# LO2: Tools and Techniques

## Predefined Functions

The principles of DRY (Don’t Repeat Yourself) and code reuse are made much easier using pre-compiled functions and code stored in libraries. There is no need to re-invent (and test!) the wheel if someone else has already opened a kwik-fit next door.

You can of course, make your own code libraries, in the same way that a third party does… set up a project of the correct type, write your code, compile and distribute it. There are several services that can be used to easily distribute your code and allow for easily getting the newest version, such as NuGet integrated with Visual Studio, or apt for Unix.

Pre-built libraries also a great example of encapsulation… a library with give you objects you can use without concerning yourself about the details of the implementation.

A screenshot of a computer

AI-generated content may be incorrect.

The NuGet browser in Visual Studio incorporates a search engine, update notifications and allows for easy installing and removing of pre-build code libraries.

System libraries provide essential functionality for programming languages that are distributed with a programming language, from low-level features such as file access and providing GUI functionality.

Here is an example code snippet that uses the System.IO library to determine if a file path is valid:

A screen shot of a computer program

AI-generated content may be incorrect.

Here, the System.IO library is providing the Path object, which has a Exists() method. We pass it a string and it returns a Boolean telling us if the string represents an existing file path.

## Screen Templates

Visual development environments allow us to define UI elements using a designer and drag and drop tools as well as through code. This makes designing compelling user interfaces much easier. They can also provide templates to make the task even easier, where similarly to code libraries, pre-defined UI forms can simply be dropped into an existing application and used. Of, course, was can also define our own templates and distribute them across an organisation. This code reuse leads to a much more integrated feel to an organisation’s applications and of course, higher quality code.

A screenshot of a computer

AI-generated content may be incorrect.

By default, Visual Studio provides a ‘Splash Screen’ form template.

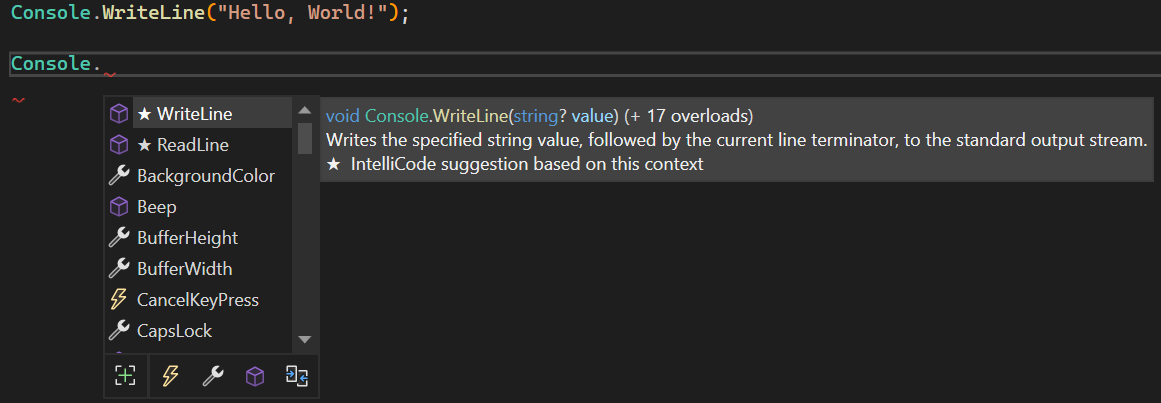
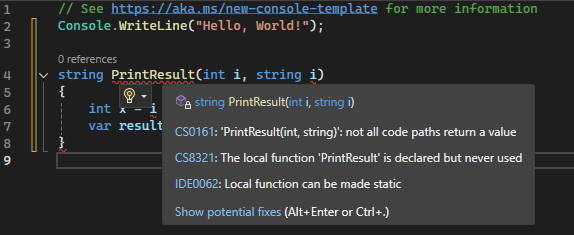
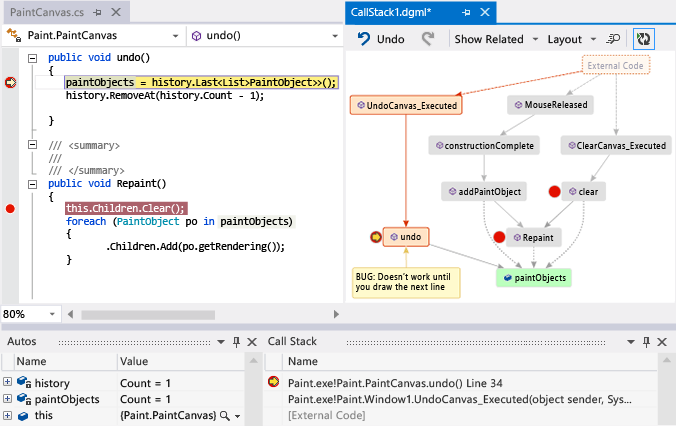
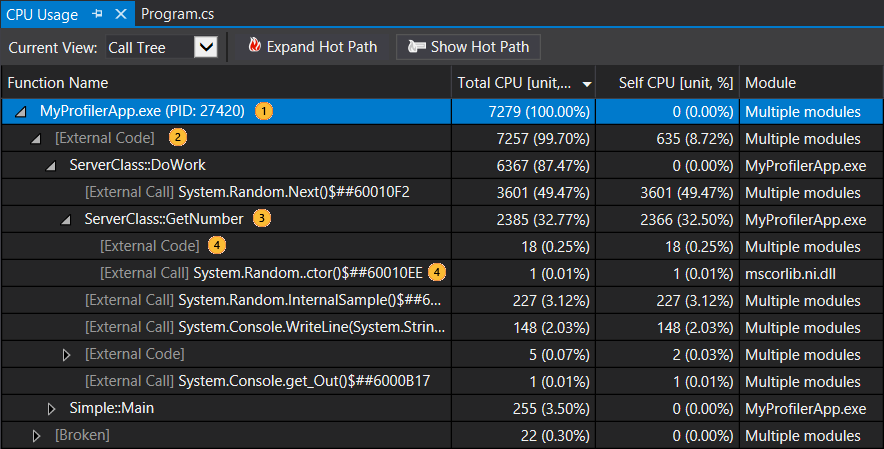
# Using Integrated Development Environment (IDE)

A modern IDE provides a long list of features to aide with development applications and will normally provide a way for additional features to be added using plugins or extensions.

Before IDE’s because the norm, a developer would user several different programs independently of each other to provide a specific functionality… a text editor to write code, an application to check syntax, a compiler, and then a separate debugger and if they were lucky, some sort of testing tools. Several feature we take for granted today would not be available such as IntelliSense and automatic formatting (more on those later).

IDE Features:

* Multi-Document Support  
  An IDE will support the view of multiple documents at once – from code files, reference documents, images or data files, an IDE makes it easy to compare or references multiple docs at once. Some IDEs will support jumping directly to a function in a document:  
  A screenshot of a computer

  AI-generated content may be incorrect.
* Syntax Help   
  Getting the syntax of your code correct is essential, and a good IDE will provide you with the tools to make this easy,
  + Autocomplete – predicts what command you wish to type and allows you to complete it without having to type it all.
  + IntelliSense – a code completion tool that provides both reference information and autocomplete functionality. Although IntelliSense is a Microsoft branded tool, something equivalent is common in all IDEs.  
    
  + Style corrections  
    Code analysis tools can provide guidance on the best or preferred practises to use when writing code. From community perceived best practise to organisation standards, these rules can be automatically applied.  
    
* Document Formatting  
  The formatting of a code file is important and an IDE can provide tools to ensure your code is well formatted using rules either defined by the coding community at large or specified by your organisation. When you code does not match up with the standards, a good IDE will be able to automatically correct your formatting.
  + Indentation – automatically indent your code to easily see code blocks.
  + Bracketing – use of indentation, colours and drawn elements to highlight the usage of brackets to define code blocks
  + Code style anaylsers – automatically highlight and correct rule breaking code, that will functionally correct does not follow standards,
* Debugging
  + Line Numbers – it seems like a very simple thing but by showing line numbers it is exponential easier to find problems from error reports and logs.
  + Watches – watches allow us to inspect the content of variables in our applications. We can also add breaks when values change or reach a threshold value.
  + Execution Stack – the ide can show us the list of calls made to reach the current execution point. Visual Studio is capable of showing this visually as code map too:  
    
  + Performance tools  
    IDE’s usually offer tools to analyse the performance of an application – execution times, memory usage etc. With these we can find and reduce bottlenecks. For example this image shows the amount of time taken by application calls during the execution fo an application:  
    

Variable Scope   
When variables are defined, they will have a scope associated with them. This scope defines access to the variables and is also used during garbage collection. Scope is split into two main categories Global and Local:  
**Global** – if an object or variable can be accessed by anywhere in the application is it said to have a Global scope. Global objects can be changed by any code within the application and should be of limited use because of the problems this can cause.  
**Local** – If an object does not have a global scope, it will have a local scope where access to it is limited to the same code block and any sub-blocks. Local scope supports the principle of encapsulation.

## Static

Static objects and methods can be used without first instantiating an instance of an object, and cannot reference any variables in the object.

For example, the static class TemperatureConvertor can be created using the New keyword:

A computer screen with many colorful text

AI-generated content may be incorrect.

We can access the static methods on this class directly:



Static methods (typically) perform faster than other methds are this is less overhead involved behind the scenes when manipulating the object stack as reduced need to create and destroy objects when they are called as well as less memory required to use them.

## Overloads

The signature of a function call defines the datatypes and number of input and output values. There are situations where a function will allow for different parameters to be used for the same function. This is called Overloading.

For example, an Add function may accept different data types:



Here, the overloaded function will return the same datatype as the values passed in to it.