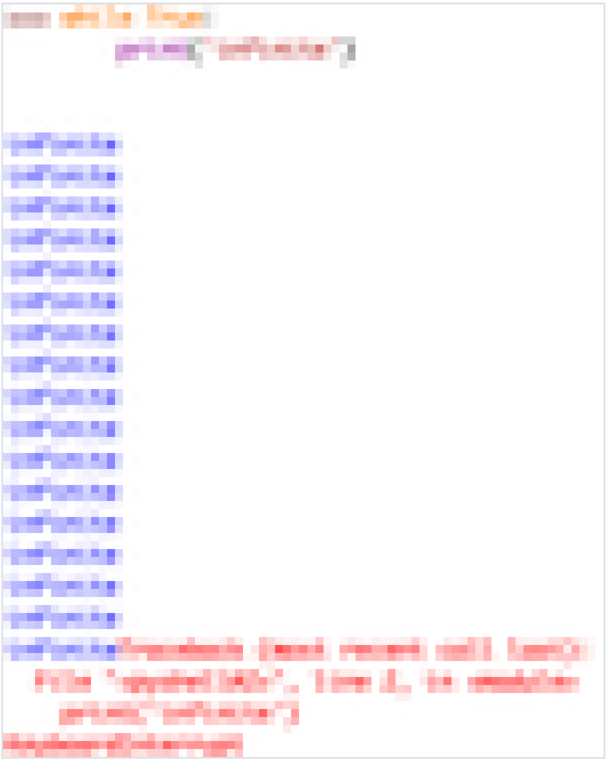
You can restart the shell from this menu. If you select that option, then you'll clear the state of the shell. It will act as though you've started a fresh instance of Python IDLE. The shell will forget about everything from its previous state:



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> x = 5
>>> print(x)
5
>>>
================================ RESTART: Shell ================================
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

In the image above, you first declare a variable, `x = 5`. When you call `print(x)`, the shell shows the correct output, which is the number `5`. However, when you restart the shell and try to call `print(x)` again, you can see that the shell prints a traceback. This is an error message that says the variable `x` is not defined. The shell has forgotten about everything that came before it was restarted.

You can also interrupt the execution of the shell from this menu. This will stop any program or statement that's running in the shell at the time of interruption. Take a look at what happens when you send a keyboard interrupt to the shell:



A `KeyboardInterrupt` error message is displayed in red text at the bottom of your window. The program received the interrupt and has stopped executing.

How to Work With Python Files

Python IDLE offers a full-fledged file editor, which gives you the ability to write and execute Python programs from within this program. The built-in file editor also includes several features, like code completion and automatic indentation, that will speed up your coding workflow. First, let’s take a look at how to write and execute programs in Python IDLE.

Opening a File

To start a new Python file, select *File* → *New File* from the menu bar. This will open a blank file in the editor, like this:

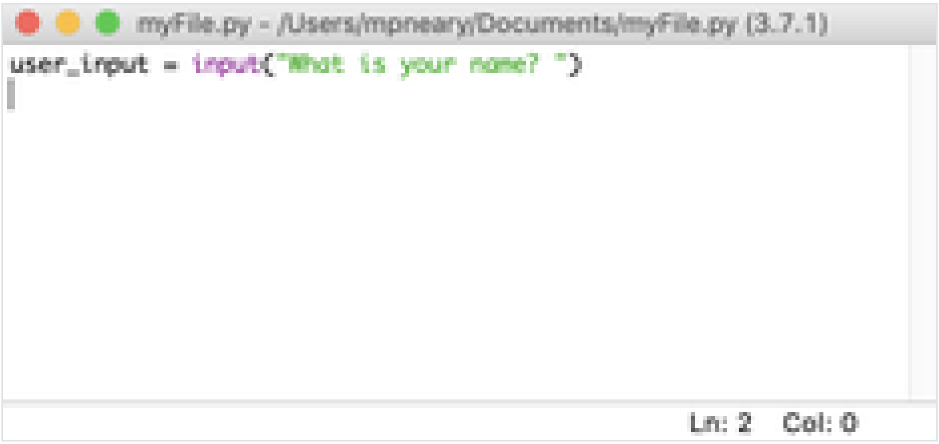
From this window, you can write a brand new Python file. You can also open an existing Python file by selecting *File* → *Open...* in the menu bar. This will bring up your operating system's file browser. Then, you can find the Python file you want to open.

