DrRacket: The Racket Programming Environment

Version 9.4.0.3

Robert Bruce Findler
and PLT

November 2, 2026

DrRacket is a graphical environment for developing programs using the Racket programming languages.

Contents

1 Interface Essentials **5**

1.1 Buttons . . . 6

1.2 Choosing a Language . . . 8

1.3 Editing with Parentheses . . . 8

1.4 Searching . . . 10

1.5 Tabbed Editing . . . 10

1.6 The Interactions Window . . . 10

1.6.1 Errors . . . 11

1.6.2 Input and Output . . . 11

1.7 Graphical Syntax . . . 14

1.7.1 Images . . . 15

1.7.2 XML Boxes and Racket Boxes . . . 15

1.8 Graphical Debugging Interface . . . 15

1.8.1 Debugger Buttons . . . 16

1.8.2 Definitions Window Actions . . . 16

1.8.3 Stack View Pane . . . 17

1.8.4 Debugging Multiple Files . . . 18

1.9 The Module Browser . . . 19

1.10 Color Schemes . . . 20

1.11 Creating Executables . . . 21

1.12 Following Log Messages . . . 22

2 Languages **23**

2.1 Language Declared in Source . . . 23

2.1.1 Initial Environment 23
2.1.2 Details Pane of Language Dialog 24
2.2 Legacy Languages 25
2.3 *How to Design Programs* Teaching Languages 25
2.4 Other Experimental Languages 29
2.5 Output Printing Styles 29

3 Interface Reference **31**
3.1 Menus 31
3.1.1 File 31
3.1.2 Edit 32
3.1.3 View 34
3.1.4 Language 35
3.1.5 Racket 36
3.1.6 Insert 37
3.1.7 Windows 38
3.1.8 Help 38
3.1.9 Popup menu 39
3.2 Preferences 39
3.2.1 Font 39
3.2.2 Colors 40
3.2.3 Editing 40
3.2.4 Warnings 43
3.2.5 General 44
3.2.6 Profiling 45
3.2.7 Browser 46

3.2.8 Tools . . . 46
3.3 Keyboard Shortcuts . . . 46
3.3.1 Moving Around . . . 47
3.3.2 Editing Operations . . . 49
3.3.3 File Operations . . . 51
3.3.4 Search . . . 51
3.3.5 Evaluation . . . 51
3.3.6 Documentation . . . 51
3.3.7 Interactions . . . 52
3.3.8 LaTeX and TeX inspired keybindings . . . 52
3.3.9 Defining Custom Shortcuts . . . 55
3.3.10 Sending Program Fragments to the REPL . . . 57
3.4 Status Information . . . 60
3.5 DrRacket Files . . . 61
3.5.1 Program Files . . . 61
3.5.2 Backup and First Change Files . . . 61
3.5.3 Preference Files . . . 62

4 Extending DrRacket **64**

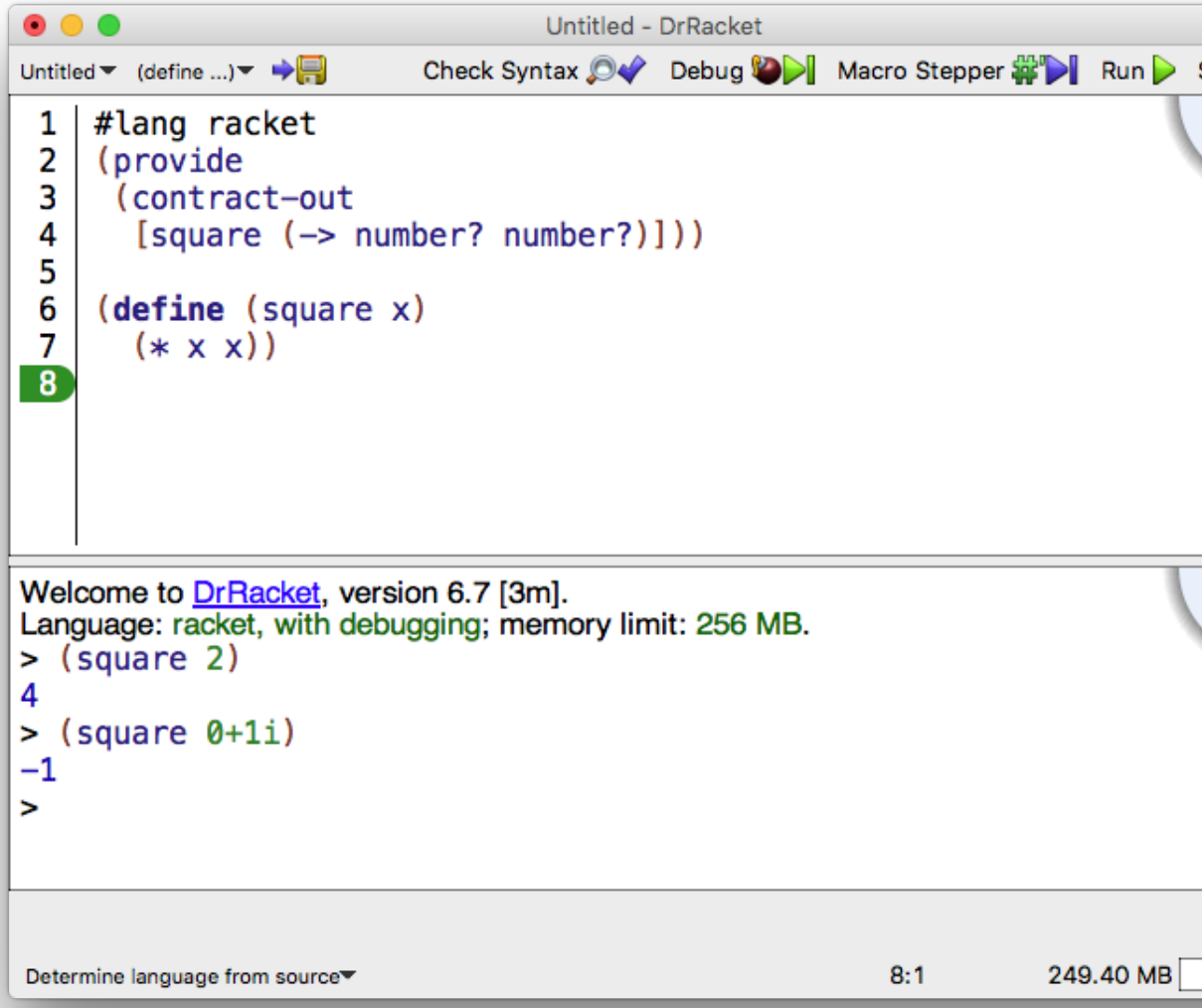
4.1 Teachpacks . . . 64
4.1.1 Adding Your Own Teachpacks to the Teachpack Dialog . . . 66
4.1.2 Extending Help Desk Search Context . . . 66
4.2 Environment Variables . . . 66

Index **69**

Index **69**

1 Interface Essentials

The DrRacket window has three parts: a row of buttons at the top, two editing panels in the middle, and a status line at the bottom.



The top editing panel, called the *definitions window*, is for defining programs. The above figure shows a program that defines the function `square`.

The bottom panel, called the *interactions window*, is for evaluating Racket expressions interactively. The Language line in the interactions window indicates which primitives are available in the definitions and interactions windows. In the above figure, the language is determined from the program source's `#lang` line.

The interactions window is described further in §1.6 "The Interactions Window", later in this manual.

Clicking the Run button evaluates the program in the definitions window, making the program's definitions available in the interactions window. Given the definition of `square` as in the figure above, typing `(square 2)` in the interactions window produces the result `4`. Clicking the Run button again resets the interactions window and re-evaluates the definitions window.

The signature boxes (available via the blue quarter circle in the upper right corner) provide access to the summary information from the documentation. The *status line* at the bottom of DrRacket's window provides information about the current line and position of the editing caret, whether the current file can be modified, and whether DrRacket is currently evaluating any expression. The recycling icon flashes while DrRacket is "recycling" internal resources, such as memory.

1.1 Buttons

The left end of the row of buttons in DrRacket contains a miniature button with the current file's name. Clicking the button opens a menu that shows the file's full pathname. Selecting one of the menu entries produces an open-file dialog starting in the corresponding directory.

Beside the filename button is a (define ...) button for a pop-up menu of names that are defined in the definitions window. Selecting an item from the menu moves the insertion point (blinking caret) to the corresponding definition.

The Save button appears whenever the definitions window is modified. Clicking the button saves the contents of the definitions window to a file. The current name of the file appears to the left of the Save button, but a file-selection dialog appears if the file has never been saved before.

The Step button—which appears only for the *How to Design Programs* teaching languages Beginning Student through Intermediate Student with Lambda—starts the Stepper, which shows the evaluation of a program as a series of small steps. Each evaluation step replaces an expression in the program with an equivalent one using the evaluation rules of DrRacket. For example, a step might replace `(+ 1 2)` with `3`. These are the same rules used by DrRacket to evaluate a program. Clicking Step opens a new window that contains the program from the definitions window, plus several new buttons: these buttons allow navigation of the evaluation as a series of steps.

The debugging interface is described further in §1.8 "Graphical Debugging Interface", later in this manual.

The Debug button—which does *not* appear for the *How to Design Programs* teaching languages—starts a more conventional stepping debugger. It runs the program in the definitions window like the Run button, but also opens a debugging panel with several other buttons that provide control over the program's execution.

Clicking the Check Syntax button annotates the program text in the definitions window. It adds the following annotations:

- **Syntactic Highlighting:** Imported variables and locally defined variables are highlighted with color changes. Documented identifiers are hyperlinked (via a right-click) to the documentation page.
- **Lexical Structure:** The lexical structure is shown with arrows overlaid on the program text. When the mouse cursor passes over a variable, DrRacket draws an arrow from the binding location to the variable, or from the binding location to every bound occurrence of the variable.

  In addition to indicating definite references with blue arrows, DrRacket also draws arrows to indicate potential references within macro definitions. Potential arrows are drawn in purple and annotated with a question mark to indicate uncertainty, because DrRacket cannot predict how such identifiers will eventually be used. Their roles may depend on the arguments to the macro and the context the macro is used in.

  Additionally, right-clicking (or Control-clicking on Mac OS) on a variable activates a pop-up menu that lets you jump from binding location to bound location and vice versa, α-rename the variable, or tack the arrows so they do not disappear.
- **Tail Calls:** Any sub-expression that is (syntactically) in tail-position with respect to its enclosing context is annotated by drawing a light purple arrow from the tail expression to its surrounding expression.
- **Require Annotations:** Right-clicking (or Control-clicking on Mac OS) on the argument to `require` activates a pop-up menu that lets you open the file that contains the `required` module.

  Passing the mouse cursor over a `require` expression inside a module shows all of the variables that are used from that `require` expression. Additionally, if no variables are used from that require expression, it is colored like an unbound variable.

  Finally, passing the mouse cursor over a variable that is imported from a module shows the module that it is imported from in a status line at the bottom of the frame.

Check Syntax also runs automatically as you edit your program, and the bottom, rightmost corner of the DrRacket window shows its status. A red dot means that something has gone wrong; move your mouse over the dot to find out what is wrong.

When nothing goes wrong, the colors indicate the stages processing of the program: blue (expanding), purple (computing check syntax information), orchid (updating the editor with the check syntax information), and green (finished).

Also, right-clicking (or control clicking) in that area yields a menu that lets you disable (or re-enable) automatic Check Syntax.

The Run button evaluates the program in the definitions window and resets the interactions window.

The Break button interrupts an evaluation, or beeps if DrRacket is not evaluating anything. For example, after clicking Run or entering an expression into the interactions window, click Break to cancel the evaluation. Click the Break button once to try to interrupt the evaluation gracefully; click the button twice to kill the evaluation immediately.

1.2 Choosing a Language

DrRacket supports multiple dialects of Racket, as well as some non-Racket languages. You specify a language in one of two ways:

- Select the Use the language declared in the source option (via the Language|Choose Language... menu item), and then specify a specific language as part of the program usually by starting the definitions-window content with `#lang`.
- Select the Language|Choose Language... menu item, and choose a specific language. After changing the language, click Run to reset the language in the interactions window. The bottom-left corner of DrRacket's main window also has a shortcut menu item for selecting previously selected languages.

Using a language declared in a program's source is the recommend mode, and it is described further in §2.1 "Language Declared in Source".

The Language|Choose Language... dialog contains a Show Details button for configuring certain details of the chosen language. Whenever the selected options do not match the default language specification, a Custom indicator appears next to the language-selection control at the top of the dialog.

See §2 "Languages" (later in this manual) for more information on the languages that DrRacket supports.

1.3 Editing with Parentheses

DrRacket's editor provides special support for managing parentheses in a program. When the blinking caret is next to a parenthesis, DrRacket shades the region between the parenthesis and its matching parenthesis. This feature is especially helpful when balancing parentheses to complete an expression. DrRacket also has a highlighting mode that uses shading for nested parentheses, described in §3.2.2 "Colors".

DrRacket also encourages a particular format for code in a language-specific manner. When you type Enter or Return, the editor inserts a new line and automatically indents it. To make DrRacket re-indent an existing line, move the blinking caret to the line and hit the Tab key. (The caret can be anywhere in the line.) You can re-indent an entire region by selecting the region and typing Tab.

DrRacket also rewrites parenthesis as you type them, in order to make them match better. If you type a closing parenthesis `)`, a closing square bracket `]`, or a closing curly brace `}`, and if DrRacket can match it back to some earlier opening parenthesis, bracket, or brace, then DrRacket changes what you type to match. DrRacket also rewrites open square brackets, usually to an open parenthesis. There are some exceptions where opening square brackets are not automatically changed to parentheses:

- If the square bracket is after `cond`-like keyword, potentially skipping some of the sub-expressions in the `cond`-like expression (for example, in a `case` expression, the square brackets start in the second sub-expression).
- If the square bracket begins a new expression immediately after a `local`-like keyword. Note that the second expression after a `local`-like keyword will automatically become an ordinary parenthesis.
- If the square bracket is after a parenthesis that is after a `letrec`-like keyword,
- If the square bracket is in a sequence and the s-expression before in the sequence is a compound expression, DrRacket uses the same kind parenthesis, brace, or bracket as before, or
- If the square bracket is in the middle of string, comment, character, or symbol.

The upshot of DrRacket's help is that you can always use the (presumably unshifted) square brackets on your keyboard to type parenthesis. For example, when typing

```
(define (length l)
  (cond
    [(empty? l) 0]
    [else (+ 1 (length (rest l)))]))
```

If you always type `[` and `]` where any of the square brackets or parentheses appear, DrRacket will change the square brackets to match the code above.

Of course, these features can be disabled and customized in the preferences dialog; see §3.2 "Preferences". Also, in case DrRacket does not produce the character you want, holding down the control key while typing disables DrRacket's parenthesis, brace, and bracket converter.

1.4 Searching

DrRacket's search and replace feature is interactive, similar to those in Safari, Firefox, and Emacs, but with a few differences.

To start a search, first select the Find menu item from the Edit menu. This will open a small editor at the bottom of the DrRacket window. Start typing in there and, as you type, all occurrences of the string you're searching for will be circled in the editor window. Watch the space right next to the search window to see how many occurrences of the search string there are in your file. When you're ready, you use the Find Again menu item to jump to the first occurrence of the search string. This will color in one of the circles. Use Find Again a second time to jump to the next occurrence.

If you click back into the definitions window, the Find Again menu item, DrRacket will move the selection to the next occurrence of the search string.

DrRacket also supports a mode where typing in the search editor takes you directly to the next occurrence of the search string, without selecting the Find Again menu item. In the preference dialog, in the Editing section and then in the General Editing section is a checkbox labeled Search using anchors. When it is checked, DrRacket shows a little red dot and a red line indicating where the *search anchor* is. When the search anchor is enabled, typing in the search window jumps to the first occurrence of the search string after the anchor.

1.5 Tabbed Editing

DrRacket's allows you to edit multiple files in a single window via tabs. The File|New Tab menu item creates a new tab to show a new file. Each tab has its own interactions window.

In the General pane of the the preferences window, a checkbox labeled Open files in separate tabs causes DrRacket to open files in new tabs in the frontmost window, rather than opening a new window for the file.

The key bindings Control-Pageup and Control-Pagedown move between tabs. On Mac OS, Command-Left-Square-Bracket and Command-Right-Square-Bracket also move between tabs.

1.6 The Interactions Window

The interactions window lets you type an expression after the > prompt for immediate evaluation. You cannot modify any text before the last > prompt. To enter an expression, the blinking caret must appear after the last prompt, and also after the space following the prompt.

