Coding Guidelines
&
Standards

Coding Standards - Meaning

- Coding standards are a set of
  - guidelines,
  - best practices,
  - programming styles and
  - conventions
- That developers adhere to when writing source code for a project.
- All big software companies follow these standards

- The best applications are coded properly.
- This sounds like an obvious statement,
- but by ‘properly’, I mean that the code not only does its job well,
- but is also easy to
    - understand
    - add to
    - maintain and
    - debug

The Problem

- When we learn a language and start coding, we usually begin to code in a specific style.
- In most cases, we’ll write in a style that we want, not one that has been suggested to us.
- But once we start to code using a particular style, it will become second nature — we’ll use that style in everything we create.
- When project gets bigger and team gets bigger your personal style will create a problem

Solution : Coding Standards & Guidelines

- Coding standards tells developers how they must write their code.
- Instead of each developer coding in their own preferred style, they will write all code to the standards.
- This makes sure that a large project is coded in a consistent style — parts are not written differently by different programmers.
- Not only does this solution make the code easier to understand, it also ensures that any developer who looks at the code at a later stage will know what to expect throughout the entire application.

But, what is the **difference** between a coding guideline and a coding standard?

- It is mandatory for the programmers to follow the coding standards. Compliance of their code to coding standards is verified during code inspection. Any code that does not conform to the coding standards is rejected during code review and the code is reworked by the concerned programmer .
- In contrast, coding guidelines provide some general suggestions regarding the coding style to be followed but leave the actual implementation of these guidelines to the discretion of the individual developers.

Some Standards

Standard headers for different modules

- The header of different modules should have standard format and information for ease of understanding and maintenance.

The following is an example of header format that is being used in some companies:

- Name of the module.
- Date on which the module was created.
- Author’s name.
- Modification history.
- Synopsis of the module. This is a small write up about what the module does.
- Different functions supported in the module, along with their input/output parameters.
- Global variables accessed/modified by the module.

Naming conventions

- Global variable names would always start with a capital letter (e.g., GlobalData)
- local variable names start with small letters (e.g., localData).
- Constant names should be formed using capital letters only (e.g., CONSTDATA).
- Method names should be starting with lower case (eg getValue())
- Prefix *is* should be used for boolean methods
- Package name should be in lower case (mypackage, edu.iitk.maths)
- File naming convention

Conventions regarding error return values and exception handling

- The way error conditions are reported by different functions in a program should be standard within an organization.
- For example, all functions while encountering an error condition should either return a 0 or 1 consistently
- This facilitates reuse and debugging.

Commenting and layout

- Single line comments for a block should be aligned with the code block
- There should be comments for all major vars explaining what they represent
- A comment block should start with a line with just /* and end with a line with */
- Trailing comments after stmts should be short and on the same line

Some Standard practices or guidelines

- Each variable should be given a descriptive name indicating its purpose.
- Use of a variable for multiple purposes can lead to confusion
- Use of variables for multiple purposes usually makes future enhancements more difficult.

- Code should be well-documented: As a rule of thumb, there should be at least one comment line on the average for every three source lines of code
- Length of any function should not exceed 10 source lines: A lengthy function is usually very difficult to understand as it probably has a large number of variables and carries out many different types of computations. For the same reason, lengthy functions are likely to have disproportionately larger number of bugs.
- Do not use GO TO statements: Use of GO TO statements makes a program unstructured. This makes the program very difficult to understand, debug, and maintain.

- Can you actually read the code? Is it spaced out clearly?
- Do you separate blocks of code into ‘paragraphs’ so that different sections are easily defined?
- Are you using indentation to show where control structures (if, else, while and other loops) begin and end, and where the code within them is?

- Are your variable naming conventions consistent throughout the code and do they briefly describe that data that they’ll contain?
- Are functions named in accordance with what they do?
- If you come back to the code in a few weeks or months, will you be able to work out what’s happening without needing to look at every line?

- How are you commenting the work?
- /*
-     changed by :
-     change description :
-     date of change :
-      client :
-      bug ID :   */

- Var iTax = “”;   // for holding value of intermediate tax

- No hard coding should be there

- Use initialization files (.ini)
- Use cascading style sheet (.css)

Example

- Var dateofreservation

- Var date_Of_Reservation

- Function calculatefinaltax()

- Function calculateFinalTax()