Chapter No. 3
"Day-to-day Coding Tools"
In this package, you will find:
A Biography of the authors of the book
A preview chapter from the book, Chapter NO.3 "Day-to-day Coding Tools"
A synopsis of the book’s content
Information on where to buy this book

About the Authors

Martino Sabia is a curious-minded developer with close to 30 years of coding experience. Throughout his years of working with different platforms and languages, he has always kept his mind fresh while finding creative ways of using different technologies. Based in Italy, Martino has spent his career in various start-up companies, working in numerous roles from junior developer to software architect. Now he is the Project Lead for Deltatre; he works on consumer-facing, heavy-traffic websites and media-streaming platforms in the sports industry.

Cathy Wang is an experienced designer who specializes in service design and experience strategy. She has worked on many cross-channel projects and served as a design lead for enterprise services around the globe in fields ranging from Telecom to public sectors. Cathy has worked for world-class design agencies to help bring visions to life. In her free time, she builds web projects and apps. She is infinitely curious about new technologies and the experiences they can bring.

For More Information:
Python Tools for Visual Studio

Like many other developers, Python developers have always had to find ways to manage the development workflow between different tools. Most of the time, this happens without using a comprehensive guide that is available in a complete IDE which is specifically designed for Python development.

The rare, exceptional IDEs that offer complete guides are often expensive and don't provide hands-on steps to help speed up the development process.

Visual Studio, as a matured and well-developed tool over the last few decades, has dominated the market of compiled languages and languages that are strictly oriented toward Windows and .NET. Packed with handy tools and functionalities to speed up and facilitate the workflow of developers, it helps users to render repetitive tasks, manage projects, and provide a detailed outlook into the structure of a project. However, most importantly, it helps users gain a clear view into the inner structure of the code.

In the last few years, Microsoft has started exploring how to integrate new languages into Visual Studio; as a result, Python Tools for Visual Studio (PTVS) was developed. It's a well-developed tool that is already on its second release and is commonly used by professional developers as their new IDE of choice for Python projects.

PTVS has everything that a Python developer can dream of: consistent project files management, interactive debugging and code completion features with the rock solid Microsoft IntelliSense technology, project templates, a first-class Django integration package, virtual environment management right in the IDE for REPL, and a native code-based IDE that loads and reacts fast.

This book will focus more on the integration of Python in Visual Studio than the language itself. It will try to delve into the power offered by the tool and venture into the feasibility of its day-to-day usage for a developer. We will show real examples of how to use PTVS with Django and how to deal with occasional difficulties when it comes to integrating well-known libraries into a Python project on Microsoft Windows.

What This Book Covers

Chapter 1, Introduction to PTVS, provides a high-level overview of PTVS and the interaction between Visual Studio and a Python interpreter.

Chapter 2, Python Tools in Visual Studio, provides an in-depth analysis of the tools, type checking, inner functionalities, and automatisms (IntelliSense and REPL) of PTVS.

Chapter 3, Day-to-day Coding Tools, talks about browsing through the code and the flexible setting of Python environments. It also talks about refactoring and the debugging process.

For More Information:
Chapter 4, *Django in PTVS*, shows how to harness the powerful Visual Studio IDE and tooling to speed up Django development.

Chapter 5, *Advanced Django in PTVS*, provides an in-depth look at remote task management and schema migrations using the third-party Python libraries Fabric and South.

Chapter 6, *IPython and IronPython*, provides an overview of the IPython library and how it's integrated in Visual Studio. It also provides an introduction to IronPython and its integration with the .NET framework.

For More Information:
In this chapter, we will go through the coding tools that are essential during a normal day of work for a Python programmer in Visual Studio.

First we will analyze how to handle projects and solutions in Visual Studio, and then we will go through the refactoring of functionalities. Finally, we will go through the debugger functions that are available.