If you're interested in reading the source code for a Python module, then you can select *File* → *Path Browser*. This will let you view the modules that Python IDLE can see. When you double click on one, the file editor will open up and you'll be able to read it.

The content of this window will be the same as the paths that are returned when you call `sys.path`. If you know the name of a specific module you want to view, then you can select *File* → *Module Browser* and type in the name of the module in the box that appears.

Editing a File

Once you've opened a file in Python IDLE, you can then make changes to it. When you're ready to edit a file, you'll see something like this:



```
user_input = input("What is your name? ")
```

The contents of your file are displayed in the open window. The bar along the top of the window contains three pieces of important information:

1. **The name** of the file that you're editing
2. **The full path** to the folder where you can find this file on your computer
3. **The version** of Python that IDLE is using

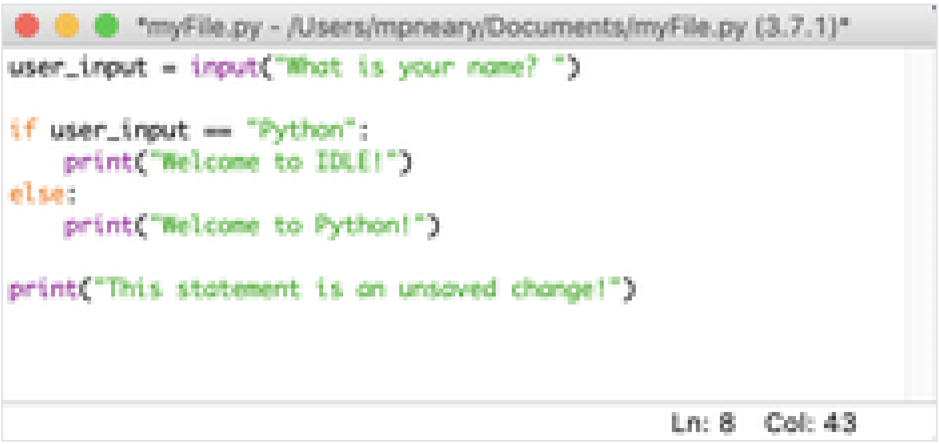
In the image above, you're editing the file `myFile.py`, which is located in the `Documents` folder. The Python version is 3.7.1, which you can see in parentheses.

There are also two numbers in the bottom right corner of the window:

1. **Ln:** shows the line number that your cursor is on.
2. **Col:** shows the column number that your cursor is on.

It's useful to see these numbers so that you can find errors more quickly. They also help you make sure that you're staying within a certain line width.

There are a few visual cues in this window that will help you remember to save your work. If you look closely, then you'll see that Python IDLE uses asterisks to let you know that your file has unsaved changes:



```
user_input = input("What is your name? ")

if user_input == "Python":
    print("Welcome to IDLE!")
else:
    print("Welcome to Python!")

print("This statement is an unsaved change!")
```

The file name shown in the top of the IDLE window is surrounded by asterisks. This means that there are unsaved changes in your editor. You can save these changes with your system's standard keyboard shortcut, or you can select *File → Save* from the menu bar. Make sure that you save your file with the `.py` extension so that syntax highlighting will be enabled.

Executing a File

When you want to execute a file that you've created in IDLE, you should first make sure that it's saved. Remember, you can see if your file is properly saved by looking for asterisks around the filename at the top of the file editor window. Don't worry if you forget, though! Python IDLE will remind you to save whenever you attempt to execute an unsaved file.

To execute a file in IDLE, simply press the F5 key on your keyboard. You can also select *Run → Run Module* from the menu bar. Either option will restart the Python interpreter and then run the code that you've written with a fresh interpreter. The process is the same as when you run `python3 -i [filename]` in your terminal.

When your code is done executing, the interpreter will know everything about your code, including any global variables, functions, and classes. This makes Python IDLE a great place to inspect your data if something goes wrong. If you ever need to interrupt the execution of your program, then you can press `Ctrl` + `C` in the interpreter that's running your code.