When you type a complete expression and hit Enter or Return, DrRacket evaluates the ex-

pression and prints the result. After printing the result, DrRacket creates a new prompt for another expression. Some expressions return a special "void" value; DrRacket never prints void, but instead produces a new prompt immediately.

If the expression following the current prompt is incomplete, then DrRacket will not try to evaluate it. In that case, hitting Enter or Return produces a new, auto-indented line. You can force DrRacket to evaluate the expression by typing Alt-Return or Command-Return (depending on your platform).

To copy the previous expression to the current prompt, type ESC-p (i.e., type Escape and then type p). Type ESC-p multiple times to cycle back through old expressions. Type ESC-n to cycle forward through old expressions. There are other keys that have these same functions; see §3.3 "Keyboard Shortcuts" and the menu item Show Active Keybindings menu item in the Edit menu.

Clicking the Run button evaluates the program in the definitions window and makes the program's definitions available in the interactions window. Clicking Run also resets the interactions window, erasing all old interactions and removing old definitions from the interaction environment. Although Run erases old > prompts, ESC-p and ESC-n can still retrieve old expressions.

1.6.1 Errors

Whenever DrRacket encounters an error while evaluating an expression, it prints an error message in the interactions window and highlights the expression that triggered the error. The highlighted expression might be in the definitions window, or it might be after an old prompt in the interactions window.

For certain kinds of errors, DrRacket turns a portion of the error message into a hyperlink. Click the hyperlink to get help regarding a function or keyword related to the error.

For some run-time errors, DrRacket shows a bug icon next to the error message. Click the bug icon to open a window that shows a "stack" of expressions that were being evaluated at the time of the error. In addition, if the expressions in the stack appear in the definitions window, a red arrow is drawn to each expression from the next deeper one in the stack.

1.6.2 Input and Output

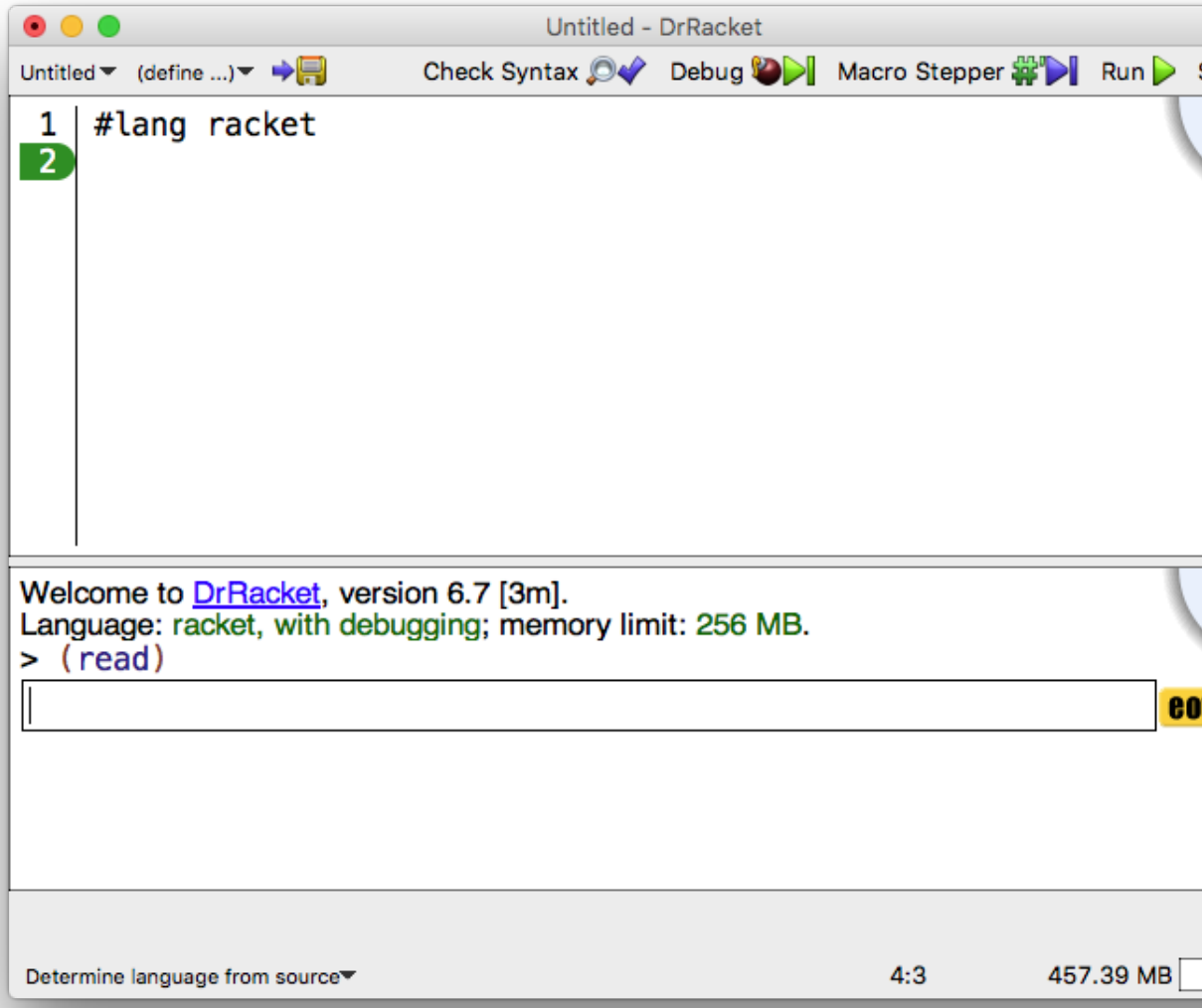
Many Racket programs avoid explicit input and output operations, obtaining input via direct function calls in the interactions window, and producing output by returning values. Other Racket programs explicitly print output for the user during evaluation using `write` or `display`, or explicitly request input from the user using `read` or `read-char`.

Explicit input and output appear in the interactions window, but within special boxes that

separate explicit I/O from normal expressions and results. For example, evaluating

```
> (read)
```

in the interactions window produces a special box for entering input:



Type a number into the box and hit Enter, and that number becomes the result of the `(read)` expression. Once text is submitted for an input box, it is moved outside the input box, and when DrRacket shows a new prompt, it hides the interaction box. Thus, if you type `5` in the above input box and hit Return, the result appears as follows:

```
> (read)
5
5
> _
```

In this case, the first `5` is the input, and the second `5` is the result of the `(read)` expression. The second `5` is colored blue, as usual for a result printed by DrRacket. (The underscore indicates the location of the blinking caret.)

Output goes into the interactions window directly. If you run the program

```
#lang racket
(define v (read))
(display v) (newline)
v
```

and provide the input S-expression `(1 2)`, the interactions window ultimately appears as follows:

```
(1 2)
(1 2)
'(1 2)
> _
```

In this example, `display` produces output immediately beneath the input you typed, and the final result is printed last. The displayed output is drawn in purple. (The above example assumes default printing. With constructor-style value printing, the final before the prompt would be `(list 1 2)`.)

Entering the same program line-by-line in the interactions window produces a different-looking result:

```
> (define v (read))
(1 2)
> (display v)
(1 2)
> v
'(1 2)
> _
```

Depending on the input operation, you may enter more text into an input box than is consumed. In that case, the leftover text remains in the input stream for later reads. For example, in the following interaction, two values are provided in response to the first `(read)`, so the second value is returned immediately for the second `(read)`:

```
> (read)
5 6
5
> (read)
6
> _
```

The following example illustrates that submitting input with Return inserts a newline character into the input stream:

```
> (read)
5
5
> (read-char)
#\newline
> _
```

The eof button that appears beside an input box inserts a single `eof-object` into the input stream, but more IO may follow in a later sequence. For example, in the following interaction, the user typed `1` and then clicked the eof button:

```
> (read-char)
1#\1
> (read-char)
#<eof>
```

At this point, however, future interactions can still take place: new calls to input functions with open a new input box and new characters will come from the same port.

Within a `#lang racket` module, the results of top-level expression print the same as the results of an expression entered in the interactions window. The reason is that `#lang racket` explicitly prints the results of top-level expressions using `(current-print)`, and DrRacket sets `(current-print)` to print values in the same way as for interactions.

1.7 Graphical Syntax

In addition to normal textual program, DrRacket supports certain graphical elements as expressions within a program. Plug-in tools can extend the available graphical syntax, but this section describes some of the more commonly used elements.

1.7.1 Images

DrRacket's Insert|Insert Image... menu item lets you select an image file from disk (in various formats such as GIF, PNG, and BMP), and the image is inserted at the current editing caret.

As an expression an image behaves like a number or string constant: it evaluates to itself. DrRacket's interactions window knows how to draw image-value results or images displayed via `print`.

A program can manipulate image values in various ways, such as using the `htdp/image` library or as an `image-snip%` value.

1.7.2 XML Boxes and Racket Boxes

DrRacket has special support for XML concrete syntax. The Special|Insert XML Box menu item inserts an embedded editor into your program. In that embedded editor, you type XML's concrete syntax. When a program containing an XML box is evaluated, the XML box is translated into an *x-expression* (or *xexpr*), which is an s-expression representation of an XML expression. Each xexpr is a list whose first element is a symbol naming the tag, second element is an association list representing attributes and remaining elements are the nested XML expressions.

XML boxes have two modes for handling whitespace. In one mode, all whitespace is left intact in the resulting xexpr. In the other mode, any tag that only contains nested XML expressions and whitespace has the whitespace removed. You can toggle between these modes by right-clicking or Control-clicking (Mac OS) on the top portion of the XML box.

In addition to containing XML text, XML boxes can also contain Racket boxes. Racket boxes contain Racket expressions. These expressions are evaluated and their contents are placed into the containing XML box's xexpr. There are two varieties of Racket box: the standard Racket box and the splicing Racket box. The standard Racket box inserts its value into the containing xexpr. The contents of the splice box must evaluate to a list and the elements of the list are "flattened" into the containing xexpr. Right-clicking or control-clicking (Mac OS) on the top of a Racket box opens a menu to toggle the box between a Racket box and a Racket splice box.

1.8 Graphical Debugging Interface

Like the Run button, the Debug button runs the program in the definitions window. However, instead of simply running it from start to finish, it lets users control and observe the program as it executes. The interface includes a panel of buttons above the definitions window, as well as extensions to the definitions window itself.

Tip: Changing the name of a file in the middle of a debugging session will prevent the debugger from working properly on that file.

The program starts out paused just before the first expression is executed. This is indicated in the definitions window by the presence of a green triangle over this expression's left parenthesis.

1.8.1 Debugger Buttons

While execution is paused, several buttons are available:

- The Go button is enabled whenever the program is paused. It causes the program to resume until it either completes, reaches a breakpoint, or raises an unhandled exception.
- The Step button is enabled whenever the program is paused. It causes the program to make a single step and then pause again.
- The Over button is only enabled when execution is paused at the start of an expression that is not in tail position. It sets a one-time breakpoint at the end of the expression (represented by a yellow circle) and causes the program to proceed. When execution reaches the one-time breakpoint, it pauses and removes that breakpoint.
- The Out button is only enabled when execution is paused within the context of another expression. Like the Over button, it sets a one-time breakpoint and continues execution. In this case, the program stops upon returning to the context or raising an unhandled exception.

If the program is running (not paused), then only the Pause button will be enabled. Clicking it will interrupt execution and pause it. In this case, the current expression may only be known approximately, and it will be represented as a gray triangle. The other features described above will still be available.

At any time, execution may be interrupted by clicking the Stop button. However, unlike with the Pause button, stopped execution cannot be continued.

1.8.2 Definitions Window Actions

When execution is paused, the definitions window supports several additional actions:

- Hovering the mouse cursor over a parenthesis may reveal a pink circle. If so, right-clicking or control-clicking (Mac OS) will open a menu with options to Pause at this point or Continue to this point. The former sets an ordinary breakpoint at that location; the latter sets a one-time breakpoint and resumes execution. An ordinary breakpoint appears as a red circle, and a one-time breakpoint appears as a yellow circle.

Tip: If the debugged program is not a module, then the *first time* it is debugged, breakpoints will only become available in expressions as they are evaluated. However, the next time the program is debugged, the debugger will remember the set of breakable locations from the previous session.

Tip: Clicking the Run button after a debugging session will cause all breakpoints to disappear from the definitions window. These breakpoints are not forgotten, and clicking Debug again will restore them. However, breakpoints do *not* persist across restarts of DrRacket.

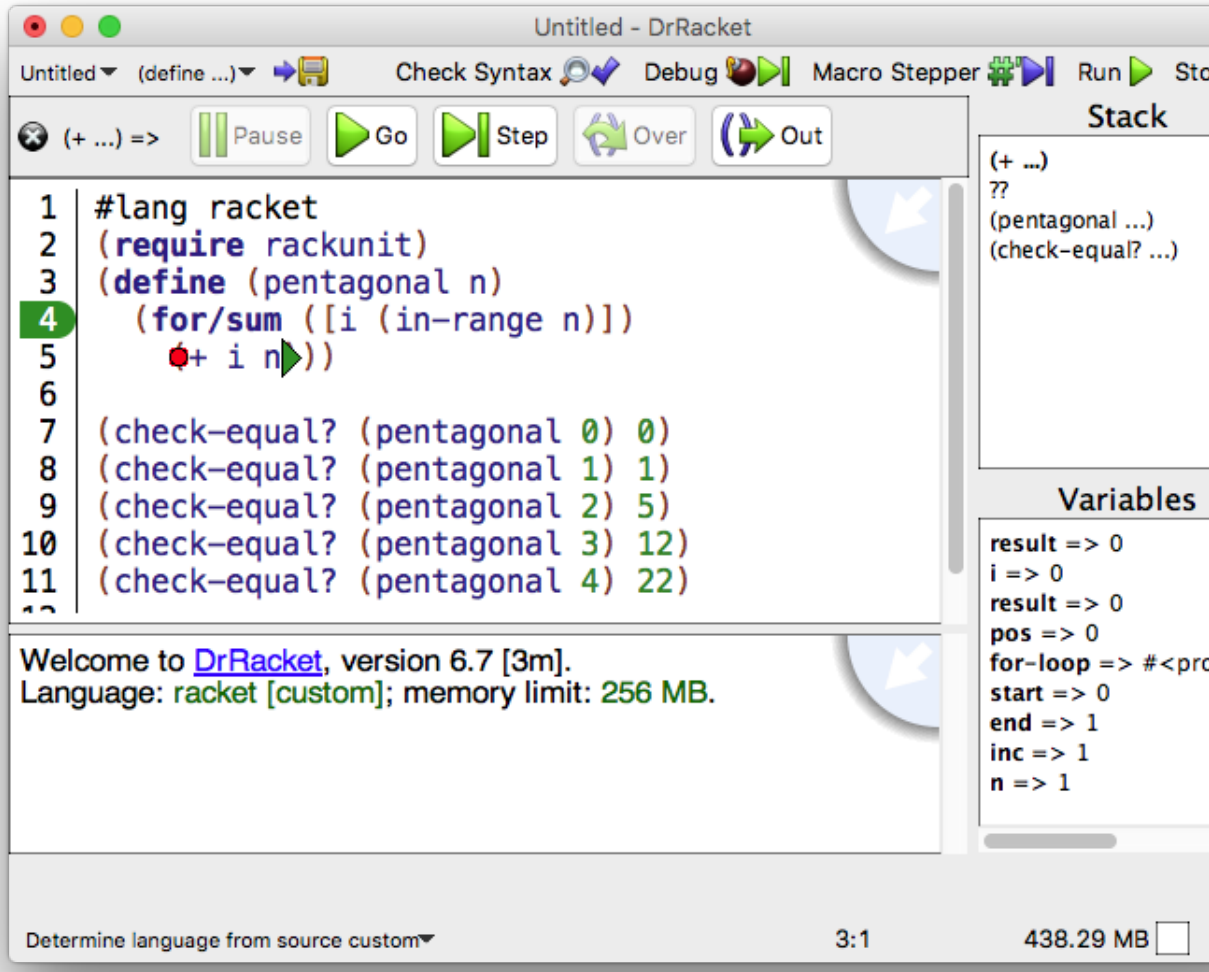
- If execution is paused at the start of an expression, then right-clicking or control-clicking (Mac OS) on the green triangle opens a menu with the option to Skip expression.... Selecting this opens a text box in which to enter a value for the expression. The expression is skipped, with the entered value substituted for it.
- If execution is paused at the end of an expression, then the expression and its value are displayed to the left of the button bar. Right-clicking or control-clicking (Mac OS) on the green triangle opens a menu with options to Print return value to console and Change return value.... The former displays the return value in the interactions window; the latter opens a text box in which to enter a substitute value.
- Hovering the mouse cursor over a bound variable displays the variable's name and value to the right of the button bar. Right-clicking or control-clicking (Mac OS) opens a menu with options to Print value of <var> to console or (set! <var> ...). The former displays the variable's value in the interactions window; the latter opens a text box in which to enter a new value for the variable.

1.8.3 Stack View Pane

In addition, while execution is paused, the stack view pane at the right of the DrRacket frame is active. The top of the pane shows a list of active stack frames. Mousing over a frame produces a faint green highlighting of the corresponding expression. Clicking on the frame selects that frame, which makes its lexical variables visible. The selected frame is indicated by a bold font.

The bottom of the pane shows the lexical variables in the selected stack frame.

The following screenshot illustrates several aspects of the debugger interface. The red circle in the body of `for/sum` is a breakpoint, and the green triangle at the end of the `(+ i n)` is where execution is currently paused. The expression's return value is displayed at the left of the button bar, and the value of `i` and other variables are displayed in the stack view pane.



1.8.4 Debugging Multiple Files

To debug a program that spans several files, make sure that all of the files are open in DrRacket. Click the Debug button in the window containing the main program. As this program loads additional files that are present in other windows or tabs, message boxes will pop up asking whether or not to include the file in the debugging session. Including the file

means that it will be possible to set breakpoints, inspect variables, and single-step in that file.

Tip: A file may only be involved in one debugging session at a time. If you try to debug a file that loads another file that is already being debugged, a message box will pop up explaining that the file cannot be included in another debugging session.

1.9 The Module Browser

The module browser shows you the structure of all of the files in your program. It can be opened via the Show menu, or via the Module Browser menu items in the Racket menu. It is also available from the command-line via `raco dependencies-graph`.

A module browser window contains a square for each module. The squares are colored based on the number of lines of code in the module. If a module has more lines of code, it gets a darker color. If a module is red, it means that DrRacket did not find a source file for it.

In addition, for each normal import, a blue line drawn is from the module to the importing module. Similarly, purple lines are drawn for each for-syntax, for-template or for-meta import. In the initial module layout in the standalone module browser window, modules to the left import modules to the right. When the winow is embedded in the DrRacket window, module above import those below. (Since modules can be moved around interactively, that property might not be preserved.)

To open the file corresponding to the module, double click on the box for that module.

The module browser will also show you the phases that each module is loaded in; choose the Long, with phases menu item in the Names pop-up menu. The integers indicate the phases and if `#f` is present, it means the module is loaded `for-label`.

In the standalone window view, there is a bar along the bottom that helps you find your way in a module graph. Specifically, if you type something there, then all of the modules whose filenames match what you type will turn green in the module window. This bar is only visible in the stand alone module browser window (via the Racket menu or `raco depedencies-graph`).

The Visible Packages menu controls which subset of the modules are visible. To start, all of the modules in the same package as the package of the initial file are shown.

The Visible Submodules menu also controls which subset of the modules are visible. To start, all of the modules that are not submodules are shown. Use this menu to add in additional submodules.

1.10 Color Schemes

DrRacket comes with a selection of color schemes, available in the preferences dialog's color panel.

You can add your own color schemes to DrRacket, too. The first step is to create a package (see §1.7 "Creating Packages") and add an `"info.rkt"` file to it. The file should define `framework:color-schemes` as a list of hashes that describe the color schemes.

As an example, this is the specification of the `"Modern"` style:

```
#lang info
(define framework:color-schemes
  '(#hash((colors
           .
           ((framework:syntax-color:scheme:string
             #(211 72 255))
            (framework:syntax-color:scheme:constant
             #(211 72 255))
            (framework:syntax-color:scheme:comment
             #(194 158 31))
            (framework:syntax-color:scheme:parenthesis
             #(0 150 255))))
          (name . "Modern"))))
```

Each of the keys, e.g., `'framework:syntax-color:scheme:string`, maps to a color and possibly to some style information. All keys accept colors (the vectors shown above represent colors in red–green–blue format), but only some accept style information. To find out which are which and to get a complete list of the possible keys, click the button labeled Style & Color Names at the bottom of the Color Schemes tab of the Colors tab in the preferences dialog. If one can accept style information, then you may include any of `'bold`, `'underline`, `'italic` or `` `#s(background ,color) `` in the list with the color.

Full details on the specification of the info files can be found in the documentation for the function `color-prefs:register-info-based-color-schemes`.

You may have to restart DrRacket (and, at least the first time after you add the `"info.rkt"` file, re-run `raco setup`) to see changes to your color scheme.

Color schemes are not limited only to the colors that DrRacket already knows about. If you are adding your own plugin to DrRacket, you can add new names that can be mapped in the color scheme. See `color-prefs:register-color-preference` for more information.

1.11 Creating Executables

DrRacket's Create Executable... menu item lets you create an executable for your program that you can start without first starting DrRacket. To create an executable, first save your program to a file and set the language and teachpacks. Click Run, just to make sure that the program is working as you expect. The executable you create will not have a read-eval-print-loop, so be sure to have an expression that starts your program running in the definitions window before creating the executable.

Once you are satisfied with your program, choose the Create Executable... menu item from the Racket menu. You will be asked to choose an executable file name or an archive file name. In the latter case, unpack the generated archive (on this machine or another one) to access the executable. In either case, you will be able to start the executable in the same way that you start any other program on your computer.

The result of Create Executable... is either a *launcher executable*, a *stand-alone executable*, or a *distribution archive*, and it uses either a *Racket* (textual) or *GRacket* (graphical) engine. For programs implemented with certain languages, Create Executable... will prompt you to choose the executable type and engine, while other languages support only one type or engine.

Each type has advantages and disadvantages:

- A *launcher executable* uses the latest version of your program source file when it starts. It also accesses library files from your DrRacket installation when it runs. Since a launcher executable contains specific paths to access those files, launchers usually cannot be moved from one machine to another.
- A *stand-alone executable* embeds a compiled copy of your program and any Racket libraries that your program uses. When the executable is started, it uses the embedded copies and does not need your original source file. It may, however, access your DrRacket installation for DLLs, frameworks, shared libraries, or helper executables. Consequently, a stand-alone executable usually cannot be moved from one machine to another.
- A *distribution archive* packages a stand-alone executable together with any needed DLLs, frameworks, shared libraries, and helper executables. A distribution archive can be unpacked and run on any machine with the same operating system as yours.

DrRacket's gives you the most options when it infers a language from a program's source. Most other languages only allow one type of executable. The teaching languages, for example, create stand-alone executables in distributions. The legacy languages create only launchers.

Tip: Disable debugging in the language dialog before creating your executable. With debugging enabled, you will see a stack trace with error messages, but your program will run

more slowly. To disable debugging, open the language dialog, click the Show Details button, and select No debugging or profiling, if it is available.

When you create an executable in some languages, you can supply additional files to determine the executable's icon and similar properties, depending on the platform. The file's purpose is determined by its suffix:

- On Windows, supply an `".ico"` file for an icon. Only 16x16, 32x32, or 48x48 images from the `".ico"` file are used.
- On Mac OS, supply an `".icns"` file for an icon. You can set the application's creator with an `".creator"` file (whose first four bytes are used), and you can set documents for the application through a `".utiexports"` file (see `'uti-exports` in `create-embedding-executable` for more information).
- On Unix, supply a `".png"` or `".ico"` file for an icon.

1.12 Following Log Messages

The Show Log menu item in the View menu opens a pane in the DrRacket window showing log messages.

Along the top of the window is a text field that should be filled with a description of which log messages are interesting, as described in the §15.5 "Logging" section of *The Racket Reference*.

2 Languages

This chapter describes some of the languages that are available for use within DrRacket. The list here is potentially incomplete, because new languages can be added through DrRacket plug-ins.

2.1 Language Declared in Source

The The Racket Language mode in DrRacket is a kind of meta-language, where the program itself specifies its language, usually through a `#lang` line.

More generally, when using this mode, the definitions window must contain a module in some form. Besides `#lang`, a Racket module can be written as `(module ...)`; aside from comments, the definitions window must contain exactly one module.

2.1.1 Initial Environment

DrRacket initializes the environment before it starts running the program in ways slightly different than the `racket` command-line binary does to reflect the extra GUI capabilities that it provides. Using the default settings in the language dialog, it initializes the parameters described in this subsection. Changing the defaults can result in additional parameters changing, as described in the next subsection.

DrRacket initializes the `pretty-print-size-hook` and `pretty-print-print-hook` to specially recognize

- `snip%`s, which it just inserts into the editor
- convertible values from `pict/convert`, which are rendered with padding specified by `convert-bounds-padding` around the bounding box to reduce edge effects
- convertible values from `file/convertible` (which it converts to png images)
- numbers that are `exact?` and `real?`, but not `integer?`, which it prints specially to allow showing repeating decimal representations and mixed fractions, and
- values that match the predicates passed to `drracket:language:add-snip-value`.

It also initializes the `global-port-print-handler` to use `pretty-print` instead of just `print`.

DrRacket initializes the `error-display-handler` to specially highlight the source locations of the errors and to include icons that provide access to the stacktrace.

2.1.2 Details Pane of Language Dialog

In the details pane of the language dialog, some of the configuration options correspond to using various libraries and thus can be used without DrRacket. Here's how, for the ones that are straightforward (the ones not mentioned here require more sophisticated configuration of various libraries).

- **Dynamic Properties**: The radio buttons coresond to various uses of the `errortrace/errortrace-lib` library.

  The *No Debugging or profiling* option means not to use the library at all.

  The *Debugging* option means `(current-compile (make-errortrace-compile-handler))` as well as adding `(build-path "compiled" "errortrace")` to `use-compiled-file-paths`.

  The *Debugging and profiling* option means to use `errortrace/errortrace-lib` library's `profiling-enabled` in conjunction with `current-eval`.

  The *Syntactic test suite coverage* option means to use `test-coverage-enabled` in conjunction with `current-eval`.

  The *Use the #lang's default choice for instrumentation* option means to use one of the previous options, as specified by the `'drracket:default-instrumentation` key from `read-language`.

  The other three checkboxes save compiled `.zo` files and adjust the compiler.

  The *populate compiled/ directories* option corresponds to

```
(current-load/use-compiled
 (make-compilation-manager-load/use-compiled-handler))
(manager-skip-file-handler
 (λ (p)
   (file-date-in-paths
    p
    (cons (CACHE-DIR) (current-library-collection-paths)))))
```

  plus adding either `(build-path "compiled" "drracket")` or `(build-path "compiled" "drracket" "errortrace")` to the front of `use-compiled-file-paths`, depending if the *Debugging* option is set or not.

  The *Preserve stacktrace* option corresponds to

```
(compile-context-preservation-enabled #t)
```

  The *Enforce constant definitions (enables some inlining)* option corresponds to calling `compile-enforce-module-constants`; checking it passes `#t` and leaving it unchecked passes `#f`.

  The *Run program in separate process* option uses a separate OS-level process to run the program in the definitions window when clicking Run. Starting the program is

slower in this mode, but it provides additional protections when developing unsafe libraries (e.g., when using the FFI).

- **Output Syntax**: The output syntax options correspond to settings in the `racket/pretty` library and the `mzlib/pconvert` library.
- **Collection Paths**: This corresponds to setting the `current-library-collection-paths` parameter.
- **Command-line arguments**: This corresponds to setting the `current-command-line-arguments` parameter.
- **Automatic #lang line**: This panel controls the default `#lang` that DrRacket inserts into new windows (or tabs). If the top option of the radio box is chosen (the default), DrRacket uses the last `#lang` that was edited in a definitions window. If the lower option of the radio box is chosen, DrRacket always uses the same `#lang` line, as given in the text field.

2.2 Legacy Languages

DrRacket supports several historically useful variants of Scheme without a `#lang` prefix:

- The R5RS language contains those primitives and syntax defined in the R^{5}RS Scheme standard. See the `r5rs` library for details.
- The *PLT Pretty Big* language provides a language roughly compatible with a language in earlier versions of DrRacket. It evaluates a program in the same way as `load`, and it starts by importing the following modules: `mzscheme`, `racket/gui/base`, `mzlib/class`, `mzlib/etc`, `mzlib/file`, `mzlib/list`, `mzlib/unit`, `mzlib/include`, `mzlib/defmacro`, `mzlib/pretty`, `mzlib/string`, `mzlib/thread`, `mzlib/math`, `mzlib/match`, and `mzlib/shared`.
- The Swindle language starts with the same bindings as `swindle`, and evaluates the program like `load`.

2.3 *How to Design Programs* Teaching Languages

Five of DrRacket's languages are specifically designed for teaching:

- The Beginning Student language is a small version of Racket that is tailored for beginning computer science students.
- The Beginning Student with List Abbreviations languages is an extension to Beginning Student that prints lists with `list` instead of `cons`, and accepts `quasiquoted` input.

- The Intermediate Student language adds local bindings and higher-order functions.
- The Intermediate Student with Lambda language adds anonymous functions.
- The Advanced Student language adds mutable state.

The teaching languages are different from conventional Racket in a number of ways:

- *Case-sensitive identifiers and symbols* — In a case-sensitive language, the variable names `x` and `X` are distinct, and the symbols `'x` and `'X` are also distinct. In a case-insensitive language, `x` and `X` are equivalent and `'x` and `'X` represent the same value. The teaching languages are case-sensitive by default. Case-sensitivity can be adjusted through the detail section of the language-selection dialog.
- *All numbers are exact unless `#i` is specified* — In the Beginning Student through Intermediate Student with Lambda languages, numbers containing a decimal point are interpreted as exact numbers. This interpretation allows students to use familiar decimal notation without inadvertently triggering inexact arithmetic. Exact numbers with decimal representations are also printed in decimal. Inexact inputs and results are explicitly marked with `#i`.
- *Procedures must take at least one argument* — In the Beginning Student through Intermediate Student languages, defined procedures must consume at least one argument. Since the languages have no side-effects, zero-argument functions are not useful, and rejecting such function definitions helps detect confusing syntactic mistakes.
- *Identifier required at function call position* — In the Beginning Student through Intermediate Student languages, procedure calls must be of the form `(identifier ...)`. This restriction helps detect confusing misuses of parentheses, such as `(1)` or `((+ 3 4))`, which is a common mistake among beginners who are used to the optional parentheses of algebra.
- *Top-level required at function call position* — In the Beginning Student languages, procedure calls must be of the form `(top-level-identifier ...)`, and the number of actual arguments must match the number of formal arguments if `top-level-identifier` is `defined`. This restriction helps detect confusing misuses of parentheses, such as `(x)` where `x` is a function argument. DrRacket can detect such mistakes syntactically because Beginning Student does not support higher-order procedures.
- *Primitive and defined functions allowed only in function call position* — In Beginning Student languages, the name of a primitive operator or of a defined function can be used only after the open-parenthesis of a function call (except where teachpack extensions allow otherwise, as in the `convert-gui` teachpack). Incorrect uses of primitives trigger a syntax error. Incorrect uses of defined names trigger a run-time error. DrRacket can detect such mistakes because Beginning Student does not support higher-order procedures.

- *`lambda` allowed only in definitions* — In the Beginning Student through Intermediate Student languages, `lambda` may appear only in a definition, and only as the value of the defined variable.

- *Free variables are not allowed* — In the Beginning Student through Advanced Student languages, every variable referenced in the definitions window must be defined, pre-defined, or the name of a local function argument.

- *`quote` works only on symbols, `quasiquote` disallowed* — In the Beginning Student language, `quote` and `'` can specify only symbols. This restriction avoids the need to explain to beginners why `1` and `'1` are equivalent in standard Racket. In addition, `quasiquote`, `` ` ``, `unquote`, `,`, `unquote-splicing`, and `,@` are disallowed.

- *Unmatched `cond`/`case` is an error* — In the Beginning Student through Advanced Student languages, falling through a `cond` or `case` expression without matching a clause signals a run-time error. This convention helps detect syntactic and logical errors in programs.

- *Conditional values must be `true` or `false`* — In the Beginning Student through Advanced Student languages, an expression whose value is treated as a boolean must return an actual boolean, `true` or `false`. This restriction, which applies to `if`, `cond`, `and`, and `or` expressions, helps detect errors where a boolean function application is omitted.

- *`+`, `*`, and `/` take at least two arguments in Beginning languages.* — In the Beginning Student and Beginning Student with List Abbreviations languages, mathematical operators that are infix in algebra notation require at least two arguments in DrRacket. This restriction helps detect missing arguments to an operator.

- *`and`, `or`, `nand`, and `nor` require at least 2 expressions* — In the Beginning Student through Advanced Student languages, the boolean combination forms require at least two sub-expressions. This restriction helps detect missing or ill-formed sub-expressions in a boolean expression.

- *`set!` disallowed on arguments* — In the Advanced Student language, `set!` cannot be used to mutate variables bound by `lambda`. This restriction ensures that the substitution model of function application is consistent with DrRacket's evaluation.

- *Improper lists disallowed* — A *proper list* is either an empty list or a list created by `cons`ing onto a proper list. In the Beginning Student through Advanced Student languages, `cons` constructs only *proper lists*, signaling an error if the second argument is not a proper list. Since beginning students do not need improper lists, this restriction help detect logical errors in recursive functions.

- *Dot is disallowed* — In the Beginning Student through Advanced Student languages, a delimited period `.` is disallowed, (e.g., as an improper-list constructor in a quoted form, or for defining multi-arity procedures).

- *Syntactic form names disallowed as variable names* — In the Beginning Student through Advanced Student languages, all syntactic form names are keywords that cannot be used as variable names.
- *Re-definitions are disallowed* — In the Beginning Student through Advanced Student languages, top-level names can never be re-defined.
- *Function definitions are allowed only in the definitions window* — In the Beginning Student languages, function definitions are not allowed in the interactions window.

The teaching languages also deviate from traditional Racket in printing values. Different printing formats can be selected for any language through the detail section of language-selection dialog.

- *Constructor-style output* — See §2.5 "Output Printing Styles".
- *Quasiquote-style output* — See §2.5 "Output Printing Styles".
- *Rational number printing* – In the teaching languages, all numbers that have a finite decimal expansion are printed in decimal form. For those numbers that do not have a finite decimal expansion (such as `4/3`) DrRacket offers a choice. It either prints them as mixed fractions or as repeating decimals, where the repeating portion of the decimal expansion is shown with an overbar. In addition, DrRacket only shows the first 25 digits of the number's decimal expansion. If there are more digits, the number appears with an ellipses at the end. Click the ellipses to see the next 25 digits of the expansion.

  This setting controls only the initial display of a number. Right-clicking or Control-clicking (Mac OS) on the number lets you change from the fraction representation to the decimal representation.
- *`write` output* — Prints values with `write`.
- *Show sharing in values* — Prints interaction results using the `shared` syntax, which exposes shared structure within a value. For example, the list created by `(let ([lt (list 0)]) (list lt lt))` prints as

  ```
  (shared ((-1- (list 0))) (list -1- -1-))
  ```

  instead of

  ```
  (list (list 0) (list 0))
  ```

A program in the teaching languages should be tested using the check forms — `(check-expect value value)`, `(check-within value value value)`, or `(check-error value string)`. Tests are evaluated when running the program: when there are no

tests, a warning appears in the interactions window; when all tests succeed, an acknowledgement appears in the interactions window; otherwise, a testing window appears to report the results. See §3.1.3 “View” for details on configuring the report behavior.

Tests can be disabled if necessary, see §3.1.5 “Racket” for details.

One final difference between these teaching languages and other languages is the way they save files. That is, when DrRacket saves a file and the current language is one of these five teaching languages, it inserts three lines of metadata that record the precise language (including any options set) and the teachpacks. This has two benefits: opening the file later restores the settings and the metadata is formulated in such a way as to be executable code so running `racket` or `gracket` on the file in a shell will run the program in the appropriate language.

This meta data always consists of exactly three lines, and so can be stripped out by three calls to `read-line`.

2.4 Other Experimental Languages

For information on Lazy Racket, see the Lazy Racket documentation.

For information on FrTime, see the FrTime documentation.

For information on Algol 60, see the Algol 60 documentation.

2.5 Output Printing Styles

Many Racket languages support a Output Syntax choice that determines how evaluation results are printed in the interactions window. This setting also applies to output generated by calling `print` explicitly.

The print style is the normal Racket output style. The following table illustrates the other output styles:

Input expression	Constructor	Quasiquote	write
`(cons 1 2)`	`(cons 1 2)`	`` `(1 . 2) ``	`(1 . 2)`
`(list 1 2)`	`(list 1 2)`	`` `(1 2) ``	`(1 2)`
`'(1 2)`	`(list 1 2)`	`` `(1 2) ``	`(1 2)`
`(list (void))`	`(list (void))`	`` `(,(void)) ``	`(#<void>)`
`` `(,(void)) ``	`(list (void))`	`` `(,(void)) ``	`(#<void>)`
`(vector 1 2 3)`	`(vector 1 2 3)`	`(vector 1 2 3)`	`#(1 2 3)`
`(box 1)`	`(box 1)`	`(box 1)`	`#&1`
`(lambda (x) x)`	`(lambda (a1) ...)`	`(lambda (a1) ...)`	`#<procedure>`
`'sym`	`'sym`	`'sym`	`sym`

```
(make-s 1 2)    (make-s 1 2)      (make-s 1 2)      #(struct:s 1 2)
'()             empty             `()               ()
add1            add1              add1              #<procedure:add1>
(delay 1)       (delay ...)       (delay ...)       #<promise>
(regexp "a")    (regexp "a")      (regexp "a")      #rx"a"
```

The Constructor output mode is similar to Rackets normal print mode, except that even quotable are still printed with constructors, constructor functions and forms are used to approximate some otherwise unprintable values. For example, Constructor output prints a procedure in a `lambda` form. For output to a graphical context, rational numbers are printed using a special `snip%` object that lets the user choose between improper fractions, mixed fractions, and repeating decimals.

The Quasiquote output mode is like Constructor output, but it uses `quasiquote` (abbreviated with `` ` ``) to print lists, and it uses `unquote` (abbreviated with `,`) to escape back to Constructor printing as needed.

The write output mode corresponds to traditional Scheme printing via the `write` procedure. For example, lists print by parenthesizing the printed form of the list elements, without a leading quote mark or a constructor name.

The print output mode corresponds to Racket's default printing via the `print` procedure. Output via `print` is further configurable through run-time settings, such as the `print-as-expression` parameter, and it may be adjusted by a `#lang`-specified language. For example, the `scheme` language sets the `print-as-expression` parameter to `#f`, which essentially makes print mode act like write mode.

The Constant Style option, when present, controls how `true`, `false`, and `empty` print; for the booleans, it determines if there is a `#` prefix and for the empty list, determines if it prints as `empty` or `'()`.

For any of the output styles, DrRacket sets the `global-port-print-handler` (as described in §2.1.1 "Initial Environment") so that the `print` procedure produces output as selected.

3 Interface Reference

3.1 Menus

3.1.1 File

- New : Creates a new DrRacket window.
- New Tab : Creates a new tab in the current DrRacket window.
- Open... : Opens a find-file dialog for choosing a file to load into a definitions window.
- Open Recent : Lists recently opened files. Choosing one of them opens that file for editing.
- Open Require Path... : Opens a dialog where you can enter in a `require`-like module path (e.g., `racket/base.rkt` or `data/splay-tree.rkt` or `"x.rkt"`) and edit the corresponding files in the definitions window.
- Reopen Closed Tab : Opens previously closed tabs. If no tabs were closed in current session, files from previous sessions are opened.
- Install Package... : Opens a dialog asking for the name of a package to install. See *Package Management in Racket* for more information on packages.
- Package Manager... : Provides a graphical interface to the command-line tool `raco pkg`.
- Install PLT File... : Opens a dialog asking for the location of the `".plt"` file (either on the local disk or on the web) and installs the contents of the file.
- Revert : Re-loads the file that is currently in the definitions window. All changes since the file was last saved will be lost.
- Save Definitions : Saves the program in the definitions window. If the program has never been saved before, a save-file dialog appears.
- Save Definitions As... : Opens a save-file dialog for choosing a destination file to save the program in the definitions window. Subsequent saves write to the newly-selected file.
- Save Other : Contains these sub-items
  - Save Definitions As Text... : Like Save Definitions As..., but the file is saved in plain-text format (see §3.5.1 "Program Files"). Subsequent saves also write in plain-text format.
  - Save Interactions : Saves the contents of the interactions window to a file. If the interaction constants have never been saved before, a save-file dialog appears.

  - Save Interactions As... : Opens a save-file dialog for choosing a destination file to save the contents of the interactions window. Subsequent saves write to the newly-selected file.
  - Save Interactions As Text... : Like Save Interactions As..., but the file is saved in plain-text format (see §3.5.1 "Program Files"). Subsequent saves are write in plain-text format.

- Log Definitions and Interactions... : Starts a running of log of the text in the interactions and definitions windows, organized by executions. In a directory of your choosing, DrRacket saves files with the names `"01-definitions"`, `"01-interactions"`, `"02-definitions"`, `"02-interactions"`, etc. as you interact with various programs.
- Page Setup... : Opens a dialog box to configure various details for printing.
- Print Definitions... : Opens a dialog for printing the current program in the definitions window.
- Print Interactions... : Opens a dialog for printing the contents of the interactions window.
- Search in Files... : Opens a dialog where you can specify the parameters of a multi-file search. The results of the search are displayed in a separate window.
- Close Window : Closes this DrRacket window. If this window is the only open DrRacket window, then DrRacket quits, except on Mac OS.
- Close Tab : Closes the active tab in this DrRacket window.
- {Quit or Exit} Exits DrRacket. (On Mac OS, this menu item is in the Apple menu.)

3.1.2 Edit

All Edit menu items operate on either the definitions or interactions window, depending on the location of the selection or blinking caret. Each window maintains its own Undo and Redo history.

- Undo : Reverses an editing action. Each window maintains a history of actions, so multiple Undo operations can reverse multiple editing actions.
- Redo : Reverses an Undo action. Each window (and boxed-subwindow) maintains its own history of Undo actions, so multiple Redo operations can reverse multiple Undo actions.
- Cut : Copies the selected text to the clipboard and deletes it from the window.
- Copy : Copies the selected text to the clipboard.

- Paste : Pastes the current clipboard contents into the window.
- Paste and Indent : Pastes the current clipboard contents into the window and indents it, doing the same thing the tab key would do.
- Delete : or Clear : Deletes the selected text.
- Select All : Highlights the entire text of the buffer.
- Wrap Text : Toggles between wrapped text and unwrapped text in the window.
- Find : Opens an interactive search window at the bottom of the frame and moves the insertion point to the search string editor (or out of it, if the insertion point is already there).

  See also §1.4 “Searching”.
- Find From Selection : Just like Find, except that it copies the current selection into the search window (but without using the clipboard, so paste will paste whatever was last copied, not the new search string)
- Find Next : Finds the next occurrence of the text in the search window.
- Find Previous : Finds the next occurrence of the text in the search window, but searching backwards.
- Show Replace :
- Replace : Replaces the selection with the replace string (if it matches the find string) and finds the next occurrence of the text that was last searched for, looking forwards.
- Replace All : Replaces all occurrences of the search string with the replace string.
- Find Case Sensitive : Toggles between case-sensitive and case-insensitive search.
- Find Longest Line : Moves the insertion point to the longest line in the editor.
- Keybindings :
  - Show Active Keybindings : Shows all of the keybindings available in the current window.
  - Add User-defined Keybindings... : Choosing this menu item opens a file dialog where you can select a file containing Racket-definitions of keybindings. See §3.3.9 “Defining Custom Shortcuts” for more information.
- Spell Check String Constants : Uses `aspell` or `ispell` to look for unknown words in string constants (things that would otherwise be colored green).
- Spell Check Text (between {} in Scribble : Uses `aspell` or `ispell` to look for unknown words in Scribble text.

- Skip to Next Misspelled Word : Sets the insertion point to the next word that is misspelled in the editor (if any).
- Suggest Spelling Corrections... : Opens a window showing spelling corrections for the word at the insertion point.
- Spelling Dictionaries : A submenu that lists all of the available spelling dictionaries.
- Complete Word : Completes the word at the insertion point, using the manuals as a source of completions.
- Preferences... : Opens the preferences dialog. See §3.2 “Preferences”. (On Mac OS, this menu item is in the Apple menu.)

3.1.3 View

One each of the following show/hide pairs of menu items appears at any time.

- Toolbar :
  - Toolbar on Left : Moves the tool bar (on the top of DrRacket’s window by default) to the left-hand side, organized vertically.
  - Toolbar on Top : Moves the toolbar to the top of the DrRacket window.
  - Toolbar on Right : Moves the tool bar to the right-hand side, organized vertically.
  - Toolbar Hidden : Hides the toolbar entirely.
- Split : Splits the current window in half to allow for two different portions of the current window to be visible simultaneously.
- Collapse : If the window has been split before, this menu item becomes enabled, allowing you to collapse the split window.
- Show Definitions : Shows the definitions window.
- Hide Definitions : Hides the definitions window.
- Show Interactions : Shows interactions window.
- Hide Interactions : Hides interactions window.
- Use Vertical Layout : and Use Horizontal Layout : adjust the definitions and interactions window so they are either beside each other or with the definitions above the interactions window.
- Show Log : Shows the current log messages.
- Hide Log : Hides the current log messages.

- Show Tracing : Shows a trace of functions called since the last time Run was clicked. This menu is useful only if you have enabled tracing in the Choose Language... dialog's Details section. Profiling does not apply to all languages.
- Hide Tracing : Hides the tracing display.
- Show Profile : Shows the current profiling report. This menu is useful only if you have enabled profiling in the Choose Language... dialog's Details section. Profiling does not apply to all languages.
- Hide Profile : Hides any profiling information currently displayed in the DrRacket window.
- Show Program Contour : Shows a "20,000 foot" overview window along the edge of the DrRacket window. Each pixel in this window corresponds to a letter in the program text.
- Hide Program Contour : Hides the contour window.
- Show Line Numbers : Shows line numbers in the definitions window.
- Hide Line Numbers : Hides the line numbers in the definitions window.
- Show Column Width Guide at 102 Characters : Shows the column width guide when the current file's width is greater than 102 characters.

  The number 102 is controlled in the General Editing sub-tab of the Editing tab in the preferences dialog.
- Hide Column Width Guide for 102 Characters : Hides the column width guide, even with the file's width is greater than 102 characters.

  The number 102 is controlled in the General Editing sub-tab of the Editing tab in the preferences dialog.
- Show Module Browser : Shows the module DAG rooted at the currently opened file in DrRacket.

  See also §1.9 "The Module Browser".
- Hide Module Browser : Hides the module browser.

  See also §1.9 "The Module Browser".

Note: whenever a program is run, the interactions window is made visible if it is hidden.

3.1.4 Language

- Choose Language... : Opens a dialog for selecting the current evaluation language. Click Run to make the language active in the interactions window. See §1.2 "Choosing a Language" for more information about the languages.

- Add Teachpack... : Opens a find-file dialog for choosing a teachpack to extend the current language. Click Run to make the teachpack available in the interactions windows. See §4 “Extending DrRacket” for information on creating teachpacks.
- Clear All Teachpacks : Clears all of the current teachpacks. Click Run to clear the teachpack from the interactions window.

In addition to the above items, a menu item for each teachpack that clears only the corresponding teachpack.

3.1.5 Racket

- Run : Resets the interactions window and runs the program in the definitions window.
- Ask the Program to Quit : Breaks the current evaluation.
- Force the Program to Quit : Terminates the current evaluation.
- Reload #lang Extensions : discards any already-loaded `#lang`-specific extensions and reloads them. See also §1 “Tool support for `#lang`-based Languages”.
- Limit Memory... : Allow you to specify a limit on the amount of memory that a program running in DrRacket is allowed to consume.
- Clear Error Highlight : Removes the red background that signals the source location of an error. Also removes the highlighting for uncovered (untested) portions of the program.
- Create Executable... : Creates a separate launcher for running your program. See §1.11 “Creating Executables” for more info.
- Module Browser... : Prompts for a file and then opens a window showing the module DAG starting at the module in the selected file.

  See also §1.9 “The Module Browser”.
- Module Browser on *file* : Opens a separate window showing the module graph rooted at the file currently being edited in DrRacket, but using the saved file on the disk, instead of the version in DrRacket.

  See also §1.9 “The Module Browser”.
- Reindent : Indents the selected text according to the standard Racket formatting conventions. (Pressing the Tab key has the same effect.)
- Reindent All : Indents all of the text in either the definitions or interactions window, depending on the location of the selection or blinking caret.
- Comment Out with a Box : Boxes the selected text with a comment box.

