code libraries called **packages** that can be used for engineering computing. We will use a few key packages in this book, and there are many more available on the Internet, especially at https://engineering-computing.ricopic.one (Python Community 2024b).

There are several classes of engineering analysis performed with engineering computing. The following list captures the majority of problems, but it is far from exhaustive.

Numerical Analysis Many engineering problems can be approached by performing numerical calculations. These can be challenging or even intractable to perform manually when the problem requires many such calculations. **Numerical analysis** use systematic procedures called **algorithms** to perform the calculations with a computer. These techniques use the computer to perform, store, and organize these calculations. This class of problems, sometimes called **simulation**, comprise the majority of engineering computing problems.

Symbolic Analysis **Symbolic analysis**, sometimes called "analytic" as opposed to "numerical," is closely related to mathematics. Mathematical variables can be directly manipulated via algebraic methods (including those of calculus). Computer programs that treat these variables symbolically are called **computer algebra systems (CASs)**. Although these systems can be somewhat cumbersome, for complex problems they provide distinct advantages.

Graphical Analysis Visualization techniques are an important aspect of engineering analysis. **Graphics**—often graphs, plots, and charts—can be generated by programs much more quickly and accurately than they can be created manually. The result of an engineering computing program is often a graphic.

In this book, we will introduce all three classes of analysis.

1.2 The Development System

89 D1

In general, a computer **development system** is one that is used to write, execute, debug, and deploy computer programs. Our development system is comprised of the following components:

- A personal computer (PC) (e.g., one running the Windows, macOS, or Linux operating system)
- The Anaconda distribution of the Python 3 software
- The **Spyder integrated development environment (IDE)**

An IDE is a software application in which a programmer can write, execute, and debug their programs.

On your PC, set up your development system with the following steps:

1. Download the Anaconda distribution of the Python software from the following URL:
   www.anaconda.com/download
   Open the installer and follow the instructions for installation.
2. Download and install the Spyder IDE from the following URL:
   www.spyder-ide.org
   Open the installer and follow the instructions for installation.

1.2.1 The Anaconda Distribution of Python

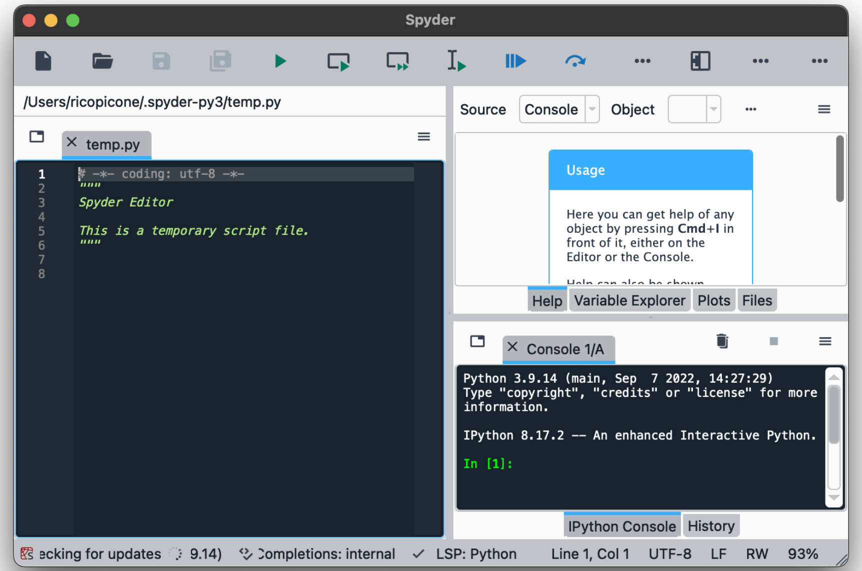
Anaconda provides a way of managing multiple **Python environments**; a Python environment is a specific version of Python with a set of packages. For a given project, it is best practice to maintain a separate environment; this allows us to specify a Python version and set of packages required to run the programs in the project. Anaconda provides a framework in which we can create an environment, called a **conda environment**.

We will use the default `base` environment. To create your own environments or add packages, see the instructions in the Anaconda documentation:
https://engineering-computing.ricopic.one/0e.

0E

1.2.2 Hello World and the Spyder IDE

When it is first loaded, the Spyder IDE looks something like what is shown in figure 1.1.