How to Improve Your Workflow

Now that you've seen how to write, edit, and execute files in Python IDLE, it's time to speed up your workflow! The Python IDLE editor offers a few features that you'll see in most professional IDEs to help you code faster. These features include automatic indentation, code completion and call tips, and code context.

Automatic Indentation

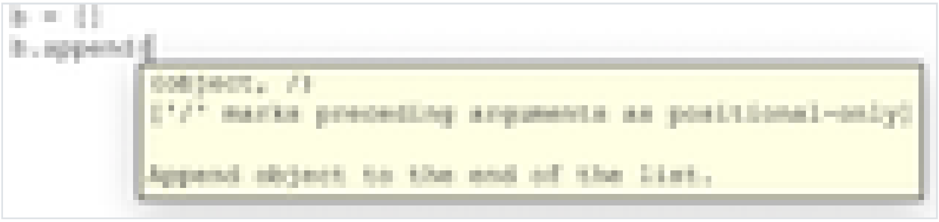
IDLE will automatically indent your code when it needs to start a new block. This usually happens after you type a colon (`:`). When you hit the enter key after the colon, your cursor will automatically move over a certain number of spaces and begin a new code block.

You can configure how many spaces the cursor will move in the settings, but the default is the standard four spaces. The developers of Python agreed on a standard style for well-written Python code, and this includes rules on indentation, whitespace, and more. This standard style was formalized and is now known as **PEP 8**. To learn more about it, check out [How to Write Beautiful Python Code With PEP 8](#).

Code Completion and Call Tips

When you’re writing code for a large project or a complicated problem, you can spend a lot of time just typing out all of the code you need. **Code completion** helps you save typing time by trying to finish your code for you. Python IDLE has basic code completion functionality. It can only autocomplete the names of functions and classes. To use autocompletion in the editor, just press the tab key after a sequence of text.

Python IDLE will also provide call tips. A **call tip** is like a hint for a certain part of your code to help you remember what that element needs. After you type the left parenthesis to begin a function call, a call tip will appear if you don’t type anything for a few seconds. For example, if you can’t quite remember how to append to a list, then you can pause after the opening parenthesis to bring up the call tip:



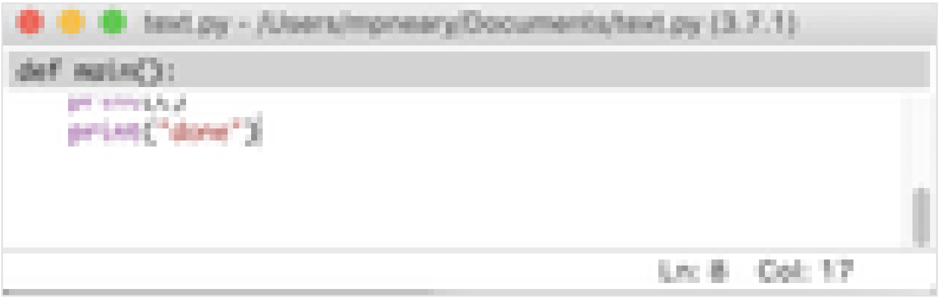
```
b = []
b.append(
        (object, /)
        ['/' marks preceding arguments as positional-only]

        Append object to the end of the list.
```

The call tip will display as a popup note, reminding you how to append to a list. Call tips like these provide useful information as you’re writing code.

Code Context

The **code context** functionality is a neat feature of the Python IDLE file editor. It will show you the scope of a function, class, loop, or other construct. This is particularly useful when you're scrolling through a lengthy file and need to keep track of where you are while reviewing code in the editor.

To turn it on, select *Options* → *Code Context* in the menu bar. You'll see a gray bar appear at the top of the editor window:



test.py - /Users/mgreary/Documents/test.py (3.7.1)

```
def main():
    print(1)
    print("done")
```

Ln: 8   Col: 17

As you scroll down through your code, the **context** that contains each line of code will stay inside of this gray bar. This means that the `print()` functions you see in the image above are a part of a main function. When you reach a line that's outside the scope of this function, the bar will disappear.