**Project handling**

One of the most important and useful features of Visual Studio is the solution and project handling. Since the whole workflow is integrated into the IDE, the developer does not have the burden of dealing with files, working paths, and libraries. All of these can be managed directly in the IDE with the powerful Visual Studio user interface.

Before we dig into the tools in detail, we will first take a look at the Visual Studio lingo relating to project handling. The two main concepts used in this chapter are the solution and the project.

**Solution**

A solution is essentially a container of projects that are bundled together to cover a unique scope. The projects can be referenced to each other and they can be of different types. For example, in a solution, you can mix a Python project with a C++ project while referencing the output of the project in the Python solution to use it as an external library. A solution also provides a way to group the whole code base of work in a single file/folder structure. You can then insert and manage the grouping in the versioning tool of your choice to share it quickly. A Visual Studio solution is also capable of maintaining shared configurations for the inner projects, while handing different commands and operations during different events (i.e. during the build of the solution).

For More Information:

Project
A project is the classical definition of a bunch of files written in the same language and which covers a single scope. The types of projects can range from a website to a library or a console application.

Visual Studio projects for Python contain the environment definition: where to target the code, the references to external libraries, and the search paths that the compiler has to search in the libraries. The last one is particularly important since PTVS does not use the computer’s PythonPath environment variable.

The deliberate and useful feature of ignoring system-wide settings allows you to reference different libraries in different projects for different Python versions. Furthermore, the dependency list in the code brings the added bonus of an easier debugging process and also provides an easy setup of a new development environment on other computers.

PTVS offers preconfigured Python projects called project templates, as we have seen in Chapter 1, Introduction to PTVS, which take care of creating the right project structure so that the developer can focus on the code.

Let's take a look at the actual tools that will handle solutions and projects. The most important and powerful one is the Solution Explorer window tool. This tool gives a complete view of the solution composition and the files and configurations available in each project.
To add something or to perform actions on a Solution or a Project, select the Solution or Project node in the tree-view.

By now, you should already know how to navigate through the code using this tool, as we learned in Chapter 2, *Python Tools in Visual Studio*. The Solution Explorer window tool also provides file handling capabilities, giving us the ability to add or remove files directly in the project structure. Just select a folder item in the project structure to insert a new file. To add something in the project root, select the project item in Solution Explorer, right-click to open the contextual menu, and go into the Add submenu as shown in the following screenshot:

For More Information:  
Day-to-day Coding Tools

Here, you can choose to add either **New Item**, **Existing Item**, or **New Folder**. If you choose to add **New Item**, the **Add New Item** window will show up as follows:

From here, you can create a new item from the various types that are available in the project.

If instead you want to add an existing file to the selected folder, select the **Add existing item** option from the contextual add menu. This will open a standard Windows browse file window; from here, you can navigate through the filesystem and select the file that you want to add to the selected folder. A copy of that file will be added to the folder.

To delete a file, just select the item in the project list and select the **Delete** menu item from the contextual menu.

For More Information:
The file contextual menu in the Solution Explorer window offers lots of functions on file, like "Delete", "Rename" and others.

Now that we have more confidence in **Solution Explorer**, let's dig a little deeper into the Python-specific options that **Solution Explorer** offers. In the following sections, you will learn more about the configuration of a Python project: the environment, the references, and the search path.

### Specifying Python environments

It's possible to specifically define a Python environment version for a project instead of using the default Python version installed on the machine. This is particularly important for projects that we work on with other developers. By default, Visual Studio uses the default Python version installed on the machine when starting a new project. To link a project to a given Python version (environment), right-click on the **Python Environment** item in **Solution Explorer** to see the contextual menu. This is shown in the following screenshot:

![The Python Environments contextual menu in the Solution Explorer window](image-url)
This contextual menu provides various functionalities such as adding or removing a Python environment and linking the project to either a virtual Python environment or an existing one. The last two options are very useful when you need to have a project running in a completely isolated environment space on the machine. The project can then be run with all of its dependencies and libraries in an isolated place, without interfering with the existing Python installations and Python path configurations on the machine.

