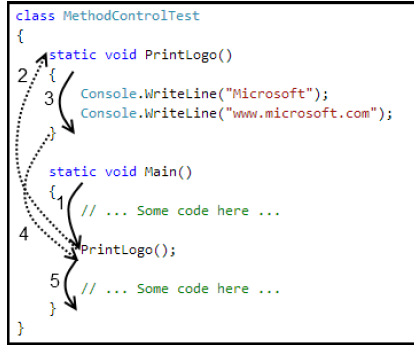
**//--------------------------------------------------------------------- Methods ----------------------------------------------------------------------//**

// To solve a certain task, especially if it is a complex one, we apply the “divide and conquer” concept. We separate the given task into smaller tasks, then develop solutions for them and put them together into one program. These smaller tasks are called **subroutines**, **functions, procedures or methods.**// Methods **must be defined at the class level, outside of any other method**. You may call this method from Main() method but must be defined at same level as the Main() method, within the overall ‘Program’ Class, outside of the ‘Main()’ method itself. A method exists only if it is declared **between the opening and closing brackets of a class** – "**{**" and "**}**". In addition a method **cannot** be declared inside another method's body.  
// **Methods make your code better structured and easy to read**, help us to **avoid code repeating ,** &if a **piece of code is used more than once in a program, it is good to separate it in a method.**  
namespace Learning\_Methods  
{  
 internal class Program  
 {  
 static void Main(string[] args)   
 {   
 Greet(); // Method **call** is the process that **invokes** the already declared method: process of **execution** of the method’s code  
 }  
 static void Greet() // **Declaring** method (method registration) so it can be successfully identified in the rest of the program.  
 { //On declaration, maintain sequence : [access modifier] <return\_type> <method\_name>([<param\_list>])   
 Console.WriteLine(“Hello World!”);// **Implementation** (creation): process of typing code that resolves a particular task.  
 }  
 }  
}  
  
class Arithmetic // Every method **must** **be contained within a class**   
{  
 int CalculateCube (int x) // Method Declaration Syntax: *access-modifiers*  *return-type*  *method-name****(formal-parameter-list)***  
 { // method-name is any valid C# identifier – Cube ; prefer PascalCasing Convention; VerbNoun  
 // parameter list- contains all variable names with type for each, which we give as input: primitive or object  
 return x \* x \* x; // Type of value method returns (type in declaration)- can also be any class type  
 }  
 public float Product(float x, float y) // Each formal-parameter must be declared for their type **individually**  
 { // **Public** access modifier shows that *method can be called by any C# class*, no matter where it is.   
 float m = x \* y; // m - local variable   
 return m; // Methods which return any value are called function  
 }  
 private void Sum(float x, float y)// Can never omit return type even if method doesn't return anything- must write void  
 { //**Private**: The method *cannot be called from anywhere, except from the class in which it is declared*.  
 Console.WriteLine("Sum is " + (x + y)); // return statement is omitted  
 } //In method declaration there **cannot** be **more than one access modifier**.  
   
 static int Add(int x, int y) => x + y; // **Expression-bodied member**s shorten the syntax for single line methods  
 // The above is just syntactic sugar, i.e, the generated IL or CIL (Common Intermediate Language) is same; **=> Lambda Operation** //To call a **static method** there is **no need to have an instance of a class** in which the static method is declared.

 // What **identifies a method** is its **signature - name of the method, and list of parameters.** These two elements define the so-called method specification. The *return type is not part of the method signature*. The reason is that if two methods differ only by their return value types, for the program is not clear enough which of them must be called.// A modifier is a keyword in C#, which gives additional information to the compiler for a certain code. If a method is declared **without an access modifier** (either **public** or **private**), it is **accessible from all classes** in the current assembly, but not accessible for any other assemblies (let say from other projects in Visual Studio). Access modifier controls visibility. **Static** is **not** an access modifier; it can be used alongwith **Public.**// The area where a **local variable** exists, and **can be used**, begins from the line where the variable is declared and ends at the closing curly bracket "}" of the method body. This is the so-called area of visibility of the variable (**variable scope**).  
// **Invoking** a method/ **Calling** a method: process of **execution** of the method’s code, placed into its body. MethodName();  
// When a method executes it **takes control over the program**. Main() method is the point of entry for the compiler, it calls PrintLogo() . The control passes to PrintLogo(). If in this method, however, we call another one, the caller will give the control to the called method. The called method will return back the control to the caller right after its execution finishes. The execution of the caller will continue from that line, where it was before calling the other method.

// A method can be invoked from its own body. Such a call is referred to as **recursion**

// If a method is called in the same class, where it is declared and implemented, it can be called at a line before the line at which it is declared; We are allowed to invoke (call) a method before it is declared in code.

// From within Main() method, we *can invoke a method defined within a separate class* but *using dot operator* and simply calling the method name if it is **public static** OR by *creating an instance of the class* first, then calling if it is **public but non static**.

}