How to Debug in IDLE

A **bug** is an unexpected problem in your program. They can appear in many forms, and some are more difficult to fix than others. Some bugs are tricky enough that you won't be able to catch them by just reading through your program. Luckily, Python IDLE provides some basic tools that will help you debug your programs with ease!

Interpreter DEBUG Mode

If you want to run your code with the built-in debugger, then you'll need to turn this feature on. To do so, select *Debug → Debugger* from the Python IDLE menu bar. In the interpreter, you should see `[DEBUG ON]` appear just before the prompt (>>>), which means the interpreter is ready and waiting.

When you execute your Python file, the debugger window will appear:

In this window, you can inspect the values of your local and global variables as your code executes. This gives you insight into how your data is being manipulated as your code runs.

You can also click the following buttons to move through your code:

- **Go:** Press this to advance execution to the next breakpoint. You'll learn about these in the next section.
- **Step:** Press this to execute the current line and go to the next one.
- **Over:** If the current line of code contains a function call, then press this to step *over* that function. In other words, execute that function and go to the next line, but don't pause while executing the function (unless there is a breakpoint).
- **Out:** If the current line of code is in a function, then press this to step *out* of this function. In other words, continue the execution of this function until you return from it.

Be careful, because there is no reverse button! You can only step forward in time through your program's execution.

You'll also see four checkboxes in the debug window:

1. **Globals:** your program's global information

2. **Locals:** your program’s local information during execution
3. **Stack:** the functions that run during execution
4. **Source:** your file in the IDLE editor

When you select one of these, you’ll see the relevant information in your debug window.

Breakpoints

A **breakpoint** is a line of code that you’ve identified as a place where the interpreter should pause while running your code. They will only work when *DEBUG* mode is turned on, so make sure that you’ve done that first.

To set a breakpoint, right-click on the line of code that you wish to pause. This will highlight the line of code in yellow as a visual indication of a set breakpoint. You can set as many breakpoints in your code as you like. To undo a breakpoint, right-click the same line again and select *Clear Breakpoint*.

Once you’ve set your breakpoints and turned on *DEBUG* mode, you can run your code as you would normally. The debugger window will pop up, and you can start stepping through your code manually.

Errors and Exceptions

When you see an error reported to you in the interpreter, Python IDLE lets you jump right to the offending file or line from the menu bar. All you have to do is highlight the reported line number or file name with your cursor and select *Debug → Go to file/line*from the menu bar. This is will open up the offending file and take you to the line that contains the error. This feature works regardless of whether or not *DEBUG* mode is turned on.

Python IDLE also provides a tool called a **stack viewer**. You can access it under the *Debug* option in the menu bar. This tool will show you the traceback of an error as it appears on the stack of the last error or exception that Python IDLE encountered while running your code. When an unexpected or interesting error occurs, you might find it helpful to take a look at the stack. Otherwise, this feature can be difficult to parse and likely won't be useful to you unless you're writing very complicated code.

How to Customize Python IDLE

There are many ways that you can give Python IDLE a visual style that suits you. The default look and feel is based on the colors in the Python logo. If you don't like how anything looks, then you can almost always change it.

To access the customization window, select *Options → Configure IDLE* from the menu bar. To preview the result of a change you want to make, press *Apply*. When you're done customizing Python IDLE, press *OK* to save all of your changes. If you don't want to save your changes, then simply press *Cancel*.

There are 5 areas of Python IDLE that you can customize:

1. Fonts/Tabs
2. Highlights
3. Keys
4. General
5. Extensions

Let's take a look at each of them now.

Fonts/Tabs

The first tab allows you to change things like font color, font size, and font style. You can change the font to almost any style you like, depending on what's available for your operating system. The font settings window looks like this:

You can use the scrolling window to select which font you prefer. (I recommend you select a fixed-width font like Courier New.) Pick a font size that's large enough for you to see well. You can also click the checkbox next to *Bold* to toggle whether or not all text appears in bold.

This window will also let you change how many spaces are used for each indentation level. By default, this will be set to the PEP 8 standard of four spaces. You can change this to make the width of your code more or less spread out to your liking.

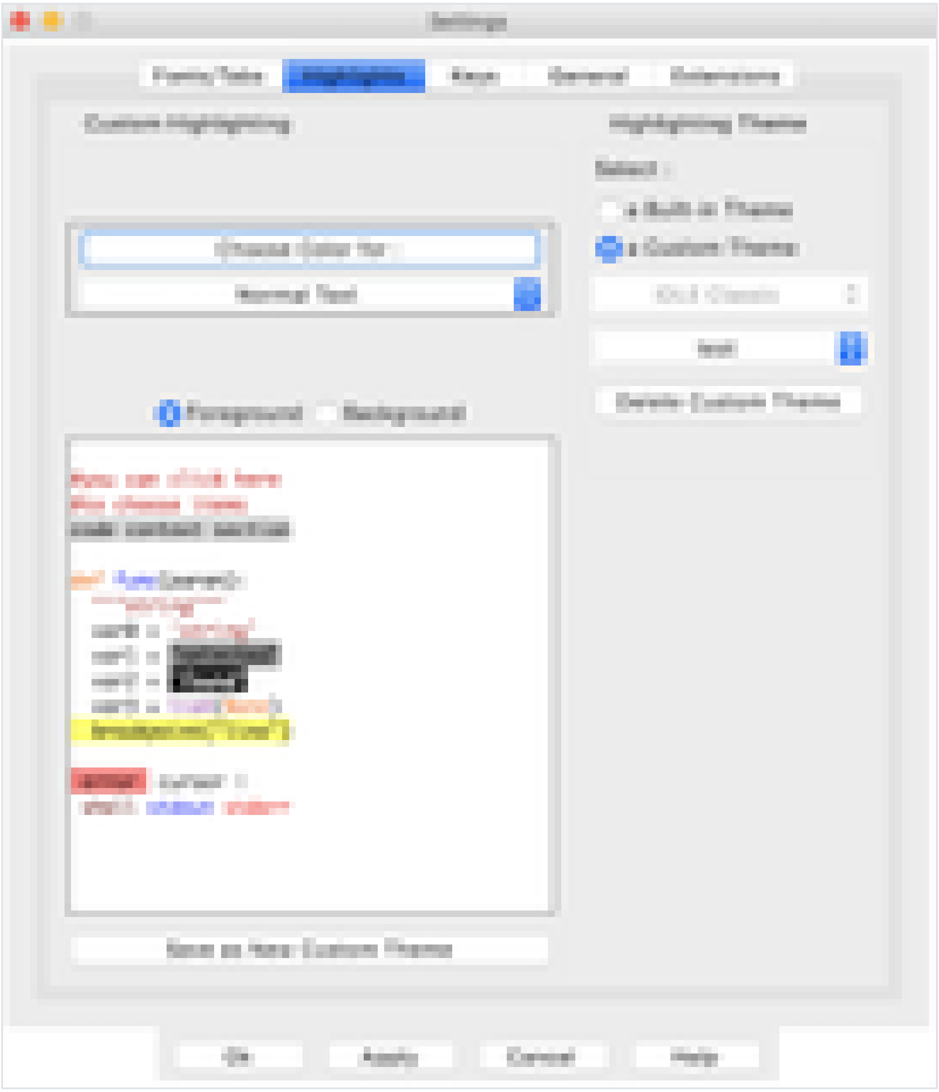
Highlights

The second customization tab will let you change highlights. **Syntax highlighting** is an important feature of any IDE that highlights the syntax of the language that you’re working in. This helps you visually distinguish between the different Python constructs and the data used in your code.

Python IDLE allows you to fully customize the appearance of your Python code. It comes pre-installed with three different highlight themes:

1. IDLE Day
2. IDLE Night
3. IDLE New

You can select from these pre-installed themes or create your own custom theme right in this window:

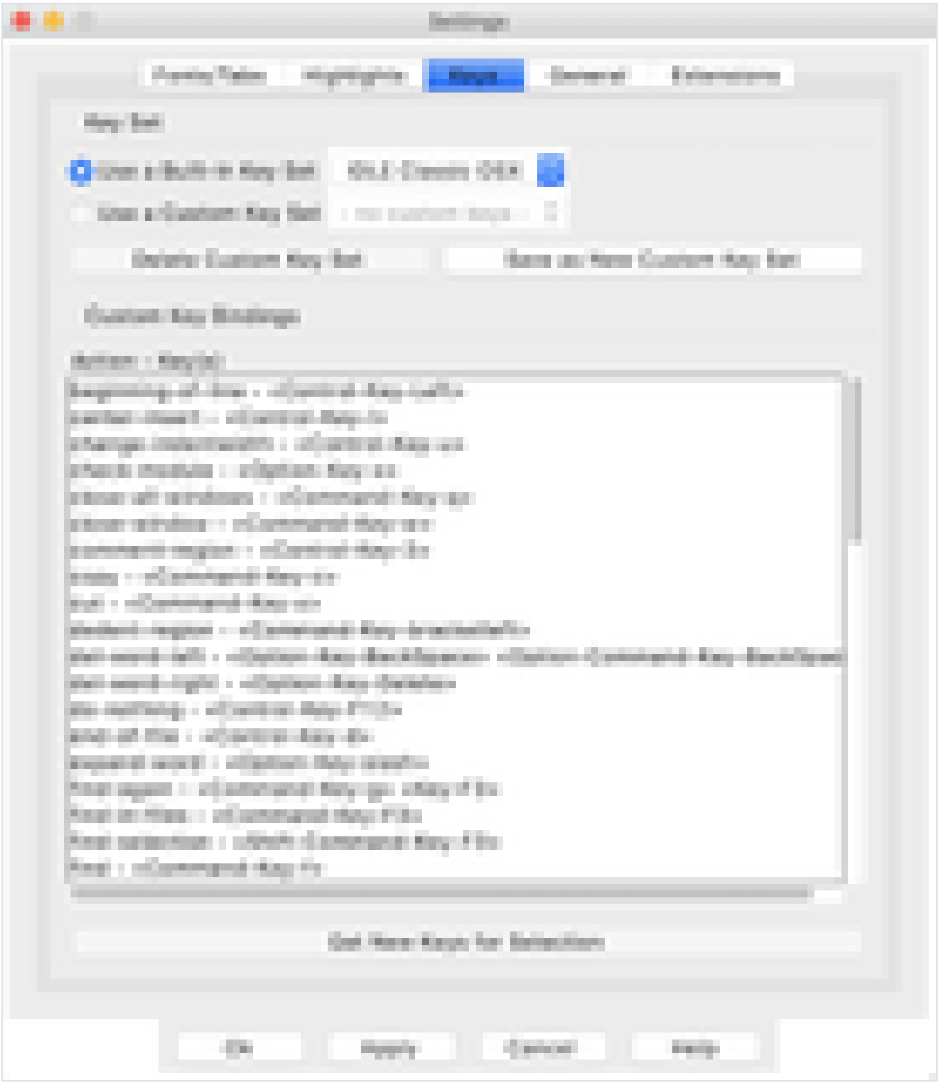


Unfortunately, IDLE does not allow you to install custom themes from a file. You have to create customs theme from this window. To do so, you can simply start changing the colors for different items. Select an item, and then press *Choose color for*. You'll be brought to a color picker, where you can select the exact color that you want to use.

You’ll then be prompted to save this theme as a new custom theme, and you can enter a name of your choosing. You can then continue changing the colors of different items if you’d like. Remember to press *Apply* to see your changes in action!

Keys

The third customization tab lets you map different key presses to actions, also known as **keyboard shortcuts**. These are a vital component of your productivity whenever you use an IDE. You can either come up with your own keyboard shortcuts, or you can use the ones that come with IDLE. The pre-installed shortcuts are a good place to start:



Key Set

[✓ Use a Built-in Key Set] IDLE Classic OSX

Use a Custom Key Set - no custom keys -

Custom Key Bindings

Action - Key(s)

