Referential transparency is to methods as ___ is to variables.

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.

Unlike Java, what does try/catch/finally ...

... results in a value.

Name the built-in control structures.

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

What are the results of:

`def f(): Int = try {return 1} finally{return 2}

def g(): Int = try {1} finally {2}`

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

Why doesn't Scala have `break`/`continue`?

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

Give an alternate `for` expression syntax.

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

`finally` can be used...

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

What is the value of `try/catch/finally`?

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`).

Give two strategies for living without `break`/`continue`.

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

Exception handling is...

Optional.

Write a `do-while` loop.

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

Give an example of matching on a `String`.

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

What is the result of assignment?

```
() : Unit
```

What do `to()` and `until()` result in?

Range values.

What do `for` expressions result in?

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

What do `throw` expressions result in?

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

Why do scripts appear to allow re-declarations?

Conceptually, each statement is in a nested scope.

Give the `try`/`catch` syntax.

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

What is the difference between Java and Scala scoping rules?

Scala always allows shadowing.