Imagine writing a program that will take a long time to complete because of a complex calculation or because you're handling a large amount of data. If an error occurs during this computation then you're liable to lose all of the results that were calculated before the error, or your program may not finish a critical task that a program further down your pipeline is depending on. If you anticipate the possibility of errors occuring during the execution of your program then you can design your program to handle them appropriately.

The tryCatch() function is the workhorse of handling errors and warnings in R. The first argument of this function is any R expression, followed by conditions which specify how to handle an error or a warning. The last argument finallyspecifies a function or expression that will be executed after the expression no matter what, even in the event of an error or a warning.

Let's construct a simple function I'm going to call beera that catches errors and warnings gracefully.

```
beera <- function(expr){
  tryCatch(expr,
         error = function(e){
           message("An error occurred:\n", e)
         },
         warning = function(w){
           message("A warning occured:\n", w)
         },
         finally = {
           message("Finally done!")
         })
}
```

This function takes an expression as an argument and tries to evaluate it. If the expression can be evaluated without any errors or warnings then the result of the expression is returned and the message Finally done! is printed to the R console. If an error or warning is generated then the functions that are provided to the error or warning arguments are printed. Let's try this function out with a few examples.

```
beera({
  2 + 2
})
Finally done!
[1] 4

beera({
  "two" + 2
})
An error occurred:
Error in "two" + 2: non-numeric argument to binary operator

Finally done!

beera({
  as.numeric(c(1, "two", 3))
})
A warning occured:
simpleWarning in doTryCatch(return(expr), name, parentenv, handler): NAs
  introduced by coercion

Finally done!
```

Notice that we've effectively transformed errors and warnings into messages.

Now that you know the basics of generating and catching errors you'll need to decide when your program should generate an error. My advice to you is to limit the number of errors your program generates as much as possible. Even if you design your program so that it's able to catch and handle errors, the error handling process slows down your program by orders of magnitude. Imagine you wanted to write a simple function that checks if an argument is an even number. You might write the following: