

# Scala Parser Combinators

Alexander Daniel • [@lachdrache](https://twitter.com/lachdrache) • <http://lachdrache.wordpress.com>



```
class PlusMinusParser extends JavaTokenParsers {
```

```
  def expr: Parser[Int] = targetActual <- comment
```

```
  def targetActual: Parser[Int] = duration ~ duration ^^ { case target-actual => actual-target }
```

```
  def comment: Parser[String] = ".*".r
```

```
  def duration: Parser[Int] = day | hourMinute
```

```
  def day: Parser[Int] = num<~"d" ^^ { days => days*MinutesPerDay }
```

```
  def hourMinute: Parser[Int] = hour~minute ^^ { case h~m => h*60 + m }
```

```
  def hour: Parser[Int] = num<~"h"
```

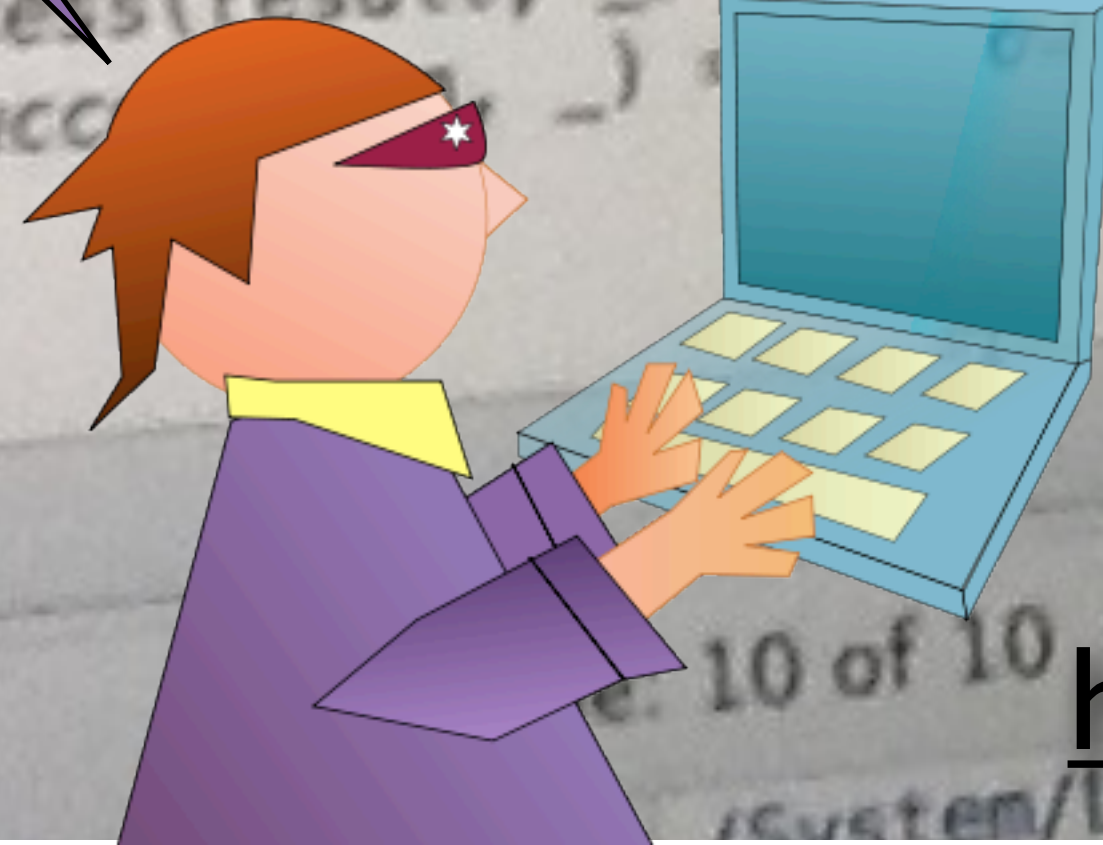
```
  def minute: Parser[Int] = num<~"m"
```

```
  def wholeNumber: Parser[Int] = wholeNumber ^^ { _.toInt }
```

```
  // 7*60 + 42
```

```
  def apply(input :String):Int = parseAll(expr, input) match {  
    case Success(result, _) => result  
    case NoSuccess(_, _) => new RuntimeException(msg)  
  }
```

