```
Scheme Interaction mode defined in ?xscheme.el?:
Major mode for interacting with an inferior MIT Scheme process.
Like ?scheme-mode? except that:

C-x C-e sends the expression before point to the Scheme process as input
M-p yanks an expression previously sent to Scheme
M-n yanks an expression more recently sent to Scheme

All output from the Scheme process is written in the Scheme process
buffer, which is initially named "*scheme*".  The result of
evaluating a Scheme expression is also printed in the process buffer,
preceded by the string ";Value: " to highlight it.  If the process
buffer is not visible at that time, the value will also be displayed
in the minibuffer.  If an error occurs, the process buffer will
automatically pop up to show you the error message.

While the Scheme process is running, the mode lines of all buffers in
?scheme-mode? are modified to show the state of the process.  The
possible states and their meanings are:

input           waiting for input
run             evaluating
gc              garbage collecting

The process buffer?s mode line contains additional information where
the buffer?s name is normally displayed: the command interpreter level
and type.


Scheme maintains a stack of command interpreters.  Every time an error
or breakpoint occurs, the current command interpreter is pushed on the
command interpreter stack, and a new command interpreter is started.
One example of why this is done is so that an error that occurs while
you are debugging another error will not destroy the state of the
initial error, allowing you to return to it after the second error has
been fixed.

The command interpreter level indicates how many interpreters are in
the command interpreter stack.  It is initially set to one, and it is
incremented every time that stack is pushed, and decremented every
time it is popped.  The following commands are useful for manipulating
the command interpreter stack:

C-c C-b pushes the stack once
C-c C-u pops the stack once
C-c C-c pops everything off
C-c C-x aborts evaluation, doesn?t affect stack

Some possible command interpreter types and their meanings are:

[Evaluator]     read-eval-print loop for evaluating expressions
[Debugger]      single character commands for debugging errors
[Where]         single character commands for examining environments

Starting with release 6.2 of Scheme, the latter two types of command
interpreters will change the major mode of the Scheme process buffer
to ?scheme-debugger-mode?, in which the evaluation commands are
disabled, and the keys which normally self insert instead send
themselves to the Scheme process.  The command character ? will list
the available commands.
```

```
For older releases of Scheme, the major mode will be
?scheme-interaction-mode?, and the command characters must be sent as
if they were expressions.

Commands:
Delete converts tabs to spaces as it moves back.
Blank lines separate paragraphs.  Semicolons start comments.
key             binding
---             -------

C-c             Prefix Command
C-x             Prefix Command
ESC             Prefix Command
DEL             backward-delete-char-untabify

C-c C-b         xscheme-send-breakpoint-interrupt
C-c C-c         xscheme-send-control-g-interrupt
C-c RET         xscheme-send-current-line
C-c C-o         xscheme-delete-output
C-c C-p         xscheme-send-proceed
C-c C-s         xscheme-select-process-buffer
C-c C-u         xscheme-send-control-u-interrupt
C-c C-x         xscheme-send-control-x-interrupt
C-c C-y         xscheme-yank

C-x C-e         xscheme-send-previous-expression

M-RET           xscheme-send-previous-expression
C-M-q           indent-sexp
C-M-x           xscheme-send-definition
C-M-z           xscheme-send-region
M-n             xscheme-yank-push
M-o             xscheme-send-buffer
M-p             xscheme-yank-pop
M-z             xscheme-send-definition
```