Conditional Statements in Python

This implementation of \texttt{absolute_value} raises several important issues:

A conditional statement in Python consists of a series of headers and suites: a required \texttt{if} clause, an optional sequence of \texttt{elif} clauses, and finally an optional \texttt{else} clause:

\begin{verbatim}
if <expression>:
    <suite>
elif <expression>:
    <suite>
else:
    <suite>
\end{verbatim}

When executing a conditional statement, each clause is considered in order. The computational process of executing a conditional clause follows.

1. Evaluate the header's expression.
2. If it is a true value, execute the suite. Then, skip over all subsequent clauses in the conditional statement.

If the \texttt{else} clause is reached (which only happens if all \texttt{if} and \texttt{elif} expressions evaluate to false values), its suite is executed.

**Boolean contexts.** Above, the execution procedures mention "a false value" and "a true value." The expressions inside the header statements of conditional blocks are said to be in \textit{boolean contexts}: their truth values matter to control flow, but otherwise their values are not assigned or returned. Python includes several false values, including 0, \texttt{None}, and the \textit{boolean} value \texttt{False}. All other numbers are true values. In Chapter 2, we will see that every built-in kind of data in Python has both true and false values.

**Boolean values.** Python has two boolean values, called \texttt{True} and \texttt{False}. Boolean values represent truth values in logical expressions. The built-in comparison operations, \texttt{>, <, >=, <=, ==, !=}, return these values.
This second example reads "5 is greater than or equal to 5", and corresponds to the function `ge` in the `operator` module.

This final example reads "0 equals -0", and corresponds to `eq` in the `operator` module. Notice that Python distinguishes assignment (`) from equality comparison (`==`), a convention shared across many programming languages.

**Boolean operators.** Three basic logical operators are also built into Python:

Logical expressions have corresponding evaluation procedures. These procedures exploit the fact that the truth value of a logical expression can sometimes be determined without evaluating all of its subexpressions, a feature called *short-circuiting*.

To evaluate the expression `<left> and <right>`:

1. Evaluate the subexpression `<left>`.
2. If the result is a false value `v`, then the expression evaluates to `v`.
3. Otherwise, the expression evaluates to the value of the subexpression `<right>`.

To evaluate the expression `<left> or <right>`:

1. Evaluate the subexpression `<left>`. 

2. If the result is a true value \( v \), then the expression evaluates to \( v \).

3. Otherwise, the expression evaluates to the value of the subexpression <right>.

To evaluate the expression `not <exp>`:

1. Evaluate `<exp>`; The value is `True` if the result is a false value, and `False` otherwise.

These values, rules, and operators provide us with a way to combine the results of comparisons. Functions that perform comparisons and return boolean values typically begin with `is`, not followed by an underscore (e.g., `isfinite, isdigit, isinstance,` etc.).

Source: [http://inst.eecs.berkeley.edu/~cs61A/book/chapters/functions.html#conditional-statements](http://inst.eecs.berkeley.edu/~cs61A/book/chapters/functions.html#conditional-statements)