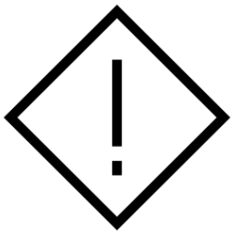
Regular Mistakes and Suggestions

In this chapter, we assemble some of the most common mistakes or don't(s) in Go-programming. We often refer to the full explanation and examples from the previous chapters.

WE'LL COVER THE FOLLOWING

In the previous chapters, we were sometimes warned with **Remarks** and **Notes** for Go misuses. Be sure to look for the specific section in this course on that subject when you encounter a difficulty in a coding situation like that.



Common pitfalls

Here is an overview of pitfalls for your convenience, referring to where you can find more explanations and examples:

- Never use `var p*a` to not confuse pointer declaration and multiplication.
- Never change the counter-variable within the for-loop itself.
- Never use the value in a for-range loop to change the value itself.
- Never use `goto` with a preceding label.
- Never forget the *braces* () after a function name to call the function specifically when calling a method on a receiver or invoking a lambda function as a Goroutine.
- Never use `new()` with maps, always use `make()`.
- When coding a `String()` method for a type, don't use `fmt.Print` or alike in the code, use the `fmt.Sprint` family of methods.

- Never forget to use `Flush()` when terminating buffered writing.
- Never ignore errors because ignoring them can lead to program crashes.
- Do not use global variables or shared memory; they make your code unsafe for running concurrently.
- When using *JSON*, make sure that the data fields you require are exported in the data structure.
- Use `println` only for debugging purposes.

Best practices

In contrast, use the following:

- Initialize a slice of maps the right way.
- Always use the “comma, ok” (or checked) form for type assertions.
- Make and initialize your types with a factory.
- Use a pointer as receiver for a method on a struct only when the method modifies the structure, otherwise use a value.
- Always use the package `html/template` to implement data-driven templates for generating HTML output safe against code injection.

These were some of the most common mistakes and overlooked errors. Best practices should be followed to enhance performance. Study the next lesson to code with proficiency keeping in view the best practices.