Creating a virtual environment in Visual Studio is straightforward. Click on either Add Virtual Environment or Add Existing Virtual Environment and follow a few steps to complete the setup.

As an example, we will create a virtual environment for our project. Clicking on Add Virtual Environment will show the following modal window:

![Add Virtual Environment Modal Window]

For More Information:
You can define the name of the virtual environment and the targeted Python version in this window. Once the Create button is pressed, Visual Studio will create the virtual environment. As a nifty bonus, if the necessary Python libraries are not installed on the machine—essentially pip, setuptools, and virtualEnv—PTVS will take care of this by downloading and installing them. Like other generic Python packages, they will be installed in the system-defined site-packages folder.

To link a project to a given Python version, just click on the Add Python Environment option in the contextual menu and the following helper box will show up:

![Add/Remove Python Environments](image)

Go online and help me find another one

OK  Cancel

For More Information:
This helper box shows a list of the Python versions installed on the machine and which are available for you to choose. Once a Python version is selected, the reference will show up in Solution Explorer.

The References item elements in a project provide the ability to tightly link a library in your project or reference to packages compiled in the .pyd files.

Right-click on the Reference option and then click on Add Reference to bring up the following helper window:
We will focus on the **Projects** tab, which shows all the other projects in your solution. If you wish to use another library project to handle a subscope of the application, select the desired project and click on **OK**. This creates a reference in your project. You can find the list of references under the reference option in the **Solution Explorer** window:

![Solution Explorer](image)

**Defining Search Paths**

The **Search Paths** functionality basically tells Visual Studio where to search for additional libraries that will be used in the project. You can reference a folder in the system or a .zip file that contains the libraries:

![Search Path Properties](image)

The Search Path contextual menu, with the "Add Folder" and "Add Zip Archive" options

---

For More Information:  
Once the folder of the .zip file is selected, you can find the libraries in the window:

![Solution Explorer](image)

### Refactoring

Refactoring is one of the biggest advancements in modern IDEs. It significantly cuts down on time and reduces the margin of error by the way in which it handles changes in the code and automated operations. Visual Studio comes with great out-of-the-box refactoring functionalities such as renaming and the creating method from a selected piece of code.

The renaming functionality can really help with potential errors in code, such as when changing the name of an element. There might be instances in the codebase where the old name is still used. Let’s have a look at the following code:

```python
class foo:
    
    """
    Documentation of the class.
    It can be multileine and contain any amount of text
    """
    @classmethod
    def bar(self, first=0, second=0):
        """This is the documentation for the method"
        return first + second

print(foo.bar())
```

For More Information:  
In this code, there's a class, foo, that has a method called bar. If bar is renamed, it will create an error by referencing to a nonexistent method.

Visual Studio's refactoring functionality helps the renaming process by taking all the references of the element into account. Select the element that you wish to demand and then access the refactoring function in the code contextual menu by right-clicking on it:

Access refactoring function by right clicking on the element

Select the Rename function to start the two-step process:

For More Information:
You are prompted to enter a new name for the element. There's also a checkbox that permits you to preview the references of the element to be renamed. When the checkbox is unchecked, all the references that are found will be renamed; you will not be able to preview which references are going to be renamed.

In a situation where you wish to preview the reference change, a preview window will be shown. As shown in the preceding screenshot, in this window all references of the old element name can be found at the top panel and the code preview will be at the bottom. All the files of the project will be analyzed to refactor the element correctly. A checkbox near each reference provides the option to activate the refactoring of that reference. Clicking on the **Apply** button will rename the element and all the selected references.
The other refactoring function is **Extract Method**. This comes in handy when you wish to reuse a piece of code somewhere else as a function or a method. Visual Studio can generate it as a function/method. As an example, refer to the following screenshot:

```python
@classmethod
def doSomething(self, first=0, second=0):
    """This is the documentation for the method""
    return first + second
```