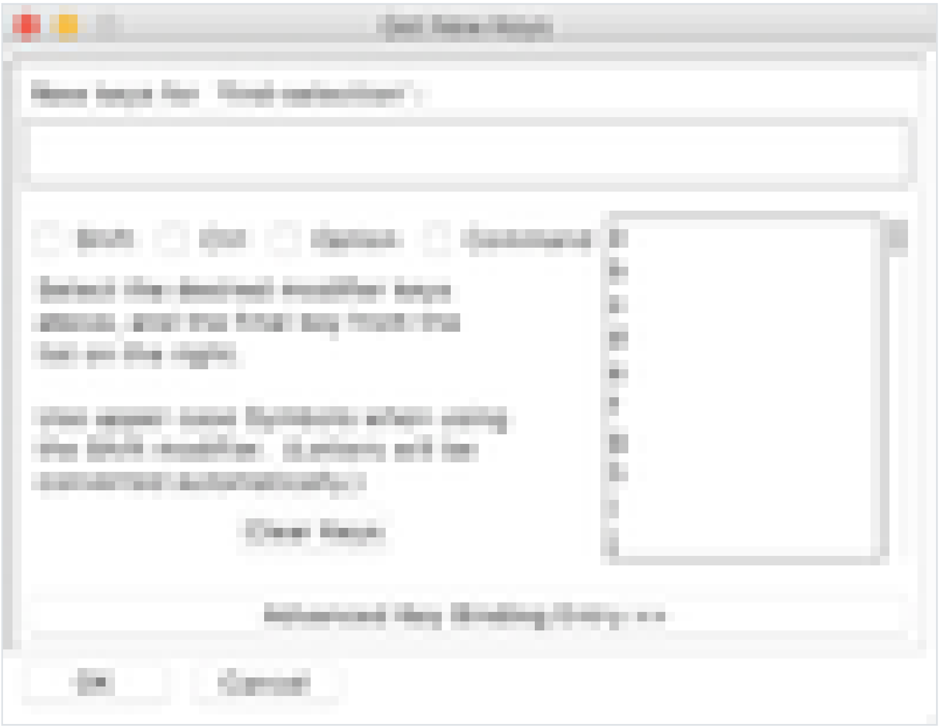
```
beginning-of-line - <Control-Key-Left>
center-insert - <Control-Key-l>
change-indentwidth - <Control-Key-u>
check-module - <Option-Key-x>
close-all-windows - <Command-Key-q>
close-window - <Command-Key-w>
comment-region - <Control-Key-3>
copy - <Command-Key-c>
cut - <Command-Key-x>
dedent-region - <Command-Key-bracketleft>
del-word-left - <Option-Key-BackSpace> <Option-Command-Key-BackSpac
del-word-right - <Option-Key-Delete>
do-nothing - <Control-Key-F12>
end-of-file - <Control-Key-d>
expand-word - <Option-Key-slash>
find-again - <Command-Key-g> <Key-F3>
find-in-files - <Command-Key-F3>
find-selection - <Shift-Command-Key-F3>
find - <Command-Key-f>
```

The keyboard shortcuts are listed in alphabetical order by action. They're listed in the format *Action - Shortcut*, where *Action* is what will happen when you press the key combination in *Shortcut*. If you want to use a built-in key set, then select a mapping that matches your operating system. Pay close attention to the different keys and make sure your keyboard has them!

Creating Your Own Shortcuts

The customization of the keyboard shortcuts is very similar to the customization of syntax highlighting colors. Unfortunately, IDLE does not allow you to install custom keyboard shortcuts from a file. You must create a custom set of shortcuts from the *Keys*tab.