- Comment Out with ... : Uses the characters shown in the menu item to comment out the selected region of text. This menu item changes based on the `#lang` line.
- Uncomment : Removes all `;` characters at the start of each selected line of text or removes a comment box around the text. Uncommenting only removes a `;` if it appears at the start of a line and it only removes the first `;` on each line.
- Disable Tests : Stops tests written in the definitions window from evaluating when the program is Run. Tests can be enabled using the Enable Tests menu item. Disabling tests freezes the contents of any existing test report window.
- Enable Tests : Allows tests written in the definitions window to evaluate when the program is Run. Tests can be disabled using the Disable Tests menu item.

3.1.6 Insert

- Insert Comment Box : Inserts a box that is ignored by DrRacket; use it to write comments for people who read your program.
- Insert Image... : Opens a find-file dialog for selecting an image file in GIF, BMP, XBM, XPM, PNG, or JPG format. The image is treated as a value.
- Insert Fraction... : Opens a dialog for a mixed-notation fraction, and inserts the given fraction into the current editor.
- Insert Large Letters... : Opens a dialog for a line of text, and inserts a large version of the text (using semicolons and spaces).

  Most of the dialog is self-explanatory: type in the top space to preview the semicolons in the bottom area. The numbers in the font choice item show the (relative) widths of the letter "w" in the given font to help you pick out the more boldfaced fonts (which tend to look better).
- Insert λ : Inserts the symbol λ (as a Unicode character) into the program. The λ symbol is normally bound the same as `lambda`.
- Insert XML Box : Inserts an XML; see §1.7.2 "XML Boxes and Racket Boxes" for more information.
- Insert Racket Box : Inserts a box to contain Racket code, typically used inside an XML box; see §1.7.2 "XML Boxes and Racket Boxes".
- Insert Racket Splice Box : Inserts a box to contain Racket code, typically used inside an XML box; see also §1.7.2 "XML Boxes and Racket Boxes".

3.1.7 Windows

- Bring Frame to Front... : Opens a window that lists all of the opened DrRacket frames. Selecting one of them brings the window to the front.
- Most Recent Window : Toggles between the currently focused window and the one that most recently had the focus.

Additionally, after the above menu items, this menu contains an entry for each window in DrRacket. Selecting a menu item brings the corresponding window to the front.

3.1.8 Help

- Documentation : Opens the main documentation page. This is the clearing house for all documentation about the language family of the programming language in the definitions window.
- About DrRacket... : Shows the credits for DrRacket.
- Check for Updates... : Checks to see if a new version of DrRacket has been released.
- Related Web Sites : Provides links to related web sites.
- Tool Web Sites : Provides links to web sites for installed tools.
- Submit Bug Report... : The preferred mechanism for sending in bug reports to the Racket development team. It automatically collects information about your Racket installation that sometimes can help diagnose problems.
- Saved Bug Reports : A submenu that holds bug reports that you've started editing but haven't yet sent in.
- Saved Bug Reports : A submenu that holds bug reports that you've started editing but haven't yet sent in.
- Configure Command Line for Racket... : This menu item is available under Mac OS and Windows.

  On MacOS, after prompting you for your password, it adds a file named `"racket"` to `"/etc/init.d/"` that contains the location of the `"bin"` directory. This has the effect of adding that path to the default PATH environment variable (unless your shell explicitly changes the default behavior).

  On Windows, it changes the `HKEY_CURRENT_USER\Environment\Path` registry key to add the location of the `"bin"` directory. Once it finishes, newly created Command shells (`cmd.exe`) should have Racket in their path.
- Interact with DrRacket in English : Changes DrRacket's interface to use English; the menu item appears only when the current language is not English. Additional menu items switch DrRacket to other languages.

3.1.9 Popup menu

The following additional menu items may be available when activating the popup (context) menu in either the definitions or interactions window.

- Tack/Untack Arrow(s) : Tack binding relationship arrows so they do not disappear. Repeat to untack arrows.
- Open Defining File : Opens the file with the currently selected identifier was defined.
- Jump to Next Bound Occurrence : Selects the next occurence of the variable.
- Jump to Binding Occurrence : Selects where the identifier was bound in the text. If imported the name of the imported library is selected.
- Add Require Prefix : Adds prefix-in, prompts for prefix, and renames instances of imported identifier to include the prefix.
- Remove Unused Requires : Refactoring tool to remove unused requires in module.
- Rename : Renames all occurrences of selected variable.
- Collapse S-expression : Caused the current S-expression to be rendered as $(\cdots)$
- Expand S-expression : Expands a collapsed S-expression (displayed as centered ellipses) back to original text.
- Search in Help Desk for ... : Opens browser and triggers search of documentation for selected term.
- Save Image... : Save the clicked-on image to file.

Standard editing functions are also available via the context menu: Undo, Redo, copy, cut, paste, Select All, and Select All.

3.2 Preferences

The preferences dialog consists of several panels.

3.2.1 Font

This panel controls the main font used by DrRacket.

3.2.2 Colors

The Colors panel has several sub-panels that let you configure the colors that DrRacket uses.

- Color Schemes

  DrRacket comes with a number of different color schemes that you can choose from. The Classic and White on Black schemes are actually a matched pair that, under Mac OS and Linux will be chosen based on the operating system's dark-mode configuration. Similarly for the two Tol's color-blind safe modes.

- Background

  This section of the preferences configures the default background color and the colors used behind matching parentheses. The Parenthesis color scheme menu also includes options that shade nested parentheses darker and lighter, based on the nesting depth.

The remaining panels inside the Colors panel allow the configuration of many different colors that DrRacket uses for specific parts of its interface.

3.2.3 Editing

The Editing panel consists of several sub-panels:

- Indenting

  This panel controls which keywords DrRacket recognizes for indenting, and how each keyword is treated.

  To decide how a particular line is indented, DrRacket starts at the beginning of the line and looks for an enclosing s-expression. It finds the first symbol after that open parenthesis, and uses that to determine how to indent the line.

  If the symbol is not listed in any of the lists in the preferences dialog, then it is indented like a function call, e.g.:

```
(f a
   b
   c)
```

```
(f
 a
 b
 c)
```

  If the symbol is listed in the Begin-like Keywords section, it gets one extra space when the first sub-expression is on a different line and otherwise it looks like an application.

```
(begin a
       b
       c)

(begin
  a
  b
  c)
```

If the symbol is listed in the Define-like Keywords section, then all of the subexpressions are indented one extra space, no matter what happens on the first line:

```
(define x
  1)

(struct s t
  (a b c))

(define
  (f x)
  1)
```

If the symbol is listed in the Lambda-like Keywords section, then it is indented like define, except that there is three extra spaces of indentation added when the first subexpression is not on the same line as the keyword:

```
(λ (x y)
  1)

(lambda
    (x y)
  1)
```

If the symbol is listed in the For/fold-like Keywords section, it is indented just like the define section, except that the first two sub-expressions are treated specially:

```
(for/fold ([acc 0])
          ([x (in-range 10)]
           [y (in-list l)])
  b
  c)
```

If the symbol is not listed in any of the list boxes in the various sections, the contents of the Extra Regexp panel are consulted. They are regular expressions that are matched against the printed representation of the symbol, and if they match, the corresponding indentation style is used.

- Square bracket

  This panel controls which keywords DrRacket uses to determine when to rewrite `[` to `(`. For `cond`-like keywords, the number in parenthesis indicates how many subexpressions are skipped before square brackets are started.

See §1.3 “Editing with Parentheses” for details on how the entries in the columns behave.

- General Editing
  - Map delete to backspace — If checked, the editor treats the Delete key like the Backspace key.
  - Color syntax interactively — If checked, DrRacket colors your syntax as you type.
  - Wrap words in editor buffers — If checked, DrRacket editors auto-wrap text lines by default. Changing this preference affects new windows only.
  - Search using anchors — If checked, DrRacket's searching mode will jump directly to the first search hit, using an “anchor” to determine where to search if the search string changes.
  - Disable caret blinking — If checked, the insertion point in the definitions and interactions windows (and other editors) does not blink.
  - Normalize pasted strings — If checked, DrRacket adjusts strings that are pasted into the editor to avoid confusion. For example, non-breaking spaces look just like spaces but are not considered separators like ordinary spaces are. If this is checked DrRacket will automatically turn those non-breaking spaces into regular spaces. Similarly with other (less common) characters.
  - Treat alt key as meta — If checked, DrRacket will use the alt (option) key for some Emacs-like keybindings, instead of using it for menu shortcuts. This option is only available on Mac OS.
  - Enable overwrite mode keybindings — If checked, DrRacket enables the insert keybinding to swap into overwrite mode
  - Treat command key as meta — If checked, DrRacket will use the command key for some Emacs-like keybindings, instead of using it for menu shortcuts. This option is only available on Mac OS.
  - Always use the platform-specific linefeed convention — If checked, DrRacket always saves files with CRLF line terminators. If unchecked, DrRacket looks at each file as it is opened and if every line is terminated with CRLF (and there is at least one line), then it saves the file with CRLF terminators and otherwise it is saved with LF terminators (following the Mac OS and Linux convention). When a file is going to be saved with CRLF terminators, then the status line at the bottom of the DrRacket window shows “CRLF”.

    This option is available only under Windows. On other operating systems, all files are always saved with LF line terminators.
  - Enable keybindings in menus — If checked, some DrRacket menu items have keybindings. Otherwise, no menu items have key bindings. This preference is designed for people who are comfortable editing in Emacs and find the standard menu keybindings interfere with the Emacs keybindings.

- Enable automatic parentheses, square brackets, and quotes — If checked, typing an open parenthesis, curly brace, square bracket, double quote, or vertical bar character automatically inserts a matching one. Additionally, backspace will automatically remove matched empty pairs of such characters when the caret is between them in many cases.
- Add one pixel of extra space between lines — If checked, then an extra pixel of whitespace is added between lines in the editor. The default value is platform-specific; some fonts (notably those with ╔═╗ characters) only look right with this unchecked.
- Maximum character width guide — If checked, DrRacket draws a vertical line in files that exceed the given maximum width. The vertical line shows where the given maximum width is.
- Show line numbers — If checked, DrRacket shows line numbers for the file being edited in the left-hand column
- Show definitions/interactions labels — If checked, then the teaching languages show big friendly labels indicating which window is the definitions window and which is the interactions window.]
- Startup Open Files — If the Restore files from previous session is selected, DrRacket will reopen the files that were open when DrRacket last exited, as it starts up. When Open a blank window is selected, DrRacket starts with a new, fresh window each time. When Ask me each time is selected and some files were open in the previous session, DrRacket will open a dialog box asking if you would like to open the same files as last time. If DrRacket is started from the command-line (or via Mac OS's open command) and files are supplied, then this preference is ignored, and only those files are opened.
- Automatic `#lang` line — When new files are opened in DrRacket, it will use this for the first line of the buffer.

• Racket

- Highlight between matching parens — If checked, the editor marks the region between matching parenthesis with a gray background (in color) or a stipple pattern (in monochrome) when the blinking caret is next to a parenthesis.
- Automatically adjust closing parens — If checked, the editor automatically converts a typed `)` to `]` to match `[`, or it converts a typed `]` to `)` to match `(`.
- Automatically adjust opening square brackets If checked, the editor changes typed `[` to match the context (as explained in §1.3 "Editing with Parentheses").
- Flash paren match — If checked, typing a closing parenthesis, square bracket, or quotation mark flashes the matching open parenthesis/bracket/quote.

3.2.4 Warnings

• Ask before changing save format — If checked, DrRacket consults the user before saving a file in non-text format (see §3.5.1 "Program Files").

- Verify exit — If checked, DrRacket consults the user before exiting.
- Ask about normalizing strings — If checked, DrRacket consults the user before normalizing a string pasted into the editor.
- Only warn once when executions and interactions are not synchronized — If checked, DrRacket warns the user on the first interaction after the definitions window, language, or teachpack is changed without a corresponding click on Run. Otherwise, the warning appears on every interaction.
- Ask about clearing test coverage — If checked, when test coverage annotations are displayed DrRacket prompts about removing them. This setting only applies to the PLT languages. DrRacket never asks in the teaching languages.
- Check for newer Racket versions — If checked, DrRacket periodically polls a server to determine whether a newer version of DrRacket is available.

3.2.5 General

- Number of recent items — controls the length of the Open Recent menu (in the File menu).
- Make backups for unsaved files — If checked, the editor generates copies of open files (see §3.5.2 "Backup and First Change Files") for files that have not been saved after thirty seconds.
- Create first-change files — If checked, when saving a file for the first time in each editing session, the original copy of the file is copied to a new file in the same directory. The new files has the same name as the original, except that it ends in either `".bak"` or `"~"`.
- Show status-line — If checked, DrRacket shows a status line at the bottom of each window.
- Count column numbers from one — If checked, the status line's column counter counts from one. Otherwise, it counts from zero.
- Display line numbers in buffer; not character offsets — If checked, the status line shows a ⟨*line*⟩:⟨*column*⟩ display for the current selection rather than the character offset into the text.
- Automatically print to PostScript file — If checked, printing will automatically save PostScript files. If not, printing will use the standard printing mechanisms for your computer.
- Open files in separate tabs (not separate windows) — If checked, DrRacket will use tabs in the front-most window to open new files, rather than creating new windows for new files.

- Automatically open interactions window when running a program — If checked, DrRacket shows the interactions window (if it is hidden) when a program is run.
- Automatically switch to the module language when opening a module — If checked, DrRacket will recognize files that have a `#lang` line and adjust the language setting automatically.
- Put the interactions window beside the definitions window — If checked, DrRacket puts the interactions window to the right of the definitions window. By default, the interactions window is below the definitions window.
- Always show the #lang line in the Module language — If checked, the module language always shows the `#lang` line (even when it would ordinarily be scrolled off of the page), assuming that the `#lang` line is the first line in the file.
- Show test coverage results summary — If checked, DrRacket shows a little note indicating that all tests were covered in addition to not changing the colors. If not checked, DrRacket remains silent (unchanged) when test coverage is enabled but all of the code is covered.
- Save files whenever switching tabs or windows — If checked, DrRacket will automatically save any unsaved files when switching between windows or tabs.
- Don't ask and don't save files when clicking Run with unsaved tabs or windows — If checked, DrRacket will not show a warning dialog about unsaved tabs when clicking the Run button, nor will it save the files automatically. If this is checked, the previous checkbox's value is ignored.
- Show stacktrace for exn:fail:syntax exceptions? — If checked, DrRacket will include the multi-stop sign icon that, when clicked, shows the stacktraces at the point where syntax errors are raised. This is off by default as, for most languages, these correspond to internal implementation information, not helpful debugging information. When debugging macros, however turning this on can be useful.

3.2.6 Profiling

This preference panel configures the profiling report. The band of color shows the range of colors that profiled functions take on. Colors near the right are used for code that is not invoked often and colors on the right are used for code that is invoked often.

If you are interested in more detail at the low end, choose the Square root check box. If you are interested in more detail at the upper end, choose the Square check box.

3.2.7 Browser

This preferences panel allows you to configure your HTTP proxy. Contact your system administrator for details.

3.2.8 Tools

This preference panel allows you to configure the currently active plugins.

3.3 Keyboard Shortcuts

Most key presses simply insert a character into the editor, such as `a`, `3`, or `(`. Other keys and key combinations act as keyboard shortcuts that move the blinking caret, delete a line, copy the selection, etc. Keyboard shortcuts are usually trigger by key combinations using the Control, Meta, or Command key.

The keyboard shortcuts are displayed using the same string-encoded notation for modifier keys as `map-function` in `keymap%` and the Keybindings view;

- `s:` — All platforms: Shift
- `c:` — All platforms: Control
- `a:` — Mac OS: Option
- `m:` — Windows: Alt; Unix: Meta; Mac OS: Command, when `map-command-as-meta-key` produces `#t`
- `d:` — Mac OS: Command
- `l:` — All platforms: Caps Lock
- `g:` — Windows: Control plus Alt as AltGr; see `get-control+meta-is-altgr` in `key-event%`
- `?:` — All platforms: allow match to character produced by opposite use of Shift, AltGr/Option, and/or Caps Lock, when available; see `get-other-shift-key-code` in `key-event%`

If a modifier key is not specified, it matches whether that modifier is pressed or not pressed. A `~` prior to a modifier indicates the keybinding only activates if that modifier is not pressed.

