A quick introduction

Imagine you want to know how to get from "Old street" to "New street". You call a random person to get help and you ask "What is the fastest way I can get from Old street 19 to New street 3?". Would the random person be able to help you?

There are a lot of "New" and "Old" streets in the world, so first he would need to know what country and city you are in. If the street layout is complicated then maybe you need to name some buildings that are near you. Of course because speed is an issue then your means of transportation is also important; are you by foot, bike, car or truck? You can go on foot into places where you can't get by truck.

Those questions around the immediate question "How do I X?" is the context of the problem, it helps other people orient to the problem. Giving advice without context could mean that the other person will describe the wrong city.

In person it's easy to ask those questions and can be done in rapid succession, although on the forums this will result in a lot of back-and-forth questions that can be avoided. So, how to properly give the context to the problem?

How to ask a good question

People on the forums have limited time available. So, to speed things up, here is a small template for asking a question that will get better answers and answers quicker:

> *The gist of the problem you are having.*

> *How did you encounter the problem?*
> *What are you trying to accomplish?*
> *What is the context of the problem?*
> *What are the requirements for the solution?*
> *What are the context-specific constraints/properties?*
> *Your compilable and runnable example on play.golang.org*
>
> *Other notes about the situation (production/school/playing/learning)*

Things to keep in mind:

- Take your time to spell-check and make sure that your sentences are readable.
- The solution can be far away from the immediate problem. So be sure that the question contains answers to the 5 whys. The template implicitly already contains answers to 3 whys.
- The context of the question is important, so always give one. Not giving the context may end up hurting you, because you'll get an answer that's more suited to a different context. The context is the end-user or domain problem and goal, with the information how it tries to solve it.
- Try not to ask abstract questions but if you do then add multiple concrete examples. Asking abstract things without concrete examples (usually) wastes time. Although they can be sometimes interesting, the concrete examples allow the discussion to be precise.
- Avoid imprecise terms such as "a lot of data" or "needs to work fast". Give something measurable such as "it needs to work upto 1GB of data" or "it needs to communicate with 1000 concurrent clients within 100ms".
- If you are not able to give the compilable example in go, any language will do... many people are well versed in other languages. If you are not able to make it compile or run, it's fine too.
- Try to explain your situation:
    - Noting that "it is homework" means people can explain a bit more and won't do the homework for you.

    - Information such as "it needs to work on a cluster of X nodes" can provide situational context. Where and how the code must eventually run.
    - Noting that "it's for learning X" makes it clear that you are trying to get a deeper understanding of different approaches.
    - Noting that "I'm under NDA and can't disclose the code", means people won't bother asking for it. Although you should still try to come up with a similar situation, it helps the answerer.
- Giving a simplified example can be helpful for readers, but do not forget to give the full version as well. The simplified version is different from the full version and therefore the solutions can be different.

Of course, don't worry too much... you can't answer all possible questions. It's not needed to write a 4 page essay on the whole thing. People can always ask when something needs clarification.

The story of a bad question

Why is that template necessary? Let's put you in the shoes of the answerer, and let's say you get this question:

> *How do I use the reflect `Set` method?*

From the askers perspective this may look like a straightforward problem. Of course, you have no idea what the asker is trying to do. Is he trying to set a value on a value, struct, array or a map? This means you need to ask the first question: *Can you give an example?*.

Now the asker will provide an example:

> *I'm basically trying to do this:*
>
> ```
> m := make(map[string]int)
>
> v := reflect.ValueOf(m)
>
> key := reflect.ValueOf("hello")
>
> val := reflect.ValueOf(123)
>
> v.MapIndex(k).Set(val)
>
> print(m)
> ```
>
> *But it just panics.*

Now you've got some code that even doesn't compile. This means you need to copy paste it somewhere and fix the mistakes. Putting it on play.golang.org makes it easier to see the problem (e.g. play.golang.org/p/fCxBIL9V4Y).

The fix is simple; just use `SetMapIndex` instead. Of course that may not be the full story. The asker now comes back with another problem:

> *Yay, I've got the reflection working, but it's not fast enough.*
> *How can I make it faster?*

What does "faster" mean? What is he trying to do? So you need to ask more about the specifics of the problem. So you ask: *What are you trying to accomplish?*

> *I'm trying to implement a set package that can store generic types.*

Well, ok, but that still didn't answer why it needs to be faster. Why is he using `reflect` for that anyway, when you can use `map` instead. So you answer: *Can't you just use `map` instead (e.g. `map[type]struct{}`)? What are you writing the `set` package for?.*

> *I'm writing a program that goes over multiple sequences and that finds the common elements.*
> *The main use for is for using on nucleotides and it should work on large datasets. But it should also work on proteins, which are represented by another type.*

> *Yes, the `map[NucleotideIndex]struct{}` works a bit better, but it's still not fast enough, and I would now have to write the same code for proteins.*

Finally we have all the information that's needed. The solution is straightforward: `biogo` *https://code.google.com/p/biogo/ has most of the stuff necessary for handling large datasets and parallelism etc. Also what's the "`NucleotideIndex`". The asker may have found the answer and say: Thanks that's really nice.* but there is a possibility of:

> *Whoa, that looks nice, but I'm in a bioinformatics course and we need to write that code ourselves. The "`NucleotideIndex`" is a "`struct {Nucleotide Nucleotide; Index int}`".*

That looks weird, why would you do it that way. Nevertheless we've all done something stupid, so... now you can start suggesting something better: *Probably dealing with `map[Type]map[int]struct{}` would be much easier. Because you are dealing with sequences and the set elements are always used in an increasing order you can just store the index in an array e.g. `map[Type][]int`. Also if the memory starts to become the problem you could make it into a run-length-encoded set...*

Now we can have a more meaningful discussion about the (dis)advantages of different set types.

Hopefully after this long example, you probably understand why it's useful to provide more information. The initial back and forth questions could have been avoided. Proper question could have saved time for the question asker and the answerer.

The initial question could have been:

> *How do I use the reflect `Set` method with `MapIndex`?*
>
> *I'm trying to set a value on a generic map for a set package. But it just gives a panic when I try to do that.*
> *play.golang.org/p/fCxBlL9V4Y*
>
> *I use the set package in a program that goes over multiple sequences and find all the common elements in a sequence. The sequence elements could be nucleotides or proteins. The program needs to be able to deal with data-sizes upto 1GB.*
>
> *My current code is available at github.com/...*

> *I'm writing this for a bioinformatics course, so I need to implement it myself.*

Summary:

- The best answer depends on the context. In some cases maybe research.swtch.com/sparse would be more appropriate. If the speed isn't important using a `map` would be sufficient. So the requirements are also important.
- The problem can be somewhere else. As you saw the answerer didn't expect that the structure of the program was at fault. Using a struct `NucleotideIndex` with a `map`, meant he had to build elaborate things with reflection. Often when you fix the higher-level problem, everything else will become much easier.
- Constraints/properties matter. The property "set elements are used in increasing order" meant that there was a simple method that didn't require a full-blown `set` implementation. This specialized structure can be much faster. The information about the system, context or domain may make the problem much simpler.
- The solution may differ from your usual approaches. Maybe the asker decided to use `reflect` package, because he was used to generics in Java. Go is different language, so the final solution may look a lot different from the solution in Java.

More tips

- Smart questions
- Short, Self Contained, Correct (Compilable), Example