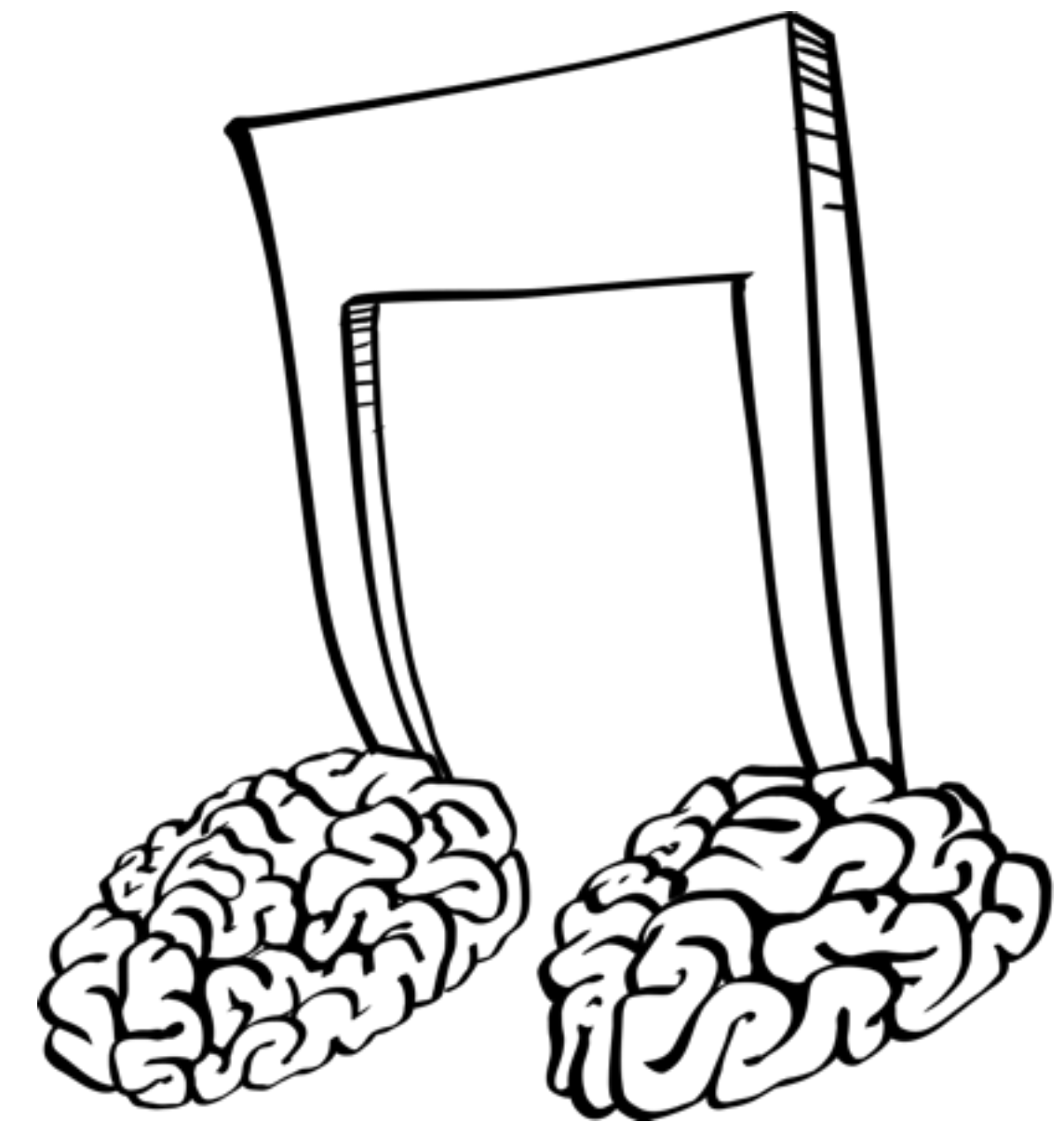
Slides are boring... Show me  
the code!!!



