


Chapter 2

Shorthand for values: variables

2.1 Defining a variable

You've typed a lot of expressions into the computer involving pictures, but every time you need a different picture, you've needed to find it (e.g. in a Web browser) and copy-and-paste it into DrRacket. This is repetitive and a pain. It would be much more convenient if you could give each picture a name and refer to it that way.

To do this, DrRacket provides a built-in function named `define`. To see how it works, type (in the Interactions pane) the line

```
> (define calendar )
```

and hit ENTER/RETURN. You won't see any "result", but now you can use the word `calendar` any time you want that picture:

```
> calendar
```



```
> (beside calendar calendar)
```



```
> (flip-vertical calendar)
```



(Note that if you leave out the space between `define` and `calendar`, or between `beside` and `calendar`, or between any two typed words, Racket won't know where one word ends and the next begins, and you'll get an error message like *reference to undefined identifier: definecalendar*.) There's nothing magical about the name `calendar` — you could have named it anything else, like `fred` or `antidisestablishmentarianism`, but since it stands for a picture of a calendar, the name `calendar` is a lot easier to remember.

Practice Exercise 2.1.1 *Define another variable to hold another picture you've found*

on the Web. Write some expressions using each of the two variables, and some using both.

You can also define a variable to hold the result of another expression, *e.g.*

```
(define two-calendars (beside calendar calendar))
```

Practice Exercise 2.1.2 Define a variable *six-calendars* whose value is a six-pack of calendars: two wide and three high. Test your definition by typing the name of the variable, all by itself, in the Interactions pane and hitting ENTER/RETURN; you should see the picture of six calendars. If not, you’ve done something wrong.

Hint: This is simpler if you use the already-defined the variable *two-calendars*.

Practice Exercise 2.1.3 Choose a reasonably small picture from this book or the Web, and store it in a variable. Then define another variable named *two-copies* whose value is two copies of that picture, side by side, by using the previous variable. Then define a third variable named *six-copies* whose value is a six-pack of the picture, two wide by three high, by using *two-copies*.

Practice Exercise 2.1.4 Construct another interesting picture from pieces in this book or on the Web, and store it in a variable with an appropriate name. Test it as before. If you use a particular piece more than once, you may want to give that piece a name of its own, so you can use its name in constructing the whole picture.

Common beginner mistakes

Recall from Chapter 1 that a *literal* is a simple expression that doesn’t “stand for” anything else; if you type it into the Interactions window, the value you get back is the same thing you typed in. By contrast, a variable *does* “stand for” something else: if you type in the name of a variable, you get back a picture rather than the variable name you typed.

You can only define a variable; you cannot define a literal. Furthermore, (in Beginner DrRacket) you cannot define a variable more than once: once you’ve decided what value it “stands for”, that’s its value until the next time you click “Run”.

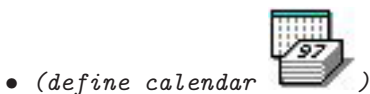
Practice Exercise 2.1.5 *Type* each of the following expressions into a newly-opened interactions pane, one by one. For each one, predict what you think it will do, then hit ENTER and see whether you were right. **Explain** the results. Several of them will produce error messages: in each such case, read and understand the error message.



- `calendar`




- `(define calendar calendar)`





- `calendar`
- `(define other-pic book)`
- `(define other-pic calendar)`



- `(define calendar`  `)`
- `other-pic`

Note that when you type an expression to be evaluated, all variables in it must already be defined, or you get the error message *reference to an identifier before its definition*

On the other hand, when you define a variable, it must *not* already be defined, or you get the error message *define: cannot redefine name*

And of course, when you define a variable, it must be an identifier rather than a literal



like `97`, or you get the error message *define: expected a function name, constant name, or function header for 'define', but found* . . .

2.2 Defining variables and the Definitions pane

As its name suggests, the Definitions pane (normally the top half of the DrRacket window) is intended to hold definitions. Try typing several variable definitions into the Definitions pane, then click the “Run” button. Everything you’ve typed in the Interactions pane will disappear, but the variable definitions are available so you can use them as much as you wish in the Interactions pane. One benefit of this is that, once you’ve found or constructed several interesting pictures, you can save them all to a file so you can easily work with the same named pictures again the next time you start DrRacket.

Practice Exercise 2.2.1 *Type the (legal) definitions from Section 2.1 into the Definitions pane, save it to a file, quit DrRacket, double-click the file, and hit the “Run” button. Type some expressions involving those variables (`calendar`, `two-calendars`, etc.) into the Interactions pane and check that they do what you expect.*

2.3 What’s in a name?

We’ve been using the names of functions for some time already, and now we’re making up new names for things. So what qualifies as a “name”? (The technical term in computer science is *identifier*.) The rules differ slightly from one programming language to another, but

*in Racket, an **identifier** can be made up of letters, numerals, and punctuation marks. It may not contain spaces, commas, # signs, parentheses, square brackets, curly braces, quotation marks, apostrophes, or vertical bars.*

An identifier may contain upper- and/or lower-case letters, and it makes a difference which one you use (*e.g.* `calendar` is different from `Calendar` or `CALENDAR`). An identifier may *not* consist entirely of numerals (if you type such a thing, DrRacket treats it as a number instead; we'll learn more about this in Chapter 3.)

For example, `rotate-cw` is a legal identifier, which happens to represent a predefined function. `rotate$cw` would also be a legal identifier, although it doesn't already stand for anything. `rotate cw` contains a space, so DrRacket would treat it as *two* identifiers, not one.

2.4 More syntax rules

Why are these new expressions, involving `define` and defined variables, legal? Based on the grammar rules you've seen so far (Syntax Rules 1 and 2 from Section 1.8), they're not. So we need some more syntax rules (for easy reference, we repeat the first two):

Syntax Rule 1 *Any literal picture is a legal expression; its value is itself.*

Syntax Rule 2 *A left-parenthesis followed by a function name, one or more legal expressions, and a right parenthesis, is a legal expression. Its value is what you get by applying the named function to the values of the smaller expressions inside it.*

Syntax Rule 3 *Any identifier, if already defined, is a legal expression. Its value is what it was defined to stand for.*

Syntax Rule 4 *A left-parenthesis followed by the word `define`, a previously-undefined identifier, a legal expression, and a right-parenthesis is a legal expression. Think of it as anything matching the pattern*
`(define new-identifier expression)`


It has no "value", but the side effect of defining the variable to stand for the value of the expression.

Worked Exercise 2.4.1 *Draw a box diagram to prove that*

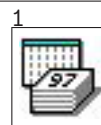
`(define calendar  97)`

is a legal expression. Assume that `calendar` is not already defined.

Solution: The picture is obviously a legal expression, by rule 1:

`(define calendar 1)`

The word `calendar` is a legal identifier. It does not qualify as a legal expression in its own right, since it isn't already defined; however, by rule 4 an undefined identifier can be combined with `define` and the picture to make a legal expression.

⁴`(define calendar )` ■

Exercise 2.4.2 Draw a box diagram to prove that
 (*rotate-cw calendar*)
 is a legal expression. Assume that *calendar* is already defined.

Exercise 2.4.3 Draw a box diagram to prove that your solution to Exercise 2.1.2 or 2.1.3
 is a legal expression.

Exercise 2.4.4 Draw a box diagram for
 (*define snark boojum calendar*)
 Assume that *calendar* is already defined, and *snark* and *boojum* are not. The whole
 thing is not a legal expression; why not?

2.5 Variables and the Stepper

Variable definitions provide another opportunity to see what’s going on “behind the scenes” by using the Stepper. Make sure you’ve got some variable definitions in the Definitions pane, then add some expressions at the end like

```
( beside (flip-vertical calendar) calendar )
```

Click the “Step” button. It’ll skip over all the simple definitions, until you get to an expression involving Syntax Rule 2 (applying a function to something) or 3 (getting the value of a variable). Any variable names in the original expression will be replaced (one by one) with the values of those variables, and then the function (if any) can be applied as usual.

Practice Exercise 2.5.1 Make up some expressions involving variables, type them into the Definitions window, and step through them. At each step, make sure you understand what’s being replaced with what, and why.

2.6 Review of important words and concepts

A **variable** is a word that “stands for” some other value. An **identifier** is any word; it may contain some punctuation marks, but it cannot contain spaces, parentheses, quotation marks, or commas.

Every variable is an identifier, but not every identifier is a variable. For example, **define**, **rotate-180**, **beside**, etc. are identifiers but not variables: they stand for operations rather than values. A variable can be defined by using Syntax Rule 4 on a previously-undefined identifier. After that, you can use it anywhere that you could use any other expression, according to Syntax Rule 2. By chaining a sequence of variable definitions, you can build complex pictures without getting nearly as confused as you might if you tried to write the whole thing as one big expression.

2.7 Reference: functions for defining variables

The only new function introduced in this chapter is **define**.