Q: Can I stop these complaints about my unused variable/import?

A: There's a good explanation at https://golang.org/doc/faq#unused_variables_and_imports.

Q: Is the defer keyword in other languages?

A: Defer was new in Go. We originally added it to provide a way to
recover from panics (see "recover" in the spec), but it turned out
to be very useful for idioms like "defer mu.Unlock()" as well.
Later, Swift added a defer statement too. It seems clearly inspired
by Go but I'm not sure how close the details are.

Q: Why is the type after the variable declaration, unlike C languages?

A: There's a good explanation at https://blog.golang.org/gos-declaration-syntax.

Q: Why not adopt classes and OOP like in C++ and Java?

A: We believe that Go's approach to object-oriented programming,
which is closer to Smalltalk than to Java/C++/Simula, is more
lightweight and makes it easier to adapt large programs. I talked
about this at Google I/O in 2010. See
https://github.com/golang/go/wiki/GoTalks#go-programming for links
to the video and slides.

Q: Why does struct require a trailing comma on a multiline definition?

A: Originally it didn't, but all statements were terminated by
semicolons. We made semicolons optional shortly after the public
release of Go. When we did that, we tried to avoid Javascript's
mistake of making the semicolon rules very complex and error-prone.
Instead we have a simple rule: every line ends in an implicit
semicolon unless the final token is something that cannot possibly
end a statement (for example, a plus sign, or a comma). One effect
of this is that if you don't put the trailing comma on the line,
it gets an implicit semicolon, which doesn't parse well. It's
unfortunate, and it wasn't that way before the semicolon rules, but
we're so happy about not typing semicolons all the time that we'll
live with it. The original proposal for semicolon insertion is at
https://groups.google.com/d/msg/golang-nuts/XuMrWI0Q8uk/kXcBb4W3rH8J.
See the next answer also.

Q: Why are list definitions inconsistent, where some need commas and some do not?

A: The ones that don't need commas need semicolons, but those
semicolons are being inserted automatically (see previous answer).
The rule is that statements are separated by semicolons and smaller
pieces of syntax by commas:

```
        import "x";
        import "y";

        var x = []int{
                1,
                2,
                3,
        }
```

When you factor out a group of imports, you still have semicolons:

```
        import (
                "x";
                "y";
        )
        var x = []int{
                1,
                2,
                3,
        }
```

But then when we made semicolons optional, the semicolons disappeared
from the statement blocks leaving the commas behind:

```
        import (
                "x"
                "y"
        )

        var x = []int{
                1,
                2,
                3,
        }
```

Now the distinction is between nothing and something, instead of
two different characters, and it's more pronounced. If we had known
from the start that semicolons would be optional I think we might
have used them in more syntactic forms, or maybe made some forms
accept either commas or semicolons. At this point that seems
unlikely, though.

Q: Why does Go name its while loops "for"?

A: C has both while(cond) {} and for(;cond;) {}. It didn't seem
like Go needed two keywords for the same thing.

Q: There seem to be a lot of new languages emerging these days,
including Rust, D, Swift and Nim, among probably others. Are there
any lessons you've learned from these other languages and their
communities that you wish you'd been able to incorporate into Go?

A: I do watch those languages for developments. I think they've
learned things from Go and I hope we've also learned things from
them. Some day I'd like the Go compiler to do a better job of
inferring ownership rules, or maybe even having lightweight ownership
expressions in the type system. Javari, Midori, Pony, and Rust are
inspirations here. I wrote a bit more about this at
https://research.swtch.com/go2017.

Q: Why the focus on concurrency and goroutines?

A: We knew from past experience that good concurrency support using
channels and lightweight processes would make writing the kinds of
systems we built at Google a lot easier, as I hope the lecture
showed. There's a bit more explanation at https://golang.org/doc/faq#csp,
and some background about our earlier experiences at
https://swtch.com/~rsc/thread/.