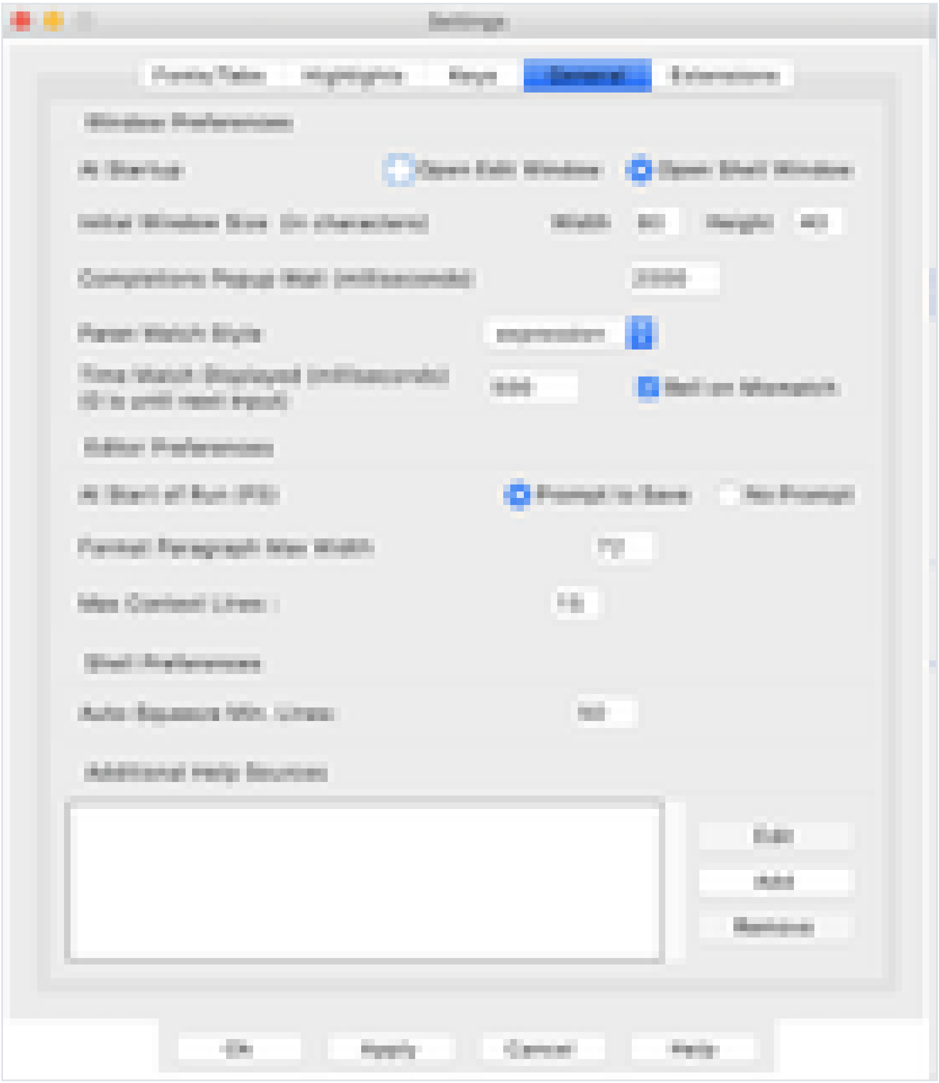
Select one pair from the list and press *Get New Keys for Selection*. A new window will pop up:



Here, you can use the checkboxes and scrolling menu to select the combination of keys that you want to use for this shortcut. You can select *Advanced Key Binding Entry >>* to manually type in a command. Note that this cannot pick up the keys you press. You have to literally type in the command as you see it displayed to you in the list of shortcuts.

General

The fourth tab of the customization window is a place for small, general changes. The general settings tab looks like this:



Window Preferences

At Startup ( ) Open Edit Window [✓ Open Shell Window]

Initial Window Size (in characters) Width 80 Height 40

Completions Popup Wait (milliseconds) 2000

Paren Match Style expression

Time Match Displayed (milliseconds) (0 is until next input) 500 [✓ Bell on Mismatch]

Editor Preferences

At Start of Run (F5) [✓ Prompt to Save] ( ) No Prompt

Format Paragraph Max Width 72

Max Context Lines : 15

Shell Preferences

Auto-Squeeze Min. Lines: 50

Additional Help Sources

Here, you can customize things like the window size and whether the shell or the file editor opens first when you start Python IDLE. Most of the things in this window are not that exciting to change, so you probably won’t need to fiddle with them much.

Extensions

The fifth tab of the customization window lets you add extensions to Python IDLE. Extensions allow you to add new, awesome features to the editor and the interpreter window. You can download them from the internet and install them to right into Python IDLE.

To view what extensions are installed, select *Options* → *Configure IDLE -> Extensions*. There are many extensions available on the internet for you to read more about. Find the ones you like and add them to Python IDLE!

Conclusion

In this tutorial, you've learned all the basics of using **IDLE** to write Python programs. You know what Python IDLE is and how you can use it to interact with Python directly. You've also learned how to work with Python files and customize Python IDLE to your liking.

You've learned how to:

- Work with the Python IDLE shell
- Use Python IDLE as a file editor
- Improve your workflow with features to help you code faster
- Debug your code and view errors and exceptions

- Customize Python IDLE to your liking