A keyboard shortcut that begins with `:` only activates if the modifiers Shift, Control, Option, Alt, Meta, or Command are not pressed.

Many of the key-binding actions can also be performed with menu items.

c:⟨*key*⟩ means press the Control key, hold it down and then press ⟨*key*⟩ and then release them both. For example: c:e (Control-E) moves the blinking caret to the end of the current line.

m:⟨*key*⟩ is the same as c:⟨*key*⟩, except with the Meta key. Depending on your keyboard, Meta may be called "Left," "Right," or have a diamond symbol, but it's usually on the bottom row next to the space bar. m:⟨*key*⟩ can also be performed as a two-character sequence: first, strike and release the Escape key, then strike ⟨*key*⟩. On Mac OS, Meta is, by default, available only through the Escape key. But the preferences dialog (in the General Editing sub-panel of the Editing panel) has check boxes that adjust the handling of the Alt key or the Command key to be meta.

`delete` is the Delete key.

`space` is the Space bar.

On most keyboards, "<" and ">" are shifted characters. So, to get m:>, you actually have to type Meta-Shift->. That is, press and hold down both the Meta and Shift keys, and then strike ">".

On Windows (and sometimes under Unix) some of these keybindings are actually standard menu items. Those keybindings will behave according to the menus, unless the Enable keybindings in menus preference is unchecked. For example, the c:e keybinding mentioned above actually toggles the visibility of the interactions window.

If you are most familiar with Emacs-style key bindings (especially on windows or some linux installations where the control key is, by default, for the menu shortcuts), you should uncheck the Enable keybindings in menus preference. Many of the keybindings below are inspired by Emacs. See also §3.3.9 "Defining Custom Shortcuts" for suggestions on how to bind keys to menu items on a selective basis.

Some keybindings are not available if Enable keybindings in menus is enabled.

And finally, the authoritative source for keybindings is the Edit menu's Show Active Keybindings menu item. Keybindings in DrRacket are often sensitive to the window that has the keyboard focus, so the contents of the window that Show Active Keybindings opens will depend where the keyboard focus was when the menu was selected.

3.3.1 Moving Around

- c:f : move forward one character
- c:b : move backward one character
- m:f : move forward one word
- m:b : move backward one word
- c:v : move forward one page

- m:v : move backward one page
- m:< : move to beginning of file
- m:> : move to end of file
- c:a : move to beginning of line (left)
- c:e : move to end of line (right)
- c:n : move to next line (down)
- c:p : move to previous line (up)
- m:c:f : move forward one S-expression
- m:c:b : move backward one S-expression
- m:c:u : move up out of an S-expression
- m:c:d : move down into a nested S-expression
- m:c:space : select forward S-expression
- m:c:p : match parentheses backward
- m:c:left : move backwards to the nearest editor box
- A-c:left : move backwards to the nearest editor box
- m:c:right : move forward to the nearest editor box
- A-c:right : move forward to the nearest editor box
- m:c:up : move up out of an embedded editor
- A-c:up : move up out of an embedded editor
- m:c:down : move down into an embedded editor
- A-c:down : move down into an embedded editor
- c:c;c:z : move the cursor to the interactions window
- c:F6 : move the cursor between different windows (usually the interactions and definitions windows, but also the search window and other editable portions of DrRacket). Also, search for "shift-focus" in the Show Active Keybindings menu's window for more, platform-specific keybindings that have this functionality

3.3.2 Editing Operations

- c:_ : undo
- c:+ : redo
- c:x;u : undo
- c:d : delete forward one character
- c:h : delete backward one character
- m:d : delete forward one word
- m:delete : delete backward one word
- c:k : delete forward to end of line
- m:c:k : delete forward one S-expression
- m:w : copy selection to clipboard
- c:w : delete selection to clipboard (cut)
- c:y : paste from clipboard (yank)
- c:t : transpose characters
- m:t : transpose words
- m:c:t : transpose sexpressions
- m:c:m : toggle dark green marking of matching parenthesis
- m:c:k : cut complete sexpression
- m:(: wrap selection in parentheses
- m:[: wrap selection in square brackets
- m:{ : wrap selection in curly brackets
- m:s:l : wrap selection in `(lambda () ...)` and put the insertion point in the argument list of the lambda
- c:c;c:o : the sexpression following the insertion point is put in place of its containing sexpression
- c:c;c:e : the first and last characters (usually parentheses) of the containing expression are removed
- c:cc;:l : wraps a let around the sexpression following the insertion point and puts a printf in at that point (useful for debugging).

- m:o : toggle overwrite mode
- c:x;r;a : Adjust nearby ASCII art rectangles (that use `+`, `-`, or `|`) to use Unicode characters.

  For example, if the insertion point is next to this rectangle:

  ```
  +-+
  | |
  +-+
  ```

  then the keystroke will turn it into this one:

  ```
  ╔═╗
  ╟ ╢
  ╚═╝
  ```

  Similarly, if the rectangle near the insertion point has mixed Unicode and ASCII, it will all be converted to the Unicode characters.
- c:x;r;w : Widen the nearby ASCII art rectangles.

  For example, if the insertion point is just to the left of the middle line of this rectangle:

  ```
  ╔═══╗
  ╟ ╟ ╢
  ╠═══╣
  ╟ ╟ ╢
  ╚═══╝
  ```

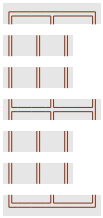
  then the keystroke will turn it into this one:

  ```
  ╔═══╗
  ╟ ╟ ╢
  ╠═══╣
  ╟ ╟ ╢
  ╚═══╝
  ```
- c:x;r;v : Make the nearby ASCII art rectangles taller.

  For example, if the insertion point is just above the the middle line of this rectangle:

  ```
  ╔═══╗
  ╟ ╟ ╢
  ╠═══╣
  ╟ ╟ ╢
  ╟ ╟ ╢
  ╚═══╝
  ```

  then the keystroke will turn it into this one:



- c:x;r;c : Centers the contents of the current line inside the enclosing cell of the enclosing ASCII art rectangle.
- c:x;r;o : Toggles the ASCII art rectangle editing mode. When the mode is enabled, key strokes that would normally break the rectangles instead enlarge them. Specifically:
  - Return and enter add a line to the enclosing rectangle and put the insertion point at the first column of the enclosing cell.
  - When in overwrite mode, if a key would overwrite one of the walls of the cell, the wall is first moved over to accomodate the new key
  - When not in overwrite mode, inserting a character will always widen the containing cell

3.3.3 File Operations

- c:x;c:s : save file
- c:x;c:w : save file under new name

3.3.4 Search

- c:s : search for string forward
- c:r : search for string backward

3.3.5 Evaluation

- f5 : Run

3.3.6 Documentation

- f1 : Search in the documentation for the words near the insertion point

- f2 : Reveal the signature box for the identifier at the insertion point (requires background check syntax to be enabled or normal check syntax to have been run).

3.3.7 Interactions

The interactions window has all of the same keyboard shortcuts as the definitions window plus a few more:

- m:p : bring the previously entered expression down to the prompt
- m:n : bring the expression after the current expression in the expression history down to the prompt
- m:h : Show the current expression history in a separate window

Also, in the interactions window, the return key is treated specially, in particular to determine if the expression that's been typed at the prompt is ready to be evaluated or not. There are three cases:

- When shift-return is typed: the expression is not submitted and instead a new line is started
- When alt-return or control return is typed: the expression is submitted for evaluation
- When return with no modified is typed: the language is consulted to determine which of the two previous actions makes the most sense. Typically the language's parser is consulted to see if the expression looks complete. If so, the expression is submitted and, if not, a new line is started. (See also `'drracket:submit-predicate`.)

3.3.8 LaTeX and TeX inspired keybindings

- c:\ m:\ c:x;l : traces backwards from the insertion point, looking for a backslash followed by a LaTeX macro name or a prefix of such a name. If a macro name is found, it replaces the backslash and the name with the corresponding key in the table below; if a (proper) prefix *p* is found, it replaces *p* with the longest common prefix of all macro names that have *p* as a prefix (unless there is only one such name, in which case it behaves as if *p* were a complete macro name).

  These are the currently supported macro names and the keys they map into:

`\Downarrow`	⇓	`\nwarrow`	↖
`\downarrow`	↓	`\Rightarrow`	⇒
`\rightarrow`	→	`\rightharpoonup`	⇀
`\leftharpoonup`	↼	`\rightharpoondown`	⇁

\leftharpoondown	$\leftharpoondown$	\rightleftharpoons	$\rightleftharpoons$
\mapsto	$\mapsto$	\searrow	$\searrow$
\swarrow	$\swarrow$	\leftarrow	$\leftarrow$
\uparrow	$\uparrow$	\Leftarrow	$\Leftarrow$
\longrightarrow	$\longrightarrow$	\Uparrow	$\Uparrow$
\Leftrightarrow	$\Leftrightarrow$	\updownarrow	$\updownarrow$
\leftrightarrow	$\leftrightarrow$	\nearrow	$\nearrow$
\Updownarrow	$\Updownarrow$	\leadsto	$\leadsto$
\aleph	$\aleph$	\prime	$\prime$
\emptyset	$\emptyset$	\nabla	∇
\diamondsuit	$\diamondsuit$	\spadesuit	$\spadesuit$
\clubsuit	$\clubsuit$	\heartsuit	$\heartsuit$
\sharp	$\sharp$	\flat	$\flat$
\natural	$\natural$	\surd	$\surd$
\neg	$\neg$	\triangle	$\triangle$
\forall	$\forall$	\all	$\forall$
\exists	$\exists$	\infty	∞
\circ	$\circ$	\partial	∂
\alpha	α	\theta	θ
\tau	τ	\beta	β
\vartheta	ϑ	\pi	π
\upsilon	υ	\gamma	γ
\varpi	ϖ	\phi	ϕ
\delta	δ	\kappa	κ
\rho	ρ	\varphi	φ
\epsilon	ϵ	\lambda	λ
\varrho	ϱ	\chi	χ
\varepsilon	ε	\mu	μ
\sigma	σ	\psi	ψ
\zeta	ζ	\nu	ν
\varsigma	ς	\omega	ω
\eta	η	\xi	ξ
\iota	ι	\Gamma	Γ
\Lambda	Λ	\Sigma	Σ
\Psi	Ψ	\Delta	Δ
\Xi	Ξ	\Upsilon	Υ
\Omega	Ω	\Theta	Θ
\Pi	Π	\Phi	Φ
\pm	$\pm$	\cap	$\cap$
\diamond	$\diamond$	\lozenge	$\lozenge$
\square	$\square$	\oplus	$\oplus$
\mp	$\mp$	\cup	$\cup$
\bigtriangleup	$\bigtriangleup$	\ominus	$\ominus$
\times	$\times$	\uplus	$\uplus$
\bigtriangledown	$\bigtriangledown$	\otimes	$\otimes$
\div	$\div$	\sqcap	$\sqcap$

\triangleright	$\triangleright$	\oslash	$\oslash$
\ast	$\ast$	\sqcup	$\sqcup$
\vee	$\vee$	\wedge	$\wedge$
\lor	$\vee$	\land	$\wedge$
\lnot	$\neg$	\triangleleft	$\triangleleft$
\angle	$\angle$	\odot	$\odot$
\star	$\star$	\dagger	$\dagger$
\bullet	$\bullet$	\ddagger	$\ddagger$
\wr	$\wr$	\amalg	$\amalg$
\leq	$\leqslant$	\geq	$\geqslant$
\equiv	$\equiv$	\models	$\models$
\prec	$\prec$	\succ	$\succ$
\precdot	$\preccurlyeq$	\succdot	$\succcurlyeq$
\sim	$\sim$	\perp	$\perp$
\bot	$\bot$	\top	$\top$
\preceq	$\preceq$	\succeq	$\succeq$
\simeq	$\simeq$	\ll	$\ll$
\gg	$\gg$	\guillemetleft	$\guillemotleft$
\guillemetright	$\guillemotright$	\asymp	$\asymp$
\parallel	$\parallel$	\subset	$\subset$
\supset	$\supset$	\approx	$\approx$
\bowtie	$\bowtie$	\subseteq	$\subseteq$
\supseteq	$\supseteq$	\nsubseteq	$\nsubseteq$
\subsetneq	$\subsetneq$	\cong	$\cong$
\sqsubsetb	$\sqsubset$	\sqsupsetb	$\sqsupset$
\neq	$\neq$	\smile	$\smile$
\sqsubseteq	$\sqsubseteq$	\sqsupseteq	$\sqsupseteq$
\doteq	$\doteq$	\frown	$\frown$
\in	$\in$	\ni	$\ni$
\notin	$\notin$	\propto	$\propto$
\vdash	$\vdash$	\dashv	$\dashv$
\cdot	$\cdot$	\sum	$\sum$
\prod	$\prod$	\coprod	$\coprod$
\int	$\int$	\iint	$\iint$
\iiint	$\iiint$	\oint	$\oint$
\oiint	$\oiint$	\oiiint	$\oiiint$
\sqrt	$\surd$	\skull	☠
\smiley	☺	\blacksmiley	☻
\frownie	☹	\Re	$\Re$
\Im	$\Im$	\S	§
\l	ł	\newpage	^L
\vdots	$\vdots$	\ddots	$\ddots$
\cdots	$\cdots$	\hdots	$\cdots$
\ldots	$\ldots$	\langle	$\langle$
\rangle	$\rangle$	\amp	&

\invamp	⅋	\multimap	⊸
\rightlollipop	⊸	\leftmultimap	⟜
\multimapinv	⟜	\leftlollipop	⟜

Additionally, all of the digits, lowercase letters, and uppercase letters can be turned into blackboard bold characters with the shortcut \b followed by the character. For example, \bN turns into ℕ and \bZ turns into ℤ.

3.3.9 Defining Custom Shortcuts

The Add User-defined Keybindings... menu item in the Keybindings sub-menu of Edit selects a file containing Racket definitions of keybindings. The file must contain a module that uses a special keybindings language, framework/keybinding-lang. To do so, begin your file with this line:

```
#lang s-exp framework/keybinding-lang
```

The framework/keybinding-lang languages provides all of the bindings from racket, racket/class, and drracket/tool-lib, except that it adjusts #%module-begin to introduce a keybinding form:

(keybinding *string-expr proc-expr*)

Declares a keybinding, where *string-expr* must produce a suitable first argument for map-function in keymap%, and the *proc-expr* must produce a suitable second argument for add-function in keymap%.

For example, this remaps the key combination “control-a” key to “!”.

```
#lang s-exp framework/keybinding-lang
(keybinding "c:a" (λ (editor evt) (send editor insert "!")))
```

Since the file contains plain Racket code, you can write keybindings files that use DrRacket’s Extension API. For example, the following file binds “control-t” and “control-=” to a execute the program and open a new tab respectively, as they were used before version 5.2.

```
#lang s-exp framework/keybinding-lang

(define modifiers
  (apply string-append
         (map (λ (p)
                (case p
                  [(ctl) "c:"] [(cmd) "d:"] [(alt meta) "~c:m:"]
```

```
                    [(shift) "s:"] [(option) "a:"]))
                (get-default-shortcut-prefix))))

(define-syntax-rule (frame-key key command)
  (keybinding
   (string-append modifiers key)
   (λ (ed evt)
     (when (is-a? ed text:basic<%>)
       (define fr (send ed get-top-level-window))
       ; note: fr could be #f
       (when fr (send fr command))))))

(frame-key "t" execute-callback)
(frame-key "=" create-new-tab)
```

Another example, this file rebinds "control-w" to delete the word behind the insertion point, but it does it by setting a new key to be an existing keyboard shortcut. If you see a key in the Show Active Keybindings dialog (in the Keybindings submenu of the Edit menu), then you can use its name with the new keystroke you want, like this:

```
#lang s-exp framework/keybinding-lang

(define (rebind key command)
  (keybinding
   key
   (λ (ed evt)
     (send (send ed get-keymap) call-function
           command ed evt #t))))

(rebind "c:w" "backward-kill-word")
```

This example shows how to bind a menu item (based on its name) to a particular key. The call at the end of the example binds "control-a" to the Run menu item.

```
#lang s-exp framework/keybinding-lang

(define (menu-bind key menu-item)
  (keybinding
   key
   (λ (ed evt)
     (define canvas (send ed get-canvas))
     (when canvas
       (define menu-bar (find-menu-bar canvas))
       (when menu-bar
         (define item (find-item menu-bar menu-item))
```

```
          (when item
            (define menu-evt
              (new control-event%
                   [event-type 'menu]
                   [time-stamp
                    (send evt get-time-stamp)]))
            (send item command menu-evt)))))))

(define/contract (find-menu-bar c)
  (-> (is-a?/c area<%>) (or/c #f (is-a?/c menu-bar%)))
  (let loop ([c c])
    (cond
      [(is-a? c frame%) (send c get-menu-bar)]
      [(is-a? c area<%>) (loop (send c get-parent))]
      [else #f])))

(define/contract (find-item menu-bar label)
  (-> (is-a?/c menu-bar%)
      string?
      (or/c (is-a?/c selectable-menu-item<%>) #f))
  (let loop ([o menu-bar])
    (cond
      [(is-a? o selectable-menu-item<%>)
       (and (equal? (send o get-plain-label) label)
            o)]
      [(is-a? o menu-item-container<%>)
       (for/or ([i (in-list (send o get-items))])
         (loop i))]
      [else #f])))

(menu-bind "c:a" "Run")
```

Note that DrRacket does not reload keybindings files automatically when you make changes, so you'll need to restart DrRacket to see changes to the file.

