1.4 The Development System

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 **VS Code 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 VS Code IDE from the following URL:
   https://code.visualstudio.com
   Open the installer and follow the instructions for installation.

1.4.1 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**.

1.4.2 Configuring VS Code with Anaconda

To configure VS Code to use the Anaconda distribution of Python, follow these steps:

Install the Python and Jupyter extensions After installing VS Code, open the application. Choose the menu item View 〉 Extensions and install the Python extension by Microsoft. This extension provides Python language support, debugging, and additional tools for Python programming. Also install the Jupyter extension, which provides notebook support in VS Code.

Open a working directory Open a directory in which to save your Python programs by selecting the menu item File 〉 Open Folder… . Navigate to a convenient location and create a new directory, such as `engineering_computing`. Select

this directory and click Open. You will save your Python programs in subdirectories of this directory.

Create a conda environment Open a terminal in VS Code by selecting the menu item Terminal > New Terminal. In the terminal, enter the following command to create a new conda environment named `ec`:

```
conda create --name ec python=3.12 jupyter numpy matplotlib sympy
```

The `python=3.12` argument specifies the Python version, and the packages `jupyter`, `numpy`, `matplotlib`, and `sympy` are installed in the environment.

Select the Python interpreter Open the Command Palette by selecting the menu item View > Command Palette... . Type `Python: Select Interpreter` and select the corresponding command that appears. Select the Python interpreter that corresponds to the `ec` environment of Anaconda created in the previous step. This will allow VS Code to use the Anaconda distribution of Python, and this interpreter will be used for all Python programs in the open directory (i.e., you will not need to select the interpreter for each program).

1.4.3 Hello World and VS Code

Create a new Python file by selecting the menu item File > New Text File. Save this file (File > Save) as `hello_world.py` in a dedicated directory.[1]

Add the following statement:

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

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

A new terminal will open in the lower pane of the VS Code window and displayed will be the output

```
Hello World!
```

Now let's edit the program as follows:

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

This should yield the same result in the terminal. Now right-click anywhere in the editor pane and select Run in Interactive Window > Run Current File in Interactive Window. This will open an interactive window that will execute the program and display the output. This is different from the terminal in that it allows us to interact with the program after it has run.

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

In the interactive window, select the Jupyter Variables button. 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, the interactive window maintains the same kernel between runs. This is convenient for debugging purposes.

A prompt appears at the bottom of the interactive window, where we can enter Python commands that are executed in the context of the program that was run. For instance, we can interact with the program(s) run in the current kernel by entering commands in the prompt; 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 clicking the button Restart, which clears all variables and unloads any packages. Similarly, to clear all variables in the kernel, we can execute the **magic command**

```
%reset
```

We will be asked to confirm, which we can do by entering `y`. Note that magic commands like this are only available in the interactive window, and cannot be used in Python programs. The interactive window is a great place to test out code snippets and debug programs. It is, in fact, a form of a **Jupyter notebook**, which is a powerful tool for data analysis and visualization. See appendix A for more information on Jupyter notebooks.

1.5 Basic Elements of a Program

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.

Consider the following program:

```
x = 4096*0.002              # float multiplication
y = 100/1.5                 # float division
z = (-3)**2 + 15 - 3.01*10  # exponent operator **
print(x,y,z)
print(x*y*z)
```

The console should print

```
8.192 66.66666666666667 -6.099999999999998
-3331.413333333333
```