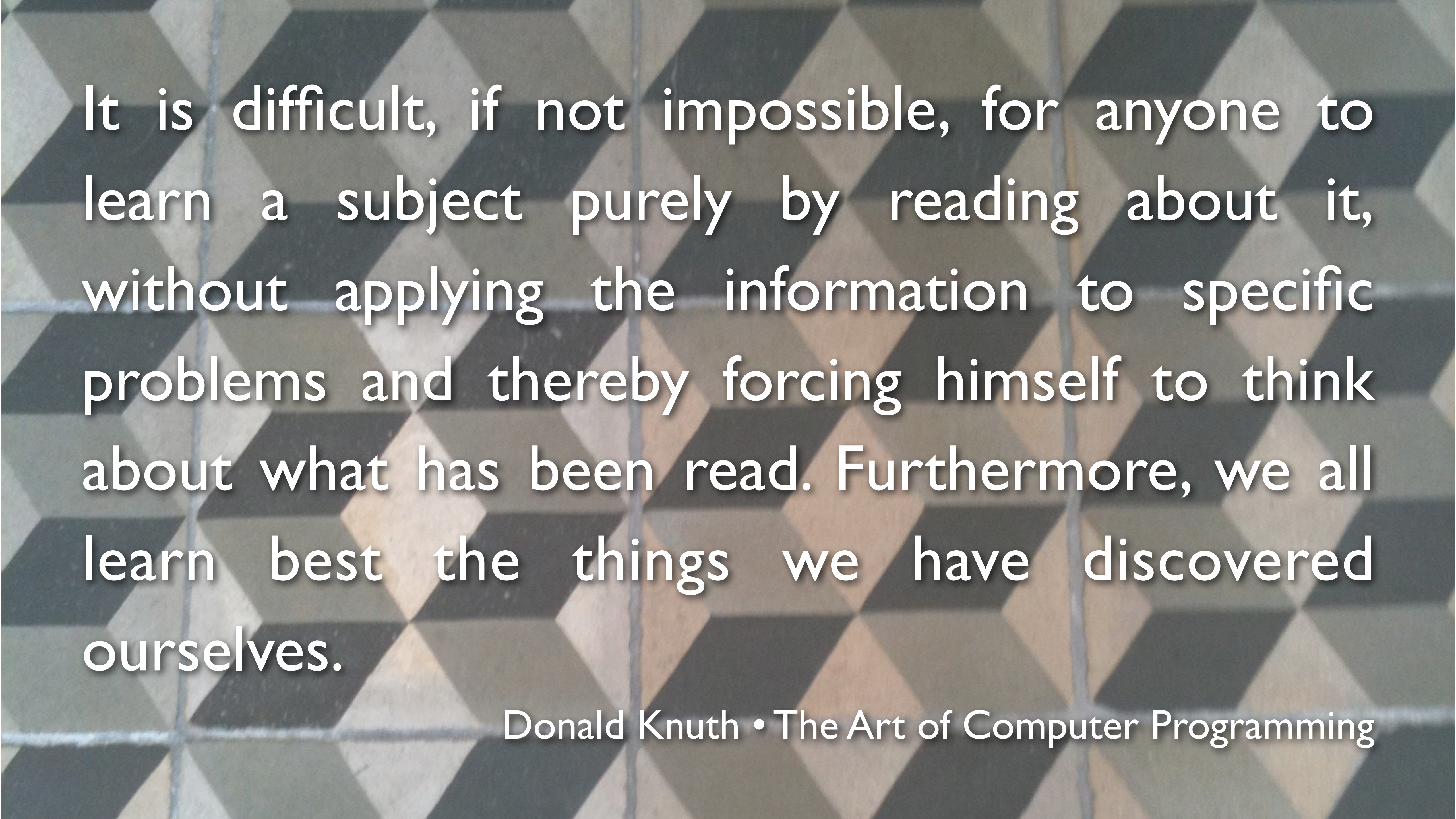
<https://github.com/AlexanderDaniel/plus-minus>



- Easy to learn and use
- Good example of an internal Scala DSL
- Great extensibility and maintainability
- Suitable for parsing small to medium sized input







It is difficult, if not impossible, for anyone to learn a subject purely by reading about it, without applying the information to specific problems and thereby forcing himself to think about what has been read. Furthermore, we all learn best the things we have discovered ourselves.

Donald Knuth • The Art of Computer Programming



Wanna  
learn  
more?



<https://github.com/AlexanderDaniel/parser-combinators#resources>

“...”

*literal*

“...”.r

*regular expression*

$P \sim Q$

*sequential composition*

$P < \sim Q, P \sim > Q$

*keep left/right only*

$P \mid Q$

*alternative*

$\text{opt}(P)$

*option*

$\text{rep}(P)$

*repetition*

$\text{repsep}(P, Q)$

*interleaved repetition*

$P \wedge \wedge f$

*result conversion*

*all other special characters*

*\* / %*

*+ -*

*..*

*= !*

*< >*

*&*

*^*

*|*

*all letters*

*all assignment operators*



First character of a  
method defines the  
precedence

