Starting with a relatively simple regular expression for matching integers or floats (**fig. [x\_1]**) we can start to see the way the rx sublanguage operates. There is an evident distinction between the atomic forms, characters, strings, and keywords, and the operators, like non capturing groups, start and end markers and character ranges. This expression shows mainly familiar operators for regular expression users, \*, + [ - ], etc. are all one to one with their standard meanings, while now the verbose character range shortcut [[:digit:]] is just one simple keyword, digit. The names for these remain, however the representation has been simplified.

One of the primary goals of the project is producing an intersection between seamless integration of the Rhombus style with the rx sublanguage and elevated readability, the start of this can be seen in the phone number example **fig. [x\_2]**. A relatively simple regular expression, the language provides a few upsides over its traditional, string-based representation:

"^((?:\\+[[:digit:]] ?)?(?:\\((?:[[:digit:]]{3}\\))|([[:digit:]]{3})) ?[[:digit:]]{3}(?:-)?[[:digit:]]{4})$"

Immediately obvious is the integration of both single and multi-line comments, this feature, inherent to Rhombus already, provides a powerful tool for readability and ease of use. In addition we get to see a further distinction between strings and operators here, traditionally + is a repeater meant to indicate 1 or more matches, however in this case when typing the string “+” matches exactly what is given, a +. Rhombus’s macro extensibility guarantees this explicit distinction between string and operator.

The IP matching example shows off the last and one of the most important parts of the rx sublanguage, the escape operator. In the rx sublanguage binding an identifier with the rx`…` form makes the static information (in this case the right-hand side) available for use within another rx form. This essentially replicates string concatenation and allows for the combination of multiple rx forms for ease of use and reading. By default, these escape forms are wrapped in an implicit non-capturing group such that any operator is applied to the whole of the escaped variable’s value rather than just the last character, class, etc. inside of it.