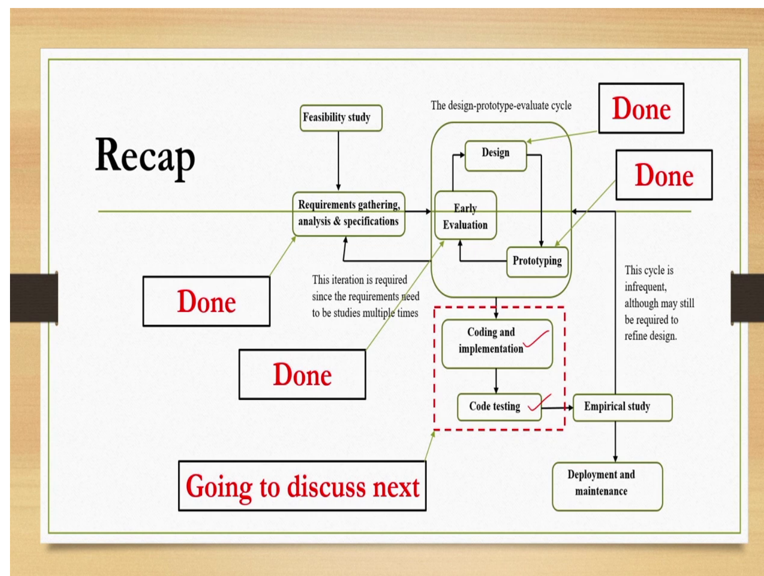
Design and Implementation of Human – Computer Interfaces
Prof. Dr. Samit Bhattacharya
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture: 27
Coding Basics

Hello and welcome to the NPTEL MOOCS course on design and implementation of human computer interfaces, lecture number 24 where we are going to start our discussion on implementation of the designs that we have learned in our previous lectures. So, before we start let us quickly recap what we have learned and where we are now.

(Refer Slide Time: 01:02)



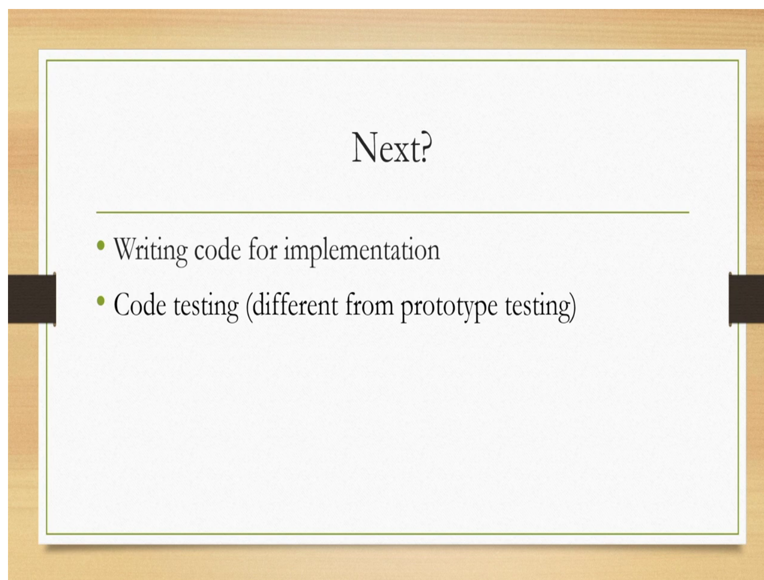
So, as you may recollect in this course our main focus is design and development of human computer interfaces in other words design and development of interactive systems. Now for interactive systems in order to design develop and deploy an interactive system we need some systematic approach. For that we should take recourse to some software development life cycle we are discussing one such development lifecycle for interactive system design.

In the life cycle there are several stages requirement gathering analysis and specification stage then design prototyping and evaluation cycle coding and implementation stage code testing stage empirical study stage and deployment and maintenance stage and there are cycles between the

stages as shown in the diagram. Among these stages we have so, far covered the requirement gathering analysis and specification stage that is done.

Then we have covered the design stage also prototyping and evaluation of prototyping have been covered in our earlier lectures. If you may recollect there are two designs we mentioned one is design of the interface where primary concern is usability and there we require prototyping and evaluation of prototyping to come up with an interface that is usable. The other design is the design of the system or code design we covered both these concepts in our earlier lectures.

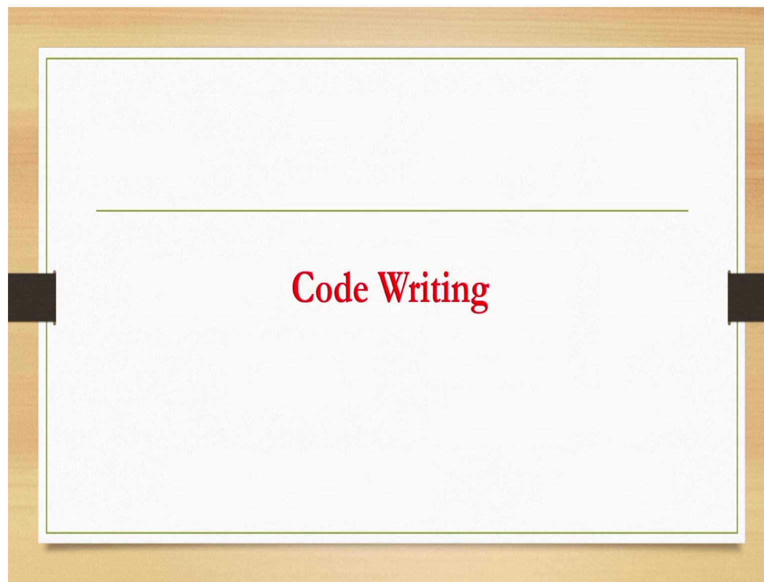
(Refer Slide Time: 02:59)



Once we managed to come up with a design of the system or design of the code. What is next? Next is writing of the actual code for implementing the system and that is followed by testing of the code. Now you should keep in mind that earlier we talked about testing of the interface in terms of building prototypes and getting it tested with domain expert users. Here what we are going to talk about is testing of the code after we have written the code to implement the system we are going to test the code.