3.3.10 Sending Program Fragments to the REPL

Users comfortable with Emacs and the conventional Lisp/Scheme-style of interaction with an "inferior process" commonly request keybindings in DrRacket that send program fragments to be evaluated at the prompt. This style of interaction is fraught with difficulty, especially for beginners, and so DrRacket, by default, does not support it. Instead, clicking DrRacket's "Run" button starts with a clean slate and sends the entire contents of the definitions window, ensuring that the state in the REPL matches what you would expect by reading the source code of the program.

Based on years of experience with Emacs modes, some of the authors consider this mode of interaction also appropriate for experienced programmers. Indeed, they go through great effort to mimic this behavior in Emacs.

That said, some people may wish to use such incremental keystroke modes anyway. Therefore the remainder of this section illustrates how to add such an incremental mode for your personal use with an example keybindings file. Specifically, the file shows how to add the ability to send expressions piecemeal to the interactions window. It also demonstrates how to pull together a bunch of pieces of DrRacket's implementation and its libraries to implement keystrokes.

```
#lang s-exp framework/keybinding-lang

(require drracket/tool-lib)

(module test racket/base)

(keybinding "c:c;c:e" (lambda (ed evt) (send-toplevel-
form ed #f)))
(keybinding "c:c;c:r" (lambda (ed evt) (send-selection ed #f)))
(keybinding "c:c;~c:m:e" (lambda (ed evt) (send-toplevel-
form ed #t)))
(keybinding "c:c;~c:m:r" (lambda (ed evt) (send-selection ed #t)))

(define/contract (send-toplevel-form defs shift-focus?)
  (-> any/c boolean? any)
  (when (is-a? defs drracket:unit:definitions-text<%>)
    (define sp (send defs get-start-position))
    (when (= sp (send defs get-end-position))
      (cond
        [(send defs find-up-sexp sp)
         ;; we are inside some top-level expression;
         ;; find the enclosing expression
         (let loop ([pos sp])
           (define next-up (send defs find-up-sexp pos))
           (cond
             [next-up (loop next-up)]
             [else
              (send-range-to-repl defs
                                  pos
                                  (send defs get-forward-sexp pos)
                                  shift-focus?)]))]
        [else
         ;; we are at the top-level
         (define fw (send defs get-forward-sexp sp))
         (define bw (send defs get-backward-sexp sp))
```

```
        (cond
          [(and (not fw) (not bw))
           ;; no expressions in the file, give up
           (void)]
          [(not fw)
           ;; no expression after the insertion point;
           ;; send the one before it
           (send-range-to-repl defs
                               bw
                               (send defs get-forward-sexp bw)
                               shift-focus?)]
          [else
           ;; send the expression after the insertion point
           (send-range-to-repl defs
                               (send defs get-backward-sexp fw)
                               fw
                               shift-focus?)])])))

(define/contract (send-selection defs shift-focus?)
  (-> any/c boolean? any)
  (when (is-a? defs drracket:unit:definitions-text<%>)
    (send-range-to-repl defs
                        (send defs get-start-position)
                        (send defs get-end-position)
                        shift-focus?)))

(define/contract (send-range-to-repl defs start end shift-focus?)
  (->i ([defs (is-a?/c drracket:unit:definitions-text<%>)]
        [start exact-positive-integer?]
        [end (start) (and/c exact-positive-integer? (>=/c start))]
        [shift-focus? boolean?])
       any)
  (unless (= start end) ;; don't send empty regions
    (define ints (send (send defs get-tab) get-ints))
    (define frame (send (send defs get-tab) get-frame))
    ;; copy the expression over to the interactions window
    (send defs move/copy-to-edit
          ints start end
          (send ints last-position)
          #:try-to-move? #f)

    ;; erase any trailing whitespace
    (let loop ()
      (define last-pos (- (send ints last-position) 1))
      (when (last-pos . > . 0)
        (define last-char (send ints get-character last-pos))
```

```
    (when (char-whitespace? last-char)
      (send ints delete last-pos (+ last-pos 1))
      (loop))))

;; put back a single newline
(send ints insert
      "\n"
      (send ints last-position)
      (send ints last-position))

;; make sure the interactions is visible
;; and run the submitted expression
(send frame ensure-rep-shown ints)
(when shift-focus? (send (send ints get-canvas) focus))
(send ints do-submission)))
```

Others may wish to use the above example to invent other keystrokes for making work in DrRacket convenient.

3.4 Status Information

The DrRacket window contains a number of different pieces of status information that tries to expose what DrRacket is doing in a relatively unobtrusive way.

In the bottom right corner of the DrRacket window there are a number of pieces of information. From left to right:

- The name of the language that DrRacket is currently using. This is also a popup menu that you can use to change the language from a list of recently used languages.
- Further over to the right are some numbers showing the line and column or the character offset of the insertion point (where characters will appear when you next type).
- The current heap size that DrRacket is using internally. The number is updated infrequently; to see the accurate number, click on it, which also triggers a major garbage collection.
- An small square that is blank unless DrRacket is doing a garbage collection and a recycle icon when DrRacket is doing garbage collection.
- A narrow blank space or a pair of parentheses. If DrRacket is working to determine the colors and where the matching parentheses are in the definitions, that will be a pair of parentheses and if not, it will be blank.

- A small person that indicates if DrRacket is running the program in the interactions window or not. (The person will run when DrRacket is running your program.)
- A circle that indicates the status of DrRacket's online expansion (only available when using "Determine language from source"). If it is red, there is an error in the program or the online expansion has been disabled. If it is blue, DrRacket is working to expand your program. If it is (dark or light) purple, DrRacket is processing the result of the expansion to compute where the arrows go and which how renaming any given variable affects any of the others. If it is green, DrRacket's online expansion has finished. If it is a spiky green ball (instead of a round green ball), and the language settings do not imply errortrace annotations, then DrRacket will use the result of the online expansion to speed up the Run button. That is, when the green spiky ball is present, DrRacket will have already expanded and compiled your program, and can reuse that work, as long as errortrace debugging, profiling, and test suite coverage are turned off.

3.5 DrRacket Files

3.5.1 Program Files

The standard file extension for a Racket program file is `".rkt"`. The extensions `".ss"`, `".scm"`, and `".sch"` are also historically popular.

DrRacket's editor can save a program file in two different formats:

- *Plain-text file format* — All text editors can read this format. DrRacket saves a program in plain-text format by default, unless the program contains images or text boxes. (Plain-text format does not preserve images or text boxes.)

  Plain-text format is platform-specific because different platforms have different newline conventions. However, most tools for moving files across platforms support a "text" transfer mode that adjusts newlines correctly.
- *Multimedia file format* — This format is specific to DrRacket, and no other editor recognizes it. DrRacket saves a program in multimedia format by default when the program contains images, text boxes, or formatted text.

  Multimedia format is platform-independent, and it uses an ASCII encoding (so that different ways of transferring the file are unlikely to corrupt the file).

3.5.2 Backup and First Change Files

Every 30 seconds, DrRacket checks each open file. If any file is modified and not saved, DrRacket saves the contents of the file in a backup file, just in case there is a power failure or

some other catastrophic error. If the original file is later saved, or if the user exits DrRacket and explicitly declines to save the file, the backup file is removed. The backup file is saved in the same directory as the original file, and the back file's name is generated from the original file's name:

- On Unix and Mac OS, a `"#"` is added to the start and end of the file's name, then a number is added after the ending `"#"`, and then one more `"#"` is appended after the number. The number is selected to make the backup filename unique.
- On Windows, the file's extension is replaced with a number to make the backup filename unique.

In DrRacket's implementation, these files are called "autosave" files, not backup files.

If the definitions window is modified and there is no current file, then a backup file is written to the user's "documents" directory. The "documents" directory is determined by `(find-system-path 'doc-dir)`.

When DrRacket first starts up, it looks to see if there are any backup files still present and if so, offers to restore the backup files, on the assumption that DrRacket was closed unintentionally. When it restores the files, DrRacket moves the current version of the original file out of the way (adding "autorec" to the name) and the moves the backup file in place of the original file.

Separately, DrRacket creates "first change" files. When you modify an existing file in DrRacket and save it, DrRacket copies the old version of the file to a special first-change file if no such file exists. The first-change file is saved in the same directory as the original file, and the first-change file's name is generated from the original file's name:

- On Unix and Mac OS, a `"~"` is added to the end of the file's name.
- On Windows, the file's extension is replaced with `".bak"`.

3.5.3 Preference Files

On start-up, DrRacket reads configuration information from a preferences file. The name and location of the preferences file depends on the platform and user:

The expression `(find-system-path 'pref-file)` returns the platform- and user-specific preference file path.

- On Unix, preferences are stored in a `".racket"` subdirectory in the user's home directory, in a file `"racket-prefs.rktd"`.
- On Windows, preferences are stored in a file `"racket-prefs.rktd"` in a subdirectory `"Racket"` in the user's application-data folder as specified by the Windows registry; the application-data folder is usually `"Application Data"` in the user's profile directory, and that directory is usually hidden in the Windows GUI.

- On Mac OS, preferences are stored in `"org.racket-lang.prefs.rktd"` in the user's preferences folder.

A lock file is used while modifying the preferences file, and it is created in the same directory as the preferences file. On Windows, the lock file is named `"_LOCKracket-prefs.rktd"`; on Unix, it is `".LOCK.racket-prefs.rktd"`; on Mac OS, it is `".LOCK.org.racket-lang.prefs.rktd"`.

If the user-specific preferences file does not exist, and the file `"racket-prefs.rktd"` in the `"defaults"` collection does exist, then it is used for the initial preference settings. (See §18.2 "Libraries and Collections" for more information about collections.) This file thus allows site-specific configuration for preference defaults. To set up such a configuration, start DrRacket and configure the preferences to your liking. Then, exit DrRacket and copy your preferences file into the `"defaults"` collection as `"racket-prefs.rktd"`. Afterward, users who have no preference settings already will get the preference settings that you chose.

4 Extending DrRacket

DrRacket supports four forms of extension to the programming environment: keybindings, teachpacks, scripts, and plugins.

- The *Keybindings* menu item allows the addition of user defined keybindings. For information on creating user defined keybindings, see §3.3.9 "Defining Custom Shortcuts".
- The *Scripts* menu provided by the Quickscript plugin, makes it easy to extend DrRacket with small Racket scripts that can be used in the definition (or interaction) window, or to graphically interact with the user. For information on creating scripts, see the Quickscript documentation.
- A *teachpack* extends the set of procedures that are built into a language in DrRacket. For example, a teachpack might extend the Beginning Student language with a procedure for playing sounds.

  Teachpacks are particularly useful in a classroom setting, where an instructor can provide a teachpack that is designed for a specific exercise. To use the teachpack, each student must download the teachpack file and select it through the Language|Add Teachpack... menu item.

  See §4.1 "Teachpacks" for information in creating teachpacks.
- A *plugin* extends the set of utilities within the DrRacket environment. For example, DrRacket's Check Syntax button starts a syntax-checking plugin. For information on creating plugins, see the DrRacket Plugins documentation.

Customization of the DrRacket environment is available via *Preferences* menu item, along the the facility to define new color schemes. Existing color-scheme packages are tagged "colorscheme".

4.1 Teachpacks

Teachpacks are designed to supplement student programs with code that cannot be expressed in a teaching language. For example, to enable students to play hangman, we supply a teachpack that

- implements the random choosing of a word,
- maintains the state variable of how many guesses have gone wrong, and
- manages the GUI.

All these tasks are beyond students in the third week and/or impose memorization of currently useless knowledge on students. The essence of the hangman game, however, is not. The use of teachpacks enables the students to implement the interesting part of this exercise and still be able to enjoy today's graphics without the useless memorization.

A single Racket source file defines a teachpack (although the file may access other files via `require`). The file must contain a module (see §6 "Modules"). Each exported syntax definition or value definition from the module is provided as a new primitive form or primitive operation to the user, respectively.

As an example, the following teachpack provides a lazy cons implementation. To test it, save the following in a file and add the file as a teachpack (or use `require`).

```
#lang racket

(provide (rename-out [:lcons lcons]) lcar lcdr)

(define-struct lcons (hd tl))

(define-syntax (:lcons stx)
  (syntax-case stx ()
    [(_ hd-exp tl-exp)
     #'(make-lcons
                (delay hd-exp)
                (delay tl-exp))]))

(define (lcar lcons) (force (lcons-hd lcons)))
(define (lcdr lcons) (force (lcons-tl lcons)))
```

Then, in this program:

```
(define (lmap f l)
  (lcons
   (f (lcar l))
   (lmap f (lcdr l))))

(define all-nums (lcons 1 (lmap add1 all-nums)))
```

the list `all-nums` is bound to an infinite list of ascending numbers.

For more examples, see the `"htdp"` sub-collection in the `"teachpack"` collection of the PLT installation.

4.1.1 Adding Your Own Teachpacks to the Teachpack Dialog

The Language|Add Teachpack... dialog is extensible in two ways. First, users can add teachpacks to the third column by clicking the button at the bottom of the column. These additions are stored in the preferences file, so one way to add site-specific teachpacks is to provide a default preferences file.

The first two columns are also extensible. When a collection has an `"info.rkt"` file (see §6.4 "`"info.rkt"` File Format") that defines `htdp-teachpacks` or `2htdp-teachpacks`, then they are expected to be either a list of (collection-relative) paths containing teachpacks to add to the dialog, or the symbol `'all`, which means that all of the (top-level) files in the collection that end with a module suffix (including `".rkt"`, `".ss"`, or `".scm"`) are teachpacks (except `"info.rkt"` or `"info.ss"`).

4.1.2 Extending Help Desk Search Context

In *How to Design Programs* Teaching Languages, the search context for the Search in Help Desk for ... item in the pop-up menu can be extended by defining the `2htdp:teachpack-modules` binding in the `info.rkt` file (see §6.4 "`"info.rkt"` File Format"). The `2htdp:teachpack-modules` binding should evaluate to a list of symbols representing the module paths to be included in the search context.

For example, the following `info.rkt` file

```
#lang info

(define scribblings '(("scribblings/intro101.scrbl")))

(define 2htdp:teachpack-modules
  '(intro101/file-operations intro101/iterated))
```

includes the modules `intro101/file-operations` and `intro101/iterated` in the help desk search context.

4.2 Environment Variables

Several environment variables can affect DrRacket's behavior:

- `PLTNOTOOLS` : When this environment variable is set, DrRacket doesn't load any tools.
- `PLTONLYTOOL` : When this environment variable is set, DrRacket only loads the tools in the collection named by the value of the environment variable. If the variable is

bound to a parenthesized list of collections, only the tools in those collections are loaded (The contents of the environment variable are `read` and expected to be a single symbol or a list of symbols).

- `PLTDRREPL` : When this environment variable is set, DrRacket starts a read-eval-print loop with all of the forms and functions from the `racket` module and those described in the DrRacket Plugins documentation.

  If it is not set to `-q`, then `(find-system-path 'init-file)` is loaded as well.

  A new thread is created to evaluate REPL expressions, so be sure to use `queue-callback` to evaluate expressions that mutate the GUI (to avoid race-conditions).

- `PLTDRCM` : When this environment variable is set, DrRacket installs the compilation manager before starting up, which means that the `".zo"` files are automatically kept up to date, as DrRacket's (or a tool's) source is modified.

  If the variable is set to `trace` then the files that are actually recompiled are shown.

- `PLTDRPAR` : When this environment variable is set, DrRacket uses `parallel-compile-files` to compile the framework and the drracket collections in parallel and then installs the compilation manager before starting up, which means that the `".zo"` files are automatically kept up to date, as DrRacket's (or a tool's) source is modified.

