Reading Data

September 20, 2018

Reading Data

- Reading and writing to files
- Quote
- Print versus write

Reading input from file

Open a file:

```
(define input (open-input-file “text.txt"))
```

Read a single line from the file:

```
(read-line input)
```

Close file:

```
(close-input-port out)
```

Reading input from file

Open a file:

```
(define input (open-input-file “text.txt"))
```

Read first 100,000 characters of file as a string:

```
(read-string 100000 input)
```

(If file contents are shorter than 100,000, all of the file will be read)

Writing to file

Open a file:

```
(define outfile (open-output-file “text.txt”))
```

Write a string to file:

```
(write “cat” outfile)
```

(Throws an error if file already exists!)

Overwriting to existing file

Open a file:

```
(define outfile
          (open-output-file #:exists ‘truncate “text.txt”)
```

Write a string to file:

```
                              (write “cat” outfile)
```

(‘truncate overwrites existing contents of file)

Appending to existing file

Open a file:

```
(define outfile
         (open-output-file #:exists ‘append “text.txt”)
```

Write a string to file:

```
                              (write “cat” outfile)
```

(‘append appends to end of existing file contents)

Quoting

Quote is a way to express data literals.
Given any Racket expression, quote returns the contents of the expression as data.

The quoted data remains unevaluated.

Quoting

(quote 3)	=> 3	a number
(quote “hi")	=> "hi"	a string
(quote a)	=> a	a symbol
(quote (+ 3 4))	=> (list '+ 3 4)	a list
(quote (a b c))	=> (list 'a 'b 'c)	a list
(quote (define x 25))	=> (list 'define 'x 25)	a list
(quote (lambda (x) (+ x 3))) =>	(list 'lambda (list 'x) (list '+ 'x 3))	a list

Symbols

Quoting a variable name does not produce a string, but another datatype: a symbol.

If we didn't have this datatype, we wouldn't be able to distinguish quoted names from strings.

'(define x 10)	=> (list 'define 'x 10)	'define is a symbol
'("define" x 10)	=> (list "define" 'x 10)	"define" is a string

Writing a Racket program to file

Quoting gives us a way to write out Racket programs without evaluating them— which is exactly what we want to do when we write programs to file.

Shorthand for Quote

' is short-hand for (quote):

```
> (first ' 'road)
'quote

> (first '(quote road))
'quote
```

Print versus Write

So far we've only used (printf), which is a print method for strings.

Racket actually has two distinct print-like methods: print and write. These can be applied to data of any type.

Print versus Write

Print prints a value in the same way that is it printed by the REPL.

Write prints a value in such a way that read on the output produces the value back.

```
>(print #f)
   #f
>(print (quote
             (lambda (x)(x))))
  '(lambda (x)(x))
```

```
>(write #f)
   #f
>(write (quote
             (lambda (x)(x))))
  (lambda (x)(x))
```