Code constructs are the declarations and code structures in an application

When considering defensive coding with coding constructs the key considerations are to minimise defects by:

* Follow best practice guidelines
* Clarity of intent
* Maximise predictability

Local variable declarations:

* Location. Used to be at the top of the method due to historical requirements. It should be declared as close to where its needed as possible. Means it’s less effort to follow the code. Clarifies intent.
* Use good naming. Consistent and clear. Camelcase to distinguish from properties
* Initialisation. Where possible initialise at the point of declaration.
* Var keyword. When the type is obvious use the var keyword as this can save time. Also useful in complex return types where you’re not quite sure. E.G. LINQ.

If statements:

* Use braces. If it only contains one operation, acceptable to have a one line if statement.
* Create the shortest, least syntactically noisy construct that is unambiguous to anyone reading
* Else. Think about whether you want to be notified of an if failure.
* Think positive. When feasible define the conditional in the positive.
* Nesting if statements. Many nested if statements make it difficult to read and harder to test because of the increased code paths.

Switch statements:

* The order of the case statements. The most common cases first. Numerical, alphabetical, logical
* Always have a default case
* Simple actions per case statement.

Enums:

* Better than using magic numbers because they add meaningful names and easier to read.
* Define within a namespace so all classes can use it and to avoid name clashes. Don’t nest within a class
* IsDefined. Can be used on an Enum to determine if the integer value has a corresponding value in the Enum. But it’s recommended not to use as it’s expensive in CPU. Use TryParse instead. Added in C# 4.0.
* Enum.TryParse Takes the string type and returns the Enum type. Otherwise returns a false. Case can be ignored.

Casting:

* Cast carefully. Blind casting casts without checking if the cast is valid.
* Check the value first using the as (safe cast) operator. Will return null if fails. Then check for the null. This way it won’t crash