Equational reasoning.

The point is to use no side effects/mutate state in methods or in the initialization blocks of `val`s.

The `val` and its initialization block can thus be swapped without worry.

... results in a value.

`if`, `while`, `for`, `try`, `match`, function calls

`f` results in 2
`g` results in 1

For one reason, their meaning would be unclear in a function literal.

Use curly braces around generators and filters to avoid semicolons. Otherwise consecutive filters must be separated with a semicolon.

...without `catch`. Even in Java!

The value is the value of `try`, unless `case` or `finally` overrule with `return`.

If `throw` is uncaught, no value is given at all (type `Nothing`).

- Use `goOn` type `Booleans`.
- Use recursion.

Optional.

```
do {
   //whatever
} while (condition)
```

```
someConstant match {
  case "salt" => "pepper"
  case "chips" => "salsa"
  case _ => "huh"
}
```

```
() : Unit
```

Range values.

They result in a new collection made up of yielded values.

They result in a value of type `Nothing`, but of course there are no instances of `Nothing`.

Conceptually, each statement is in a nested scope.

```
try {
  //code
} catch{
case ex: FileNotFoundException => //handle
case ex: IOException => //handle
}
```

Scala always allows shadowing.