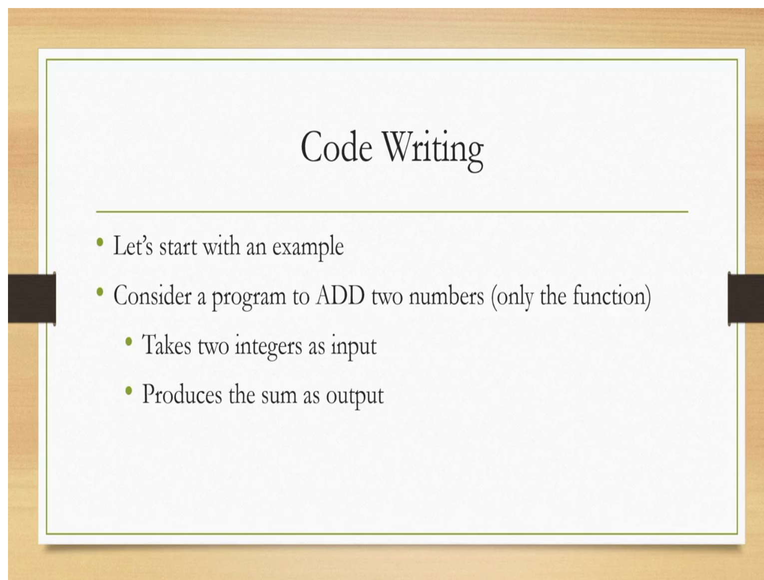
So, clearly that is different from testing of the prototype. So, these are the next things that we should learn about. In other words we have covered requirement gathering design prototyping and evaluation cycle and now we are going to focus on the next two stages namely coding and implementation and code testing in this and subsequent few lectures that is our lecture topics.

(Refer Slide Time: 04:05)



So, there are two things one is writing the code to implement and second one is testing the code to see whether the code has any problem in terms of bugs that may be syntactic problems semantic problems. So, in testing we find out these problems and in the first is that is coding and implementation we put our effort in writing codes. So, let us start our discussion on how to write code.

(Refer Slide Time: 04:40)

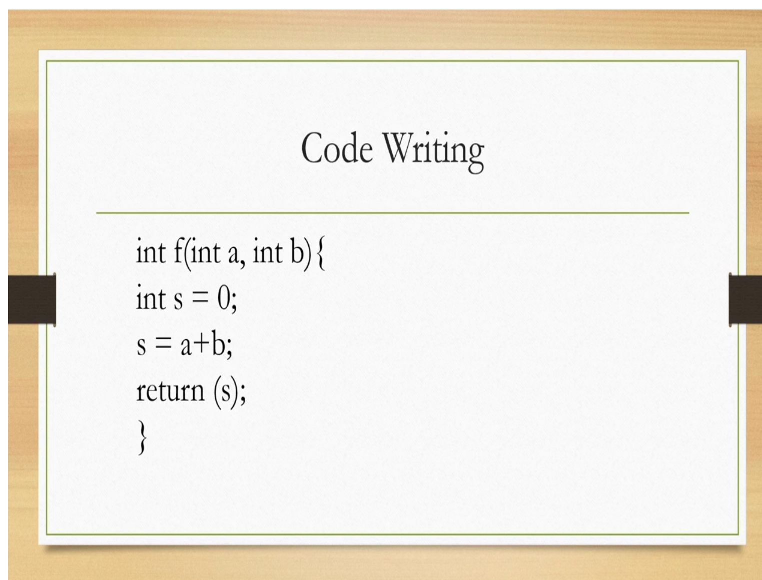


In fact all of you may have written some code by. Now and you may be familiar with code writing practices. However it is not necessary that you always follow good coding practice often

we have the tendency to write codes in a very brief manner in a very convoluted manner or avoid certain things which makes the code readable and maintainable. So, it is always advisable to follow good writing practices to take home this point let us start with an example.

Why good quadratic practices are important and what are those practices. Suppose we want to write a program to add two numbers that is a very simple program and if I give you this problem I am sure that most of you will be able to do it within no time. However let us see what are the common mistakes that we are likely to commit while writing this code. So, our problem is to add two numbers. Now this function so we want to write a function for performing this operation and this function takes as input two integers and produces the result or the sum as output.That is a very simple description of a problem.

(Refer Slide Time: 06:09)



If this problem is given to you many of you if not all will quickly come up with a solution and that solution may likely to be something like this assume C syntax. So, this is a very well known syntax I am sure all of you may be familiar with this syntax that we follow in the C language. So, here we have declared the return type of the function int given a name to the function f then within parenthesis we provided the definition of the input that is integer a and integer b.

So, a and b are two inputs of type integer then there is a variable defined inside the function s. So, in test is initialized to 0 and then s is assigned the sum of the two arguments a and b. So, this

is the result. So, at the end this result is returned as output return s is the statement used for the purpose. So, this may look like a simple program and it is actually a correct program but is it following proper code writing practices. Let us investigate in little more detail.

(Refer Slide Time: 07:37)

Code Writing

```
int f(int a, int b){
int s = 0;
s = a+b;
return (s);
}
```

Is/are there any problem(s) with this code?

If this program is given to you can you spot any problems with this piece of code I think many of you may be thinking that this is the right program. So, there is no issue but that is not necessarily the case let us see. One problem is with the name of the function. So, here we have given it a name f. Now f does not convey any meaning. So, ideally the name of the function should convey some meaning actually the purpose of the function should be conveyed in the name.

For example here we could have given a name like sum or we could have given a even more elaborate name like sum of integers that would have been a better name than giving a name like f which does not convey any meaning.

(Refer Slide Time: 08:28)

Code Writing

```
int f(int a, int b){
int s = 0;
s = a+b;
return (s);
}
```

Function name (not meaningful)

variable name (not meaningful)

Then name of the variables like a b s again ideally we should name our variables in a way such that they convey the purpose of those variables in contrast here we have given some arbitrary name. So, they do not convey any meaning. Instead suppose we have given name as integer input one or the first input integer second input or even better integer first number integer second number that would have given some idea that okay we are now talking about the two inputs where one is labeled as first number one is labeled as second number.