  If the variable is set to `trace` then the files that are actually recompiled are shown.

- `PLTDRDEBUG` : When this environment variable is set, DrRacket starts up with errortrace enabled. If the variable is set to `profile`, DrRacket also records profiling information about itself.

- `PLTDRPROFILE` : When this environment variable is set, DrRacket uses the `profile` library (with a little GUI) to collect profiling information about itself.

- `PLTDRBREAK` : When this environment variable is set, DrRacket creates a window with a break button, during startup. Clicking the button breaks DrRacket's eventspace's main thread. This works well in combination with `PLTDRDEBUG` since the source locations are reported for the breaks.

- `PLTSTRINGCONSTANTS` : When this environment variable is set, DrRacket prints out the string constants that have not yet been translated. If it is set to a particular language (corresponding to one of the files in `"string-constants"` collection) it only shows the unset string constants matching that language.

  This environment variable must be set when `".zo"` files are made. To ensure that you see its output properly, run `raco setup` with the `-c` flag, set the environment variable, and then run `raco setup` again.

- `PLTDRXREFDELAY` : When this environment variable is set, DrRacket uses an ordinary `delay` (instead of `delay/idle`) to delay the computation of the searching indices. This means that Check Syntax will start more slowly the first time, but that the startup performance is more predictable. In addition, when the environment variable is set,

DrRacket will print out that it is set, and will print when the index is started loading and when it finishes loading.

- `PLTDREASTERSECONDS` : When this environment variable is set, DrRacket pretends that the result of `current-seconds` is actually this environment variable's value, for the purposes of easter eggs. For example, setting it to 1339390801 would simulate King Kamehameha day 2012 and show the corresponding easter egg splash screen.

Index

(define ...) button
".bak"
".LOCK.org.racket-
  lang.prefs.rktd"
".LOCK.racket-prefs.rktd"
".racket"
".rkt"
".sch"
".scm"
".ss"
> prompt
\aleph keyboard shortcut
\all keyboard shortcut
\alpha keyboard shortcut
\amalg keyboard shortcut
\amp keyboard shortcut
\angle keyboard shortcut
\approx keyboard shortcut
\ast keyboard shortcut
\asymp keyboard shortcut
\beta keyboard shortcut
\bigtriangledown keyboard shortcut
\bigtriangleup keyboard shortcut
\blacksmiley keyboard shortcut
\bot keyboard shortcut
\bowtie keyboard shortcut
\bullet keyboard shortcut
\cap keyboard shortcut
\cdot keyboard shortcut
\cdots keyboard shortcut
\chi keyboard shortcut
\circ keyboard shortcut
\clubsuit keyboard shortcut
\cong keyboard shortcut
\coprod keyboard shortcut
\cup keyboard shortcut
\dagger keyboard shortcut
\dashv keyboard shortcut
\ddagger keyboard shortcut
\ddots keyboard shortcut
\Delta keyboard shortcut
\delta keyboard shortcut
\diamond keyboard shortcut
\diamondsuit keyboard shortcut
\div keyboard shortcut
\doteq keyboard shortcut
\Downarrow keyboard shortcut
\downarrow keyboard shortcut
\emptyset keyboard shortcut
\epsilon keyboard shortcut
\equiv keyboard shortcut
\eta keyboard shortcut
\exists keyboard shortcut
\flat keyboard shortcut
\forall keyboard shortcut
\frown keyboard shortcut
\frownie keyboard shortcut
\Gamma keyboard shortcut
\gamma keyboard shortcut
\geq keyboard shortcut
\gg keyboard shortcut
\guillemetleft keyboard shortcut
\guillemetright keyboard shortcut
\hdots keyboard shortcut
\heartsuit keyboard shortcut
\iiint keyboard shortcut
\iint keyboard shortcut
\Im keyboard shortcut
\in keyboard shortcut
\infty keyboard shortcut
\int keyboard shortcut
\invamp keyboard shortcut
\iota keyboard shortcut
\kappa keyboard shortcut
\l keyboard shortcut
\Lambda keyboard shortcut
\lambda keyboard shortcut
\land keyboard shortcut
\langle keyboard shortcut
\ldots keyboard shortcut
\leadsto keyboard shortcut
\Leftarrow keyboard shortcut

\leftarrow keyboard shortcut
\leftharpoondown keyboard shortcut
\leftharpoonup keyboard shortcut
\leftlollipop keyboard shortcut
\leftmultimap keyboard shortcut
\Leftrightarrow keyboard shortcut
\leftrightarrow keyboard shortcut
\leq keyboard shortcut
\ll keyboard shortcut
\lnot keyboard shortcut
\longrightarrow keyboard shortcut
\lor keyboard shortcut
\lozenge keyboard shortcut
\mapsto keyboard shortcut
\models keyboard shortcut
\mp keyboard shortcut
\mu keyboard shortcut
\multimap keyboard shortcut
\multimapinv keyboard shortcut
\nabla keyboard shortcut
\natural keyboard shortcut
\nearrow keyboard shortcut
\neg keyboard shortcut
\neq keyboard shortcut
\newpage keyboard shortcut
\ni keyboard shortcut
\notin keyboard shortcut
\nsubseteq keyboard shortcut
\nu keyboard shortcut
\nwarrow keyboard shortcut
\odot keyboard shortcut
\oiiint keyboard shortcut
\oiint keyboard shortcut
\oint keyboard shortcut
\Omega keyboard shortcut
\omega keyboard shortcut
\ominus keyboard shortcut
\oplus keyboard shortcut
\oslash keyboard shortcut
\otimes keyboard shortcut
\parallel keyboard shortcut
\partial keyboard shortcut
\perp keyboard shortcut
\Phi keyboard shortcut
\phi keyboard shortcut
\Pi keyboard shortcut
\pi keyboard shortcut
\pm keyboard shortcut
\prec keyboard shortcut
\precdot keyboard shortcut
\preceq keyboard shortcut
\prime keyboard shortcut
\prod keyboard shortcut
\propto keyboard shortcut
\Psi keyboard shortcut
\psi keyboard shortcut
\rangle keyboard shortcut
\Re keyboard shortcut
\rho keyboard shortcut
\Rightarrow keyboard shortcut
\rightarrow keyboard shortcut
\rightharpoondown keyboard shortcut
\rightharpoonup keyboard shortcut
\rightleftharpoons keyboard shortcut
\rightlollipop keyboard shortcut
\S keyboard shortcut
\searrow keyboard shortcut
\sharp keyboard shortcut
\Sigma keyboard shortcut
\sigma keyboard shortcut
\sim keyboard shortcut
\simeq keyboard shortcut
\skull keyboard shortcut
\smile keyboard shortcut
\smiley keyboard shortcut
\spadesuit keyboard shortcut
\sqcap keyboard shortcut
\sqcup keyboard shortcut
\sqrt keyboard shortcut
\sqsubsetb keyboard shortcut
\sqsubseteq keyboard shortcut
\sqsupsetb keyboard shortcut
\sqsupseteq keyboard shortcut
\square keyboard shortcut

\star keyboard shortcut
\subset keyboard shortcut
\subseteq keyboard shortcut
\subsetneq keyboard shortcut
\succ keyboard shortcut
\succdot keyboard shortcut
\succeq keyboard shortcut
\sum keyboard shortcut
\supset keyboard shortcut
\supseteq keyboard shortcut
\surd keyboard shortcut
\swarrow keyboard shortcut
\tau keyboard shortcut
\Theta keyboard shortcut
\theta keyboard shortcut
\times keyboard shortcut
\top keyboard shortcut
\triangle keyboard shortcut
\triangleleft keyboard shortcut
\triangleright keyboard shortcut
\Uparrow keyboard shortcut
\uparrow keyboard shortcut
\Updownarrow keyboard shortcut
\updownarrow keyboard shortcut
\uplus keyboard shortcut
\Upsilon keyboard shortcut
\upsilon keyboard shortcut
\varepsilon keyboard shortcut
\varphi keyboard shortcut
\varpi keyboard shortcut
\varrho keyboard shortcut
\varsigma keyboard shortcut
\vartheta keyboard shortcut
\vdash keyboard shortcut
\vdots keyboard shortcut
\vee keyboard shortcut
\wedge keyboard shortcut
\wr keyboard shortcut
\Xi keyboard shortcut
\xi keyboard shortcut
\zeta keyboard shortcut
"_LOCKracket-prefs.rktd"
A-c:down keybinding
A-c:left keybinding
A-c:right keybinding
A-c:up keybinding
About DrRacket... menu item
Add Require Prefix menu item
Add Teachpack... menu item
Add User-defined Keybindings... menu item
Adding Your Own Teachpacks to the Teachpack Dialog
Advanced Student language
alpha renaming
"Application Data"
Ask the Program to Quit menu item
automatic parenthesis
autorec
Backup and First Change Files
Beginning Student language
Beginning Student with List Abbreviations languages
Break button
Bring Frame to Front... menu item
Browser
Buttons
c:+ keybinding
c:\ keybinding, m:\ keybinding, c:x;l keybinding
c:_ keybinding
c:a keybinding
c:b keybinding
c:c;c:e keybinding
c:c;c:o keybinding
c:c;c:z keybinding
c:cc;:l keybinding
c:d keybinding
c:e keybinding
c:f keybinding
c:F6 keybinding
c:h keybinding
c:k keybinding
c:n keybinding
c:p keybinding

c:r keybinding
c:s keybinding
c:t keybinding
c:v keybinding
c:w keybinding
c:x;c:s keybinding
c:x;c:w keybinding
c:x;r;a keybinding
c:x;r;c keybinding
c:x;r;o keybinding
c:x;r;v keybinding
c:x;r;w keybinding
c:x;u keybinding
c:y keybinding
changing a parenthesis as you type
Check for Updates... menu item
Check syntax, question-mark arrows
Check syntax, purple arrows
Check Syntax button
Choose Language... menu item
Choosing a Language
Clear All Teachpacks menu item
Clear Error Highlight menu item
Clear menu item
Close Tab menu item
Close Window menu item
Collapse menu item
Collapse S-expression menu item
Color Schemes
Colors
Comment Out with ... menu item
Comment Out with a Box menu item
Complete Word menu item
Configure Command Line for Racket... menu item
Constant Style
Constructor output
Copy menu item
Create Executable... menu item
Creating Executables
Cut menu item
cycle back through old expressions
Debug button
Debug button
debugger
Debugger Buttons
Debugging Multiple Files
Defining Custom Shortcuts
definitions window
Definitions Window Actions
Delete menu item
Details Pane of Language Dialog
Disable Tests menu item
`display`
distribution archive
Documentation
Documentation menu item
DrRacket Files
DrRacket Teachpacks
DrRacket: The Racket Programming Environment
Edit
Editing
Editing Operations
Editing with Parentheses
Emacs keybindings
Emacs keybindings
Enable Tests menu item
Environment Variables
error highlighting
Errors
evaluating expressions
Evaluation
Expand S-expression menu item
Extending DrRacket
Extending Help Desk Search Context
f1 keybinding
f2 keybinding
f5 keybinding
File
file extension
File Operations
filename button
Find Case Sensitive menu item

Find From Selection menu item
Find Longest Line menu item
Find menu item
Find Next menu item
Find Previous menu item
flashing parenthesis matches
Following Log Messages
Font
Force the Program to Quit menu item
formatting Racket code
General
Go button
Graphical Debugging Interface
Graphical Syntax
gray highlight regions
Help
Hide Column Width Guide for 102 Characters menu item
Hide Definitions menu item
Hide Interactions menu item
Hide Line Numbers menu item
Hide Log menu item
Hide Module Browser menu item
Hide Profile menu item
Hide Program Contour menu item
Hide Tracing menu item
How to Design Programs Teaching Languages
I/O
Images
indenting Racket code
Initial Environment
Input and Output
Insert
Insert Comment Box menu item
Insert Fraction... menu item
Insert Image... menu item
Insert Large Letters... menu item
Insert Racket Box menu item
Insert Racket Splice Box menu item
Insert XML Box menu item
Insert λ menu item
Install Package... menu item
Install PLT File... menu item
Interact with DrRacket in English menu item
Interactions
interactions window
Interface Essentials
Interface Reference
Intermediate Student language
Intermediate Student with Lambda language
Jump to Binding Occurrence menu item
Jump to Next Bound Occurrence menu item
Keybindings
keybindings
Keybindings menu item
Keyboard Shortcuts
Language
Language Declared in Source
language levels
languages, extending
Languages
LaTeX
LaTeX and TeX inspired keybindings
launcher executable
Legacy Languages
Limit Memory... menu item
Log Definitions and Interactions... menu item
m:(keybinding
m:< keybinding
m:> keybinding
m:[keybinding
m:b keybinding
m:c:b keybinding
m:c:d keybinding
m:c:down keybinding
m:c:f keybinding
m:c:k keybinding
m:c:k keybinding
m:c:left keybinding
m:c:m keybinding

m:c:p keybinding
m:c:right keybinding
m:c:space keybinding
m:c:t keybinding
m:c:u keybinding
m:c:up keybinding
m:d keybinding
m:delete keybinding
m:f keybinding
m:h keybinding
m:n keybinding
m:o keybinding
m:p keybinding
m:s:l keybinding
m:t keybinding
m:v keybinding
m:w keybinding
m:{ keybinding
Menus
Module Browser on *file* menu item
Module Browser... menu item
Most Recent Window menu item
Moving Around
Multimedia file format
New menu item
New Tab menu item
Open Defining File menu item
Open Recent menu item
Open Require Path... menu item
Open... menu item
"org.racket-lang.prefs.rktd"
Other Experimental Languages
Out button
Output Printing Styles
Over button
overwrite mode
Package Manager... menu item
Page Setup... menu item
Paste and Indent menu item
Paste menu item
Pause
Plain-text file format
PLT Pretty Big language
PLTDRBREAK
PLTDRCM
PLTDRDEBUG
PLTDREASTERSECONDS
PLTDRPAR
PLTDRPROFILE
PLTDRREPL
PLTDRXREFDELAY
PLTNOTOOLS
PLTONLYTOOL
PLTSTRINGCONSTANTS
plugin
Popup menu
Preference Files
Preferences
Preferences
Preferences... menu item
previous expression
Print Definitions... menu item
Print Interactions... menu item
print output
printing format
Profiling
Program Files
Quasiquote output
R5RS language
Racket
"Racket"
"racket-prefs.rktd"
"racket-prefs.rktd"
"racket-prefs.rktd"
raco dependencies-graph
read
read-char
recycling icon
Redo menu item
Reindent All menu item
Reindent menu item
Related Web Sites menu item
Reload #lang Extensions menu item
Remove Unused Requires menu item

Rename menu item
Reopen Closed Tab menu item
Replace All menu item
Replace menu item
Revert menu item
Run button
Run menu item
Save button
Save Definitions As Text... menu item
Save Definitions As... menu item
Save Definitions menu item
Save Image... menu item
Save Interactions As Text... menu item
Save Interactions As... menu item
Save Interactions menu item
Save Other menu item
Saved Bug Reports menu item
Saved Bug Reports menu item
Scripts
Search
search anchor
Search in Files... menu item
Search in Help Desk for ... menu item
Searching
Select All menu item
Sending Program Fragments to the REPL
Show Active Keybindings menu item
Show Column Width Guide at 102 Characters menu item
Show Definitions menu item
Show Interactions menu item
Show Line Numbers menu item
Show Log menu item
Show Module Browser menu item
Show Profile menu item
Show Program Contour menu item
Show Replace menu item
Show Tracing menu item
Skip to Next Misspelled Word menu item
Spell Check String Constants menu item
Spell Check Text (between {} in Scribble menu item
Spelling Dictionaries menu item
Split menu item
Stack View Pane
stand-alone executable
Status Information
status line
Step button
Step button
Stepper
Submit Bug Report... menu item
Suggest Spelling Corrections... menu item
Swindle language
Tabbed Editing
Tack/Untack Arrow(s) menu item
tail calls
teachpack
Teachpacks
The Interactions Window
The Module Browser
The Racket Language mode
Tool Web Sites menu item
Toolbar Hidden menu item
Toolbar menu item
Toolbar on Left menu item
Toolbar on Right menu item
Toolbar on Top menu item
Tools
Uncomment menu item
Undo menu item
Use Horizontal Layout menu item
Use Vertical Layout menu item
View
Warnings
Windows
Wrap Text menu item
`write`
write output
x-expression
xexpr
XML Boxes and Racket Boxes
"~"
α-rename