Conditionals and Value Definitions

August 31, 2012
To express choosing between two alternatives, Scala has a conditional expression \textit{if-else}.

It looks like a \textit{if-else} in Java, but is used for expressions, not statements.

Example:

\begin{verbatim}
  def abs(x: Int) = if (x >= 0) x else -x
\end{verbatim}

$x \geq 0$ is a \textit{predicate}, of type Boolean.
Boolean Expressions

Boolean expressions $b$ can be composed of

```plaintext
true  false    // Constants
!b     // Negation
b && b  // Conjunction
b || b  // Disjunction
```

and of the usual comparison operations:

```plaintext
e <= e, e >= e, e < e, e > e, e == e, e != e
```
Here are reduction rules for Boolean expressions (e is an arbitrary expression):

\[
\begin{align*}
!true &\rightarrow false \\
!false &\rightarrow true \\
true \&\& e &\rightarrow e \\
false \&\& e &\rightarrow false \\
true \vert\vert e &\rightarrow true \\
false \vert\vert e &\rightarrow e
\end{align*}
\]

Note that \&\& and \vert\vert do not always need their right operand to be evaluated.

We say, these expressions use “short-circuit evaluation”.
Exercise: Formulate rewrite rules for if-else

\[
\text{if} \ (b) \ e_1 \ \text{then} \ e_2
\]

\[
\text{if} \ (\text{true}) \ e_1, \ \text{else} \ e_2 \rightarrow e_1
\]

\[
\text{if} \ (\text{false}) \ e_1, \ \text{else} \ e_2 \rightarrow e_2
\]
We have seen that function parameters can be passed by value or be passed by name.

The same distinction applies to definitions.

The `def` form is “by-name”, its right hand side is evaluated on each use.

There is also a `val` for, which is “by-value”. Example:

```
def z = 3 + 4
```

```
val x = 2
val y = square(x)
```

The right-hand side of a `val` definition is evaluated at the point of the definition itself.

Afterwards, the name refers to the value.

For instance, `y` above refers to 4, not `square(2)`. 
Value Definitions and Termination

The difference between val and def becomes apparent when the right hand side does not terminate. Given

```plaintext
def loop: Boolean = loop
```

A definition

```plaintext
def x = loop
```

is OK, but a definition

```plaintext
val x = loop
```

will lead to an infinite loop.
Exercise

Write functions \texttt{and} and \texttt{or} such that for all argument expressions \texttt{x} and \texttt{y}:

\begin{align*}
\text{and}(x, y) & \quad = \quad x \ \&\& \ y \\
\text{or}(x, y) & \quad = \quad x \ |\!\!\!| \ y
\end{align*}

(do not use \texttt{||} and \texttt{&&} in your implementation)

What are good operands to test that the equalities hold?