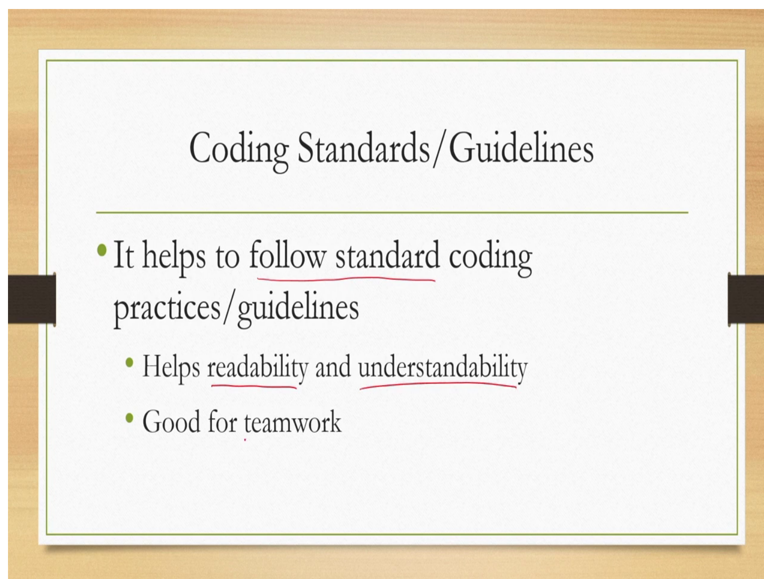
Similarly instead of s we could have written sum that would have been a better naming convention than simply writing an arbitrary character such as s to represent the variable. A third interesting thing that we should note here is that in this piece of code we have not used any indentation. So, ideally there should be indentation here in this line in this line and in this indentation means that should have been shifted right by some space all these three lines would have been shifted right rather than aligning with the top line and the last line.

These things like meaningful function names meaningful variable names proper indentation and additionally some documentation or comments conveying the purpose of the statements or the overall function adds lots of values to any program that you are writing. In fact that helps us to understand and maintain a program in a much better way than if we are not following this conventions.

(Refer Slide Time: 10:38)

So, when we are writing a program what are the things that we should remember many of these things I am sure most of you have already encountered but most of us often ignore those guidelines or those things that we should follow while writing a code. So, let us quickly recap those things that are useful in writing good programs or good codes.

(Refer Slide Time: 11:07)

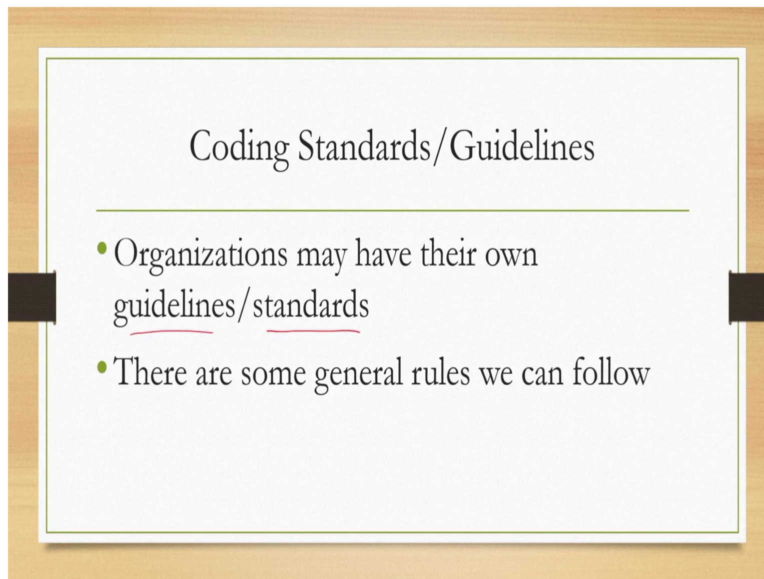


So, those are essentially coding standards or guidelines they help to follow good practices. So, if we are following standard coding guidelines or practices then the code that eventually we get are better written better understood and easier to maintain. So, they help in improving readability and understandability most importantly that helps in teamwork. So, generally code components

are written by different members of a team for large projects it is really the case that a single developer is writing the entire code.

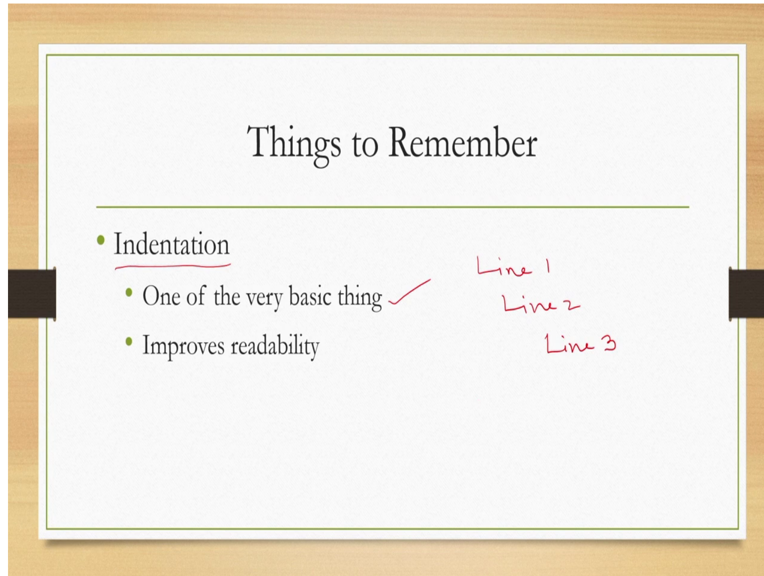
So, different team members write different pieces of the code as assigned to them. Now if a standard is followed if some guidelines are followed then it is easier to understand and integrate those different pieces of code at the end and test those. So, it helps in implementing good teamwork.

(Refer Slide Time: 12:06)



Now organizations or companies may have their own guidelines or standards that they would like to follow however there can be some generic guidelines that everybody can follow. So, it is always advisable that we are at least aware of those general guidelines.

(Refer Slide Time: 12:29)



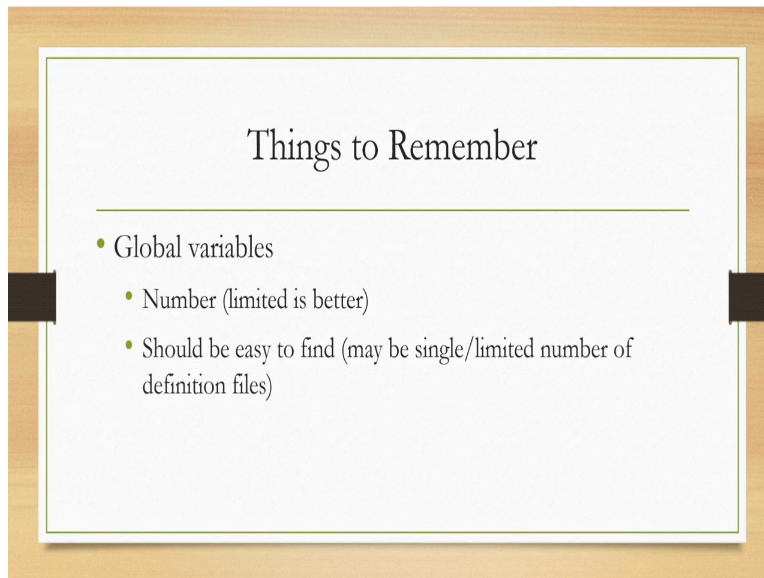
One is indentation this is one of the very basic thing. Whenever we are writing code like line one then line two line three for certain constructs if we follow this type of indentation where indentation means basically shifting towards the right the subsequent lines. For example while writing the for loop or he fails statements it is advisable to follow indentation properly then that improves readability if everything is written along the same vertical line then readability is affected. So, indentation is one of the basic things that we should keep in mind.

(Refer Slide Time: 13:23)

Next is the use of global variables where we should follow a standard or common naming convention. For example global variables may always start with uppercase letters and we can

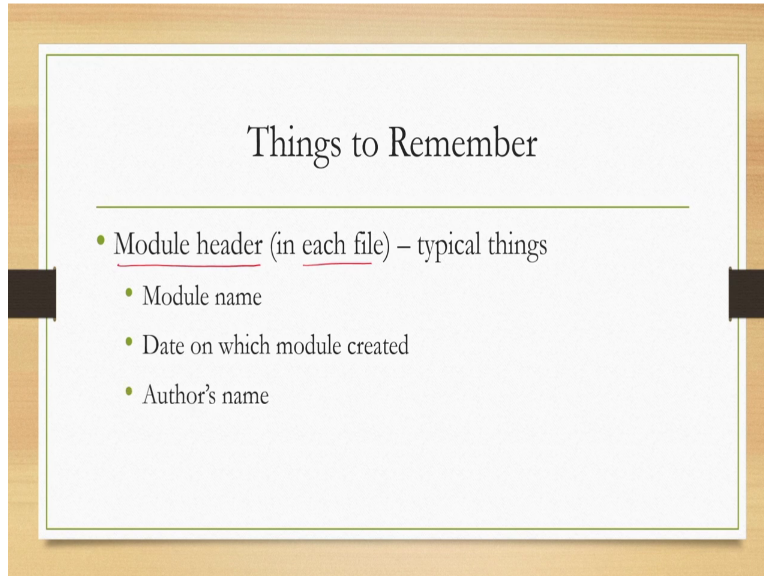
also use prefix or suffix to indicate group global variables such as gbl underscore name or name underscore gbl. So, these are some of the conventions that we can use. So, that global variables are sufficiently distinguishable from other variables or local variables clearly that helps in understanding the purpose of the global variables as well as differentiating between different types of variables.

(Refer Slide Time: 14:12)



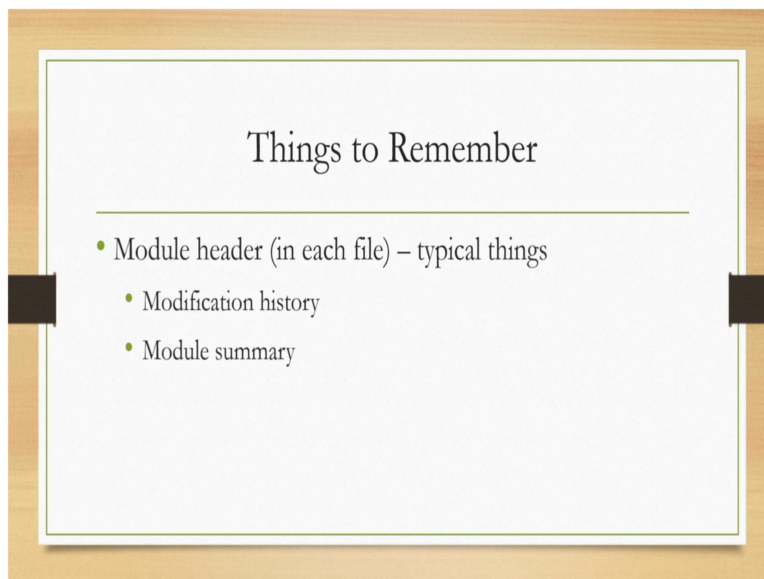
Also when we are using global variable we should remember that it is better to keep the number of global variables limited. So, we should not use them indiscriminately and in large numbers also they should be easy to find. Ideally they should be kept in a single or limited number of definition files that means they should be kept in one place. So, that it is easier to locate all the global variables used in your program.

(Refer Slide Time: 14:46)



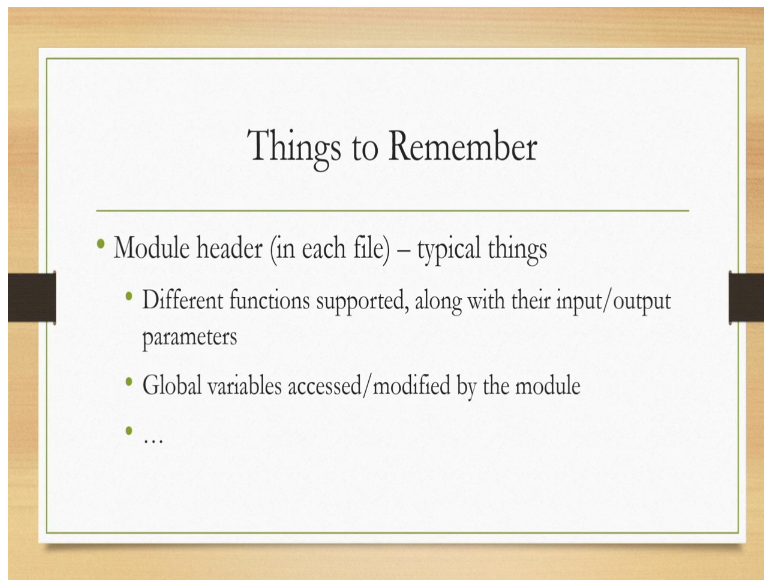
Then it is also a good practice to use module header. So, whenever you are writing or implementing a module there should be a header added to the module that means to each of the files that implements a module and there are few things typically you should keep in the header what are those things the name of the module the date on which the module was created.

