Lecture 04: Bindings
10:00 AM, Sep 12, 2018

Contents

1 Bindings 1
2 Lambda 1
3 Binding Procedures to Symbols 2

1 Bindings

Expressions can be bound to symbols, so that that expression can be called just by writing the symbol that it has been bound to.

Racket has bindings that exist before any code has been written, called built-ins. These include:

- +, which is bound to the addition procedure
- -, which is bound to the subtraction procedure
- =, which is bound to the equality procedure
- π, which is bound to the number 3.141592653589793

and many more. These bindings all exist within Racket’s top level environment.

More bindings can be added to the top level environment, using definitions. The syntax for these is (define symb expr), which evaluates expr, and then binds that value to symb. This binding is added to the top level environment, where it can be called in the same way that the existing built-in bindings can be. Within an environment, such as the top level environment, bindings can not be overwritten.

2 Lambda

In Racket, a lambda expression evaluates to a procedure. The syntax for a lambda expression is (lambda arglist body), where arglist is a list of the procedure’s formal arguments, and body is the expression that the procedure evaluates once the actual arguments have been passed. The syntax for passing arguments into a lambda expression expr is (expr arg1 arg2 ...), where each arg is an actual argument corresponding to one of expr’s formal arguments.
The application of a procedure has four steps:

1. Create new bindings, binding the procedure-application’s actual arguments to the procedures formal arguments.
2. Evaluate the body of the procedure, using these new bindings.
3. Remove the bindings created in step 1.
4. The value of the procedure-application is the value from step 2.

The result of this evaluation is called the return value.

**Example:** The following expression would evaluate to 9.

```
((lambda (x y) (+ x y)) 5 4)
```

### 3 Binding Procedures to Symbols

An important tenet of functional programming is that a procedure (which a lambda expression evaluates to) is a data object, just like a number or a list is. Exactly like other data objects, a procedure can be bound to a symbol.

**Example:** The lambda procedure from above could be bound to the symbol ‘add’ by following the code below:

```
(define add (lambda (x y) (+ x y)))
```

Then the following application would return the value 9:

```
(add 5 4)
```