In the code, the highlighted code aims to create a generic method that calculates the sum of two elements. Select the code and then select **Extract Method** from the **Refactor** submenu in the contextual menu. This brings up the **Extract Method** dialog box, which is shown in the following screenshot:

In the **Extract Method** window, you can define the name of the method in the **New method name** field and provide a path for the method to be created in the **Extract to** field. A **Preview** panel also shows the generated code. Click on **OK** to create the new method based on the selected code.
Day-to-day Coding Tools

Debugging

Visual Studio offers a large set of debugging tools; PTVS inherits a lot of them, which helps Python developers to debug code by using step-by-step execution, runtime variable watch capabilities, breakpoints, and the ability to see where the code fails during a debugging session.

The ability to see where the code breaks can significantly speed up the debugging session. In the following screenshot, we can see an example of an untracked exception:

```python
@classmethod
def doSomething(self, first=0):
    """This is the documentation"
    return first + second

@classmethod
def codeWithError(self):
    d = 0
    return 100/d

@classmethod
def average(self, *arguments):
    sum = 0
    for arg in arguments:
        sum += arg
    return sum/len(arguments)

print(myClass.doSomething())
```

An example of error dialog box during debugging a Python application

When you run the code, Visual Studio will stop the execution because it detects the raised exception of a problematic code. It highlights the exact point where the error occurred while also suggesting ways to fix it—even if right now Visual Studio may not suggest useful solutions for the problem.

The debugging process is not only about understanding where exceptions are raised, but also to understand what happens in the code when it is not behaving as expected. This is where step-by-step execution and breakpoints come in handy.
Using breakpoints
A breakpoint is a point that you can define in the code to stop the execution. Visual Studio has made it very simple to set a breakpoint. It allows better visibility of the content of variables and it follows the flow of the code. A breakpoint can be set by clicking in the gutter of the code window, which will bring up a red circle. Select **Insert Breakpoint** in the **Breakpoint** submenu in the code contextual menu.

Once the breakpoint is set, you can see it in the code window as shown in the following screenshot:

Now that the breakpoint is set, if you run the application, Visual Studio will stop its execution precisely at the breakpoint while following the flow of the code:

The IDE puts the caret on the first column of the line of code in which we set the breakpoint. When hovering around the variables in that context, we can see the current value of the variable.
Utilizing watch entries

We can also create a watch entry on a variable in order to see how the value of a variable changes during the program flow. To watch a variable, right-click on it during the debugging process and click on Add Watch in the contextual menu. The variable will be added into the Watch window as shown in the following screenshot:

Besides the watch variable, it is also possible to see all the variables in the current scope from the Locals tab:

Once a breakpoint has been hit, it's possible to use one of the following three functions to move on in the program flow: Step Into, Step Over, and Step Out. These functions are accessible through the Debug menu or the buttons available in the toolbar. Alternatively, you can also use Run to Cursor (Ctrl + F10) to run through the program until you reach where the cursor is:
• **Step Into**: This executes the next statement and stops. If the next statement is a call to a function, the debugger will stop at the first line of the function being called entering the function.

• **Step Over**: This executes the next statement. However, if the next statement is a function, calling it will not go into it. It's useful when you are not willing to follow the entire program flow of the function.

• **Step Out**: This executes the code until the end of the current function. It's useful when you do not wish to go through the entire program flow of the current function.

If you wish to just continue the execution of the program flow without going into a single line of code at the time, just press the **Continue** button in the toolbar or F5. If there are other breakpoints in your code, the execution will continue through all of them until the last one.

**Summary**

In this chapter, we introduced the tools for day-to-day coding. You are now familiar with browsing through the code with **Solution Explorer** and the flexible setting of Python environments. You also learned about the more efficient refactoring and debugging process and that setting up breakpoints and watching entries helps you trace exactly where the code breaks.

In the next chapter, we will explore how to harness the powerful Visual Studio IDE and the tools available to speed up Django development.

---

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.