(Refer Slide Time: 15:16)



Name of the author or authors; modification history so if you have made any changes in the module. So, that history that is the dates on which the changes are made and what are the changes those things should be summarized and kept in the header. Then a summary of the content of the module should be kept in the header.

(Refer Slide Time: 15:36)



Also different functions supported in the module along with their input output parameters should be listed in the header itself for a quick view of the content of the module. Global variables if any global variables are used or modified in the module those should be listed in the header file. These are some of the typical things there may be other things you may add but at least these things should be kept in the header file for each module.

So, you should keep a header for each module and in the header section you should keep all these information.

(Refer Slide Time: 16:14)

Things to Remember

- Naming conventions – maintain consistency
  - Global variable names may always start with capital letter
  - Local variable names may made up of small letters
  - Constant names may always be capital letters

Other important thing is naming conventions so when we are trying to name variables functions modules we should follow some convention and maintain consistency in naming that is very important while giving names to different components of a program. We already talked about global variables and their naming conventions. For local variables we may follow any convention standard convention such as the names may be made up of small letters for local variables constant names may always be in capital letters. So, whenever we are trying to define some constant we should always use capital letters to name it.

(Refer Slide Time: 17:10)

Things to Remember

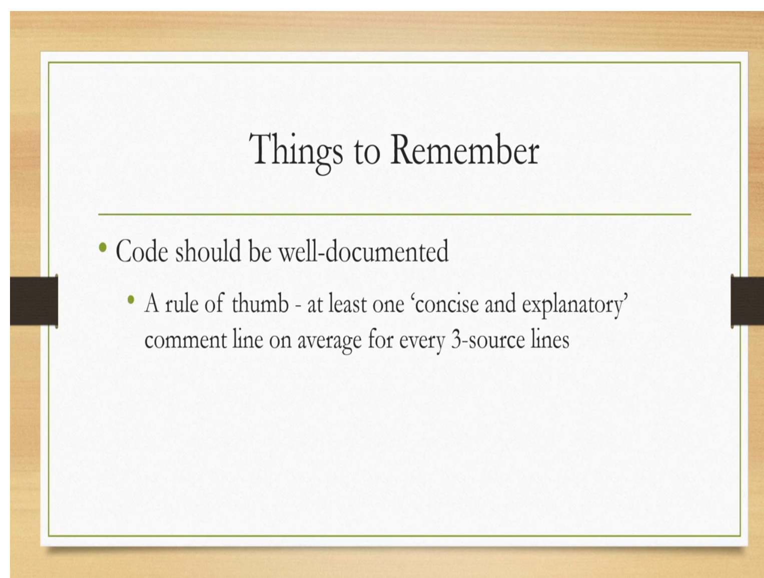
- Error and exception handling
  - Error conditions reporting should be standard
    - E.g., functions may return a 0 for error or 1 otherwise, consistently

Another important thing that we should remember is dealing with errors and exceptions how you deal with errors and exceptions actually determines the quality of your code. Ideally we should use some standard error condition reporting facility for example functions may return a 0 for error or 1 otherwise consistently. So, we should follow this convention for all functions it should not happen that.

For some functions we define the zero return value as for error and one as for success whereas for some other functions we reverse it we say that 0 is for success and 1 is for error. Whatever standard we follow we should follow it consistently across all the parts of the code. We must take care of exceptional situations. So, while execution takes place if there is some exception such as division by zero then that should be taken care of so we have to identify and take care of exceptions that may happen during execution of any code.

Another important thing is that the code that you are writing should be well documented. So, every portion of the code should have proper documentation to explain the meaning purpose and content of the code.

(Refer Slide Time: 18:36)



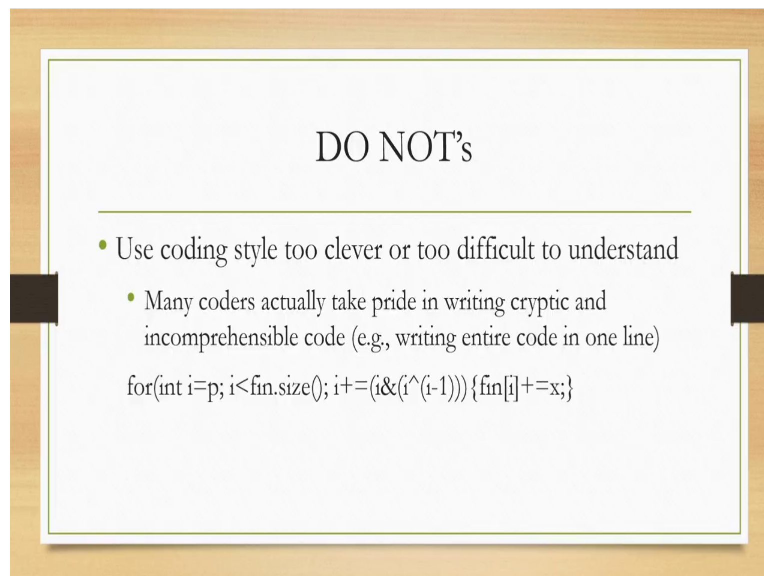
We can follow our rule of thumb as the name suggest this is just a rule of thumb. So, nothing sacrosanct about it but we can follow it is that at least one concise and explanatory comment on average for every three source lines should be there. Now you can make it every five source lines

one comment or if the source lines are very obvious and self explanatory then no need to add the comment depending on the situation you may judge. But generally for every three lines ideally a line of explanatory comment adds quality to the code and makes it more readable.

And also it is always a good idea to create a detailed user manual and technical manual to accompany your implementation. User manual is for high level view of the code whereas technical manual is for detailed understanding of the code. So, those are in brief the things that you should remember while writing the code. That is follow indentation naming conventions, module headers, handle errors and exceptions and use documentations.

