Question 1: Can you explain the difference between `:=` and `=` in Go? And more importantly, *when* would using `:=` cause a compilation error?

Question 2: You're designing a function that calculates financial transactions. You need to represent monetary values. Walk me through your thinking: would you use `float64`, `int`, or something else? What are the tradeoffs, and what would you actually choose in production code?

Question 3:

```
var x int = 10
var y int64 = 25
z := x + y
```

This code won't compile. Why? And what does this tell you about Go's philosophy compared to languages like JavaScript or Python?

Question 4: You're reviewing a colleague's code and see this:

```
var (
    name string
    age int
    salary float64
)
```

All three variables are used later in the function. What's the problem here from a code quality perspective? How would you refactor this, and why does Go encourage a different approach?

Question 5: Explain runes to me like I'm a developer coming from Python. When would using `string` vs `[]rune` actually matter in real code? Give me a concrete example where getting this wrong would cause a bug.

Question 6: Go doesn't have a `char` type like C or Java. It has `byte` and `rune` instead. What problem was the Go team trying to solve with this design? Do you think they made the right call?

Question 7: Without running the code, tell me what this prints and why:

```go
const a = 10
const b = 20.5
result := a * b
fmt.Printf("%T\n", result)
```