# The Maybe (not) Monad

# Uncaught exceptions

```
$ some_important_app
***Uncaught exception: divide by zero***
```

Now what?

"Modern" languages (Swift, Go) use type-checking to prevent this kind of error.

Essentially, programmers are forced to account for failures wherever they might occur.

## The Maybe type

Recall:

```
data Maybe a
  = Nothing
  | Just a
```

- wrap desire values with `Just`; use `Nothing` for failure/error/non-existence
- this forces all consumers to explicitly consider Nothing vs Just-wrapped.

Direct use can lead to tremendous clutter!

See Haskell file for a particularly horrifying example.

## *do* (monads) to the rescue

Like IO, but replace IO actions with *Maybe* values.

```
do
  -- line 1
  -- line 2
  -- ...
  -- line n
```

where each line is one of the following.

1. `e` where `e` is any expression of type `Maybe a` .
2. `x <- e` where `e` is any expression of type `Maybe a` .
3. `let x = e` where `e` is an expression of any type.

To evaluate the *do*, evaluate each line in order:

1. `x <- e`:
   Evaluate `e`: if `Nothing`, the whole *do* has value `Nothing`.
   If the value is `Just v` then give the name `x` to `v` and evaluate the remaining lines.
2. `e`
   Evaluate `e`: if `Nothing`, the whole *do* has value `Nothing`.
   If the value is `Just v` then evaluate the remaining lines.
3. `let x = e`: give the name `x` to the value of *e*.

See Haskell file for example.