These are some standard things that we should follow while writing a code. There are again certain things that we should avoid. So, we should consciously keep in mind things that we should avoid what are those things.

(Refer Slide Time: 20:13)

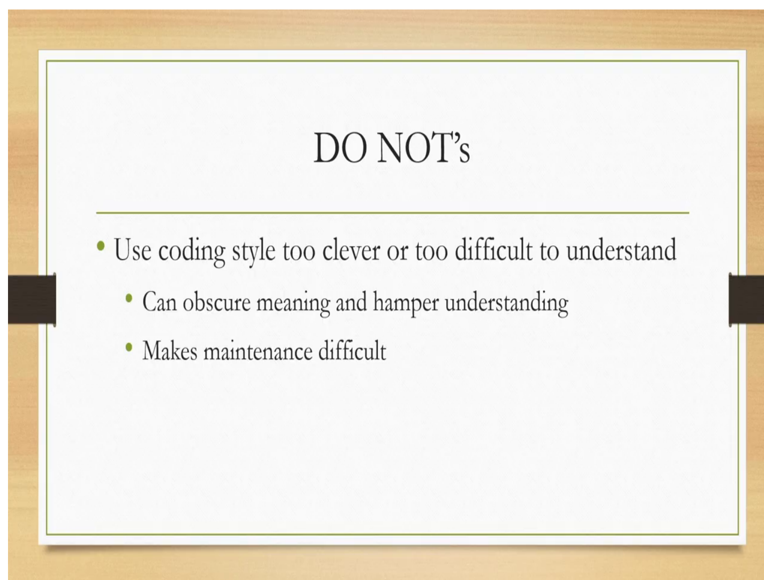


So, some of us have the tendency of writing codes that appear to be very cleverly written. Now the problem associated with that is that such quote unquote cleverly written codes are very difficult to understand and difficult to maintain only the programmer knows what the code does and others if they are asked to understand it others find it very difficult to understand the code that affects maintainability and eventually that affects team spirit.

For example many coders actually take pride in writing very cryptic and incomprehensible code such as writing the entire code in one line. So, you may have seen such examples and you probably have found them very interesting but in practice if you are following that then it actually is problematic for maintaining the code understanding the code and come up with a good teamwork. An example is this statement a for loop initialized i equal to p i less than fin dot size then i + equal to a very complex expression.

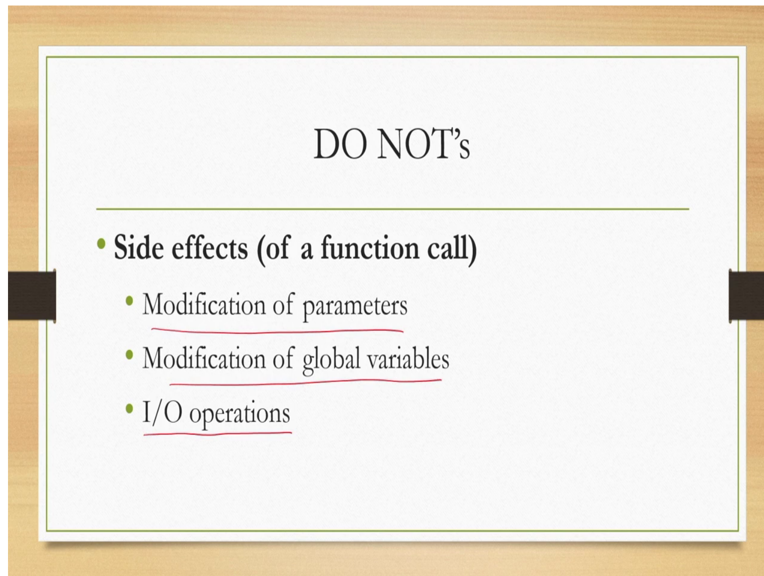
So, from this it is difficult to understand the purpose and also it is not necessary. So, ideally you should try to write codes that are simple and easy to understand. So, that they are understandable to others rather than only to you and they are easy to maintain writing such complicated code we should always try to avoid.

(Refer Slide Time: 22:03)



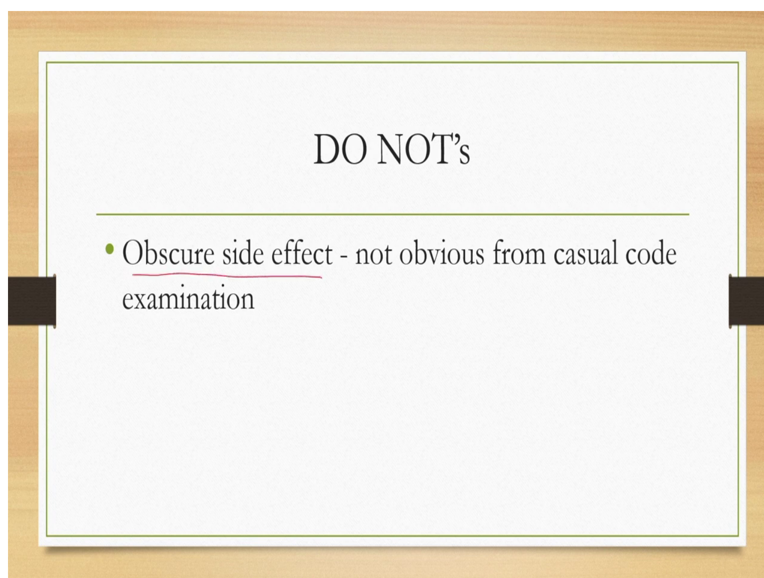
So, those coding styles can obscure meaning and hamper understanding as I already said and max maintenance difficult that is pretty obvious from the previous example.

(Refer Slide Time: 22:17)



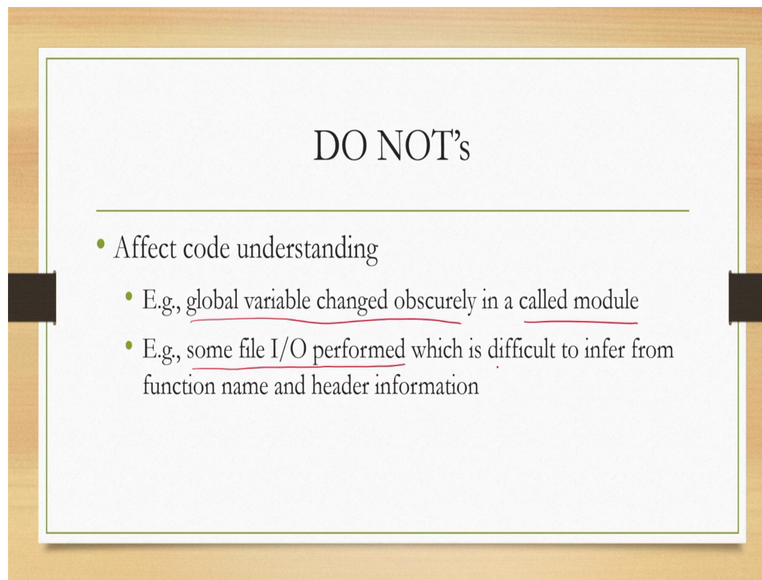
Also we should keep in mind of obscure side effects that may result from our writing. So, what is this idea of a side effect generally associated with the function calls which involves modification of parameters modification of global variables and input output operations. So, these are typically side effects associated with improper writing of code things may change without our knowledge and without our requirement.

(Refer Slide Time: 23:03)



So, when we are writing code sometimes it becomes difficult to understand from the code itself what kind of side effects it can create. So, those are obscure side effects which are not apparent from the reading of the code.

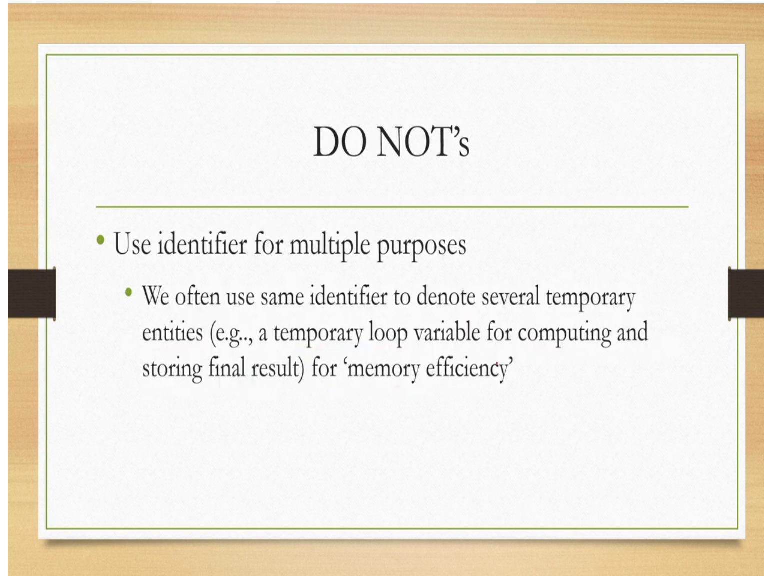
(Refer Slide Time: 23:27)



So, when we are writing code we should try to write code in a way. So, that the side effects that means whatever modifications that are likely to take place because of a function call are pretty apparent to the person who is reading the code or going through the code. Clearly obscure side effects affect understanding of the code because we do not know what is going to be changed. When for example a global variable may have changed obscurely in a called module which we are not aware of so, we are probably dealing with a changed value rather than the original value that.

We thought is stored in the global variable. Another example is some file input output operation is performed due to function call which again was not clear to us because that is an obscure side effect which in turn have changed certain things in the file which is not what we want.

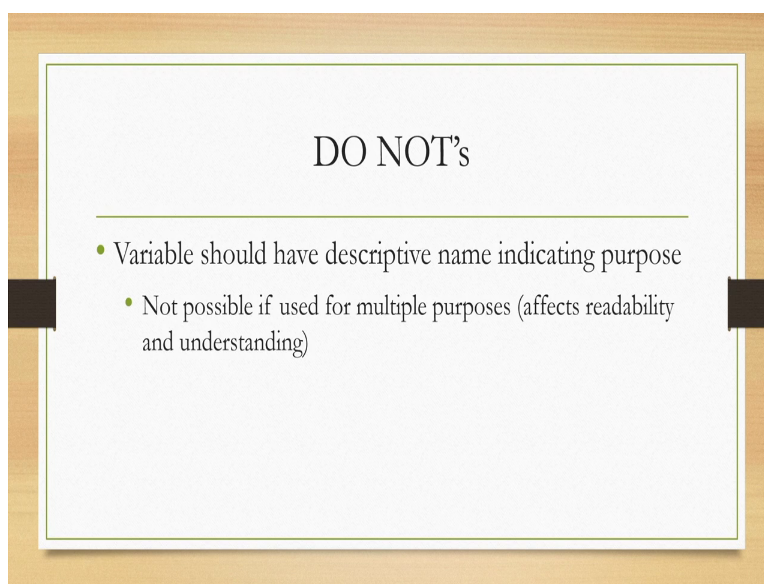
(Refer Slide Time: 24:27)



Similarly another thing that you should try to avoid is you should not use identifiers same identifier for multiple purposes. We often use the same identifier to denote several temporary entities. For example a temporary loop variable for computing and storing final results may have used the same identifier. Now this is often done to take into account memory efficiency issues but then while it may improve memory efficiency understanding the code becomes difficult.

So, ideally we should use different identifiers for different purposes rather than using the same identifier for different purposes.

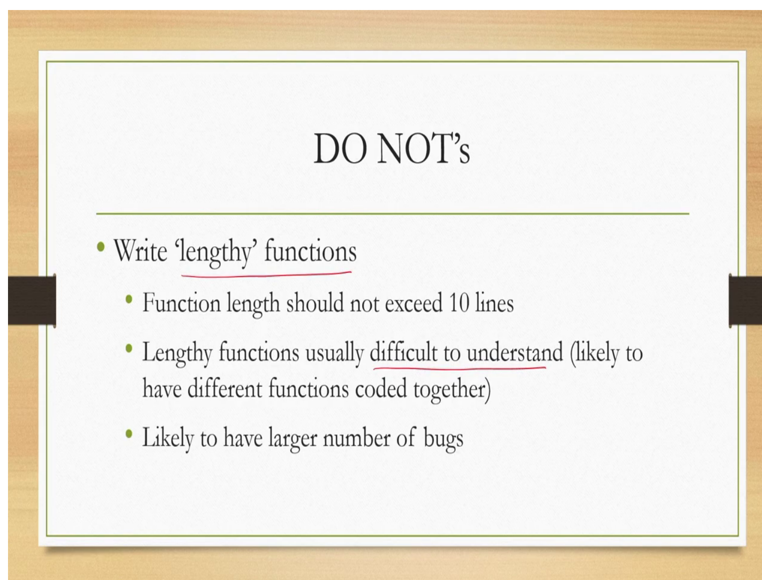
(Refer Slide Time: 25:22)