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

```
Python 3.9.14 (main, Sep  7 2022, 14:27:29)
Type "copyright", "credits" or "license" for more
information.

IPython 8.17.2 -- An enhanced Interactive Python.

In [1]:
```

Figure 1.1. The Spyder IDE when it first loads

The left pane is the code editor. It has a default **Python file**, which conventionally has extension `.py`, already queued up. Create a new Python file by selecting the menu item File ⟩ New file… . Save this file (File ⟩ Save) as `hello_world.py` in a dedicated directory.[1]

The `hello_world.py` file already contains a triple-quoted string with basic information about the file. Below the ending quotes, add the following statement:

```
print("Hello World!")
```

Save the file and run it with the menu selection Run ⟩ Run or the key F5.

The **console** pane on the lower right shows the result of the execution of the file, which is the output

```
Hello World!
```

Now let's edit the program as follows:

1. In programming, file names should not include spaces, periods (other than for the extension), or most special characters. As a word separator, the hypen - is usually fine, but the underscore _ is topically safer. For Python files, the underscore is preferable.

```
greeting = "Hello World!"
print(greeting)
```

This should yield the same result in the console. In the upper-right pane, select the Variable Explorer tab. This shows variable names, types, and values in the current **kernel**. A kernel is a computing process that runs programs. In most environments, when a program runs, a kernel is created at the start and destroyed at the end of execution. However, Spyder maintains the same kernel between runs. This is convenient for debugging purposes. For instance, we can interact with the program(s) run in the current kernel by entering commands in the console; try entering

```
greeting
```

This will return the value of the variable `greeting`. The console is a convenient place to try out statements as we work on our program. For instance, we may want to append some text to the `greeting` string. In the console, try

```
greeting + "It's a beautiful day"
```

This returns, `Hello World!It's a beautiful day`, which is close but not quite what we wanted. We should add a space character to the beginning of our addendum. So, trying it out in the console allowed us to quickly debug our code.

The persistent kernel can also cause problems. Sometimes we may want to create a new kernel by selecting the menu item Consoles › Restart kernel, which clears all variables and unloads any packages. Similarly, to clear all variables in the kernel, we can execute the console **magic command**

```
%reset
```

We will be asked to confirm, which we can do by entering `y`.

1.2.3 Configuring the Spyder IDE for Anaconda

In section 1.2.2, we used the Python distribution that Spyder has built in. We here configure Spyder to use the Anaconda distribution installed in section 1.2.1. First, we must install the `spyder-kernels` package in the `base` Anaconda environment. On a Windows PC, open the Anaconda Prompt application; on MacOS or Linux, open the Terminal application. To ensure you have activated the `base` environment, enter the following prompt:

```
conda activate base
```

Now install the `spyder-kernels` package with the command

```
conda install spyder-kernels
```

Enter `y` if prompted. After successful installation, `conda list` should display the packages installed in the `base` environment, including `spyder-kernels`. Finally, enter the command

```
which python
```

Copy or record the returned path.

In Spyder, open preferences with Ctrl + , . Navigate to the tab Python Interpreter and check Use the following Python interpreter. Either paste the path copied above in the text field or click the Select file button, then navigate to the path in question, selecting the `python` program. Click OK to complete the configuration.

You may need to restart Spyder for the changes to take effect.

1.3 Basic Elements of a Program

CO

Every programming language has a **syntax**: rules that describe the structure of valid combinations of characters and words in a program. When one first begins writing in a programming language, it is common to generate **syntax errors**, improper combinations of characters and words. Every programming language also has a **semantics**: a meaning associated with a syntactically valid program. A program's semantics describe what a program does.

In Python and in other programming languages, programs are composed of a sequence of smaller elements called **statements**. Statements do something, like perform a calculation or store a value in memory. For instance, `x = 3*5` is a statement that computes a product and stores the result under the variable name `x`. Many statements contain **expressions**, each of which produces a value. For instance, `3*5` in the previous statement is an expression that produces the value `15`.

An expression contains smaller elements called **operands** and **operators**. Common operands include **identifiers**—names like variables, functions, and modules that refer to objects—and **literals**—notations for constant values of a built-in type. For instance, in the previous expression `x` is a variable identifier and `3` and `5` are literals that evaluate to objects of the built-in `int`eger class. The `*` character in the previous expression is the multiplication operator. Python includes operators for arithmetic (e.g., `+`), assignment (e.g., `=`), comparison (e.g., `>`), logic (e.g., `or`), identification (e.g., `is`), membership (e.g., `in`), and other operations.

Example 1.1

Create a Python program that computes the following arithmetic expressions:

$$x = 4069 \cdot 0.002, \quad y = 100/1.5, \text{ and } \quad z = (-3)^2 + 15 - 3.01 \cdot 10.$$

Multiply these together (xyz) and print the product, along with x, y, and z to the console.