Also we should not use variables that do not have descriptive names indicating purpose. So, this is already mentioned earlier naming convention should be such that the name should reflect the purpose. So, we should not or do not variables that do not reflect the purpose. Now if you are using same identifier for different purposes then this naming convention you cannot follow because the purposes are different and If you want to give a name indicating the purpose then if the identifier is same for different purposes then clearly you cannot give different names to the same identifier.

So, you have to use a same identifier indicating one purpose whereas it is being used for other purposes as well. So, we should avoid that situation.

(Refer Slide Time: 26:19)



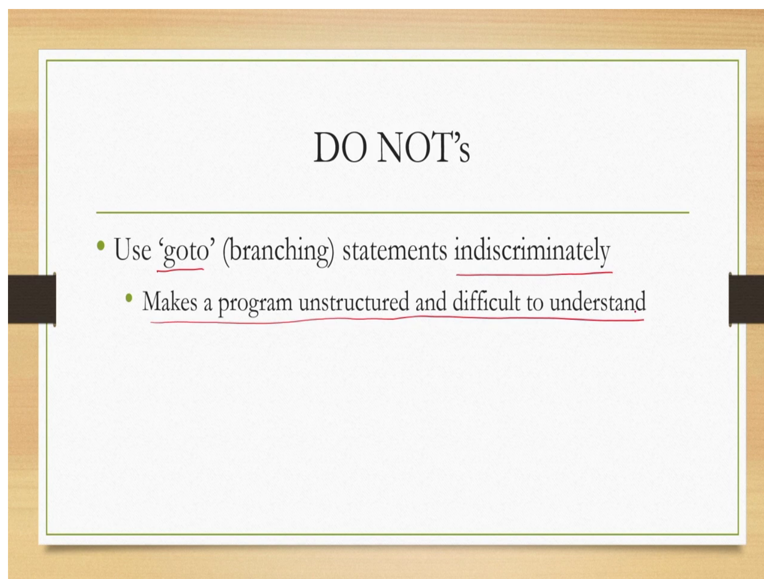
Another thing that we should avoid is write lengthy functions. Now this is a very interesting thing when we are writing functions often we are not much bothered about the length of the function. In fact most of us may not be knowing what is the appropriate length how many lines of code should be there in a function. One guideline you can follow is that function length should not be more than 10 lines of code.

So, under a function ideally you should not use more than 10 lines of codes. If you are having functions involving more than 10 lines of codes then those are usually difficult to understand and most likely such functions can be split into multiple functions you probably instead of splitting

have joined them together and put them under same function. So, any function whose length is more than 10 or maybe more than 15 lines of code probably is a good candidate to be split into multiple functions.

Also if you have lengthy functions then there is high likelihood that it will lead to larger number of bugs that is quite obvious the more lengthier the code is the possibility of having more bugs increases that is true for functions as well.

(Refer Slide Time: 27:54)



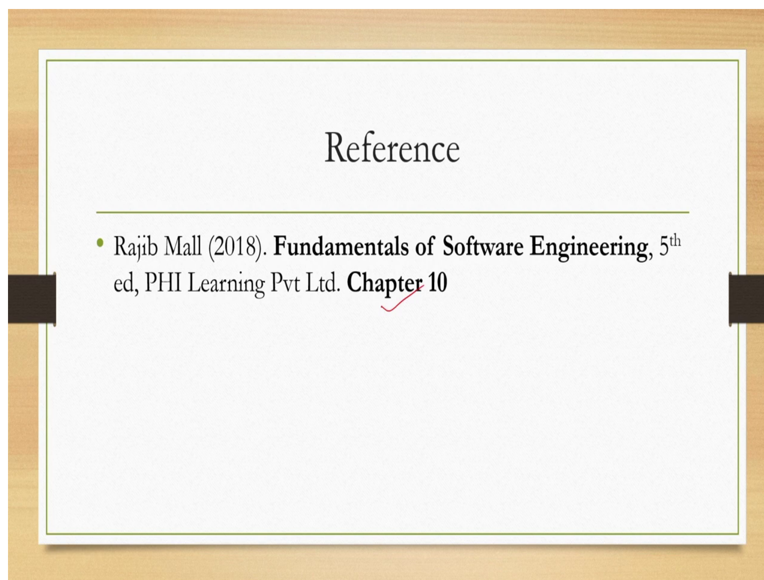
Another thing that you should avoid is to use the go to or branching statements indiscriminately that means you should be very careful while using branching statements such as go to statements. Of course this is more appropriate this is more suitable for older generation of programming environments nowadays go to statements are not there in most of the programming languages. So, this is not a big concern nowadays but still if you are using similar constructs you should be careful and should limit the number of such constructs in your code.

Otherwise it is very difficult to understand the logical flow in the code because there are too many branching and that eventually makes a program unstructured and difficult to understand. So, these are the some of the things that you should avoid should avoid lengthy functions you should avoid having same identifier for multiple purposes and so on. Also, should avoid using

too many branching constructs. So, there are things you should follow and there are things you should avoid.

These are some of the things that you should keep in mind while writing your program and I am sure many of you may be already aware of these things but occasionally we fail to keep in mind these things while we write our program. So, it is always advisable to follow these things and write a good code which is easier to understand easier to maintain and easier to modify. I hope you enjoyed the content that we covered in today's lecture.

(Refer Slide Time: 29:59)



Whatever I have mentioned you can find in this book particularly chapter 10 you can go through to get more details about these things in subsequent lectures we are going to talk about testing of the code. So, in this lecture we talked about how to write a good code which is useful for implementation then once the code is written and the system is implemented we need to test it. So, in subsequent lectures we are going to talk about how to test a code that we have already written, so looking forward to see you in the next lectures, thank you and goodbye.