15.
The Difference Between Men and Boys....

What is the value of

```
(define x
  (cons (quote chicago)
    (cons (quote pizza)
      (quote ()))))
```

The definitions we have seen so far don't have values. But from now on we will sometimes have to talk about the values of definitions, too.

What does the name x refer to?

(chicago pizza).

What is the value of

```
(set!¹ x (quote gone))
```

It doesn't have a value, but the effect is as if we had just written:

```
(define x (quote gone))
```

[1] L: `setq`, pronounced "set queue"
S: Pronounced "set bang."

Did you notice that **define** is underlined?

We have seen it before. It means that we never actually write this definition. We merely imagine it. But it does replace the two boxes on the left.

What does the name x refer to?

gone.

What is the value of

```
(set! x (quote skins))
```

Remember this doesn't have a value.

Is (**set!** ...) just like (**define** ...)

Yes, mostly.
A (**set!** ...) expression always looks like (**define** ...). The second item is always a name, the last one is always an expression.

And what is x now?

It refers to skins.

What is the value of (*gourmet y*)
where y is onion
and
gourmet is

```
(define gourmet
  (lambda (food)
    (cons food
      (cons x (quote ())))))
```

Which x do you want?

Now what does x refer to?

It still refers to skins.

So what is the value of (*cons x* (**quote** ()))

(skins).

What is the value of
(*gourmet* (**quote** onion))

(onion skins).

```
(set! x (quote rings))
```

It is as if we had written:

```
(define x (quote rings))
```

and as if we had never had any definition of x before.

What is the value of (*gourmet y*)
where y is onion

Which value of x do you want?

And now, what is x

It refers to rings.

What is the value of
(*gourmet* (**quote** onion))

It is (onion rings), since x is now rings.

Look at this:

```
(define gourmand
  (lambda (food)
    (set! x food)
    (cons food
      (cons x
        (quote ())))))
```

What about it?

Is anything unusual?

Yes, the (**lambda** ...) contains two expressions in the value part.

What are they?

The first one is
(**set!** *x food*).
The one after that is

```
(cons food
  (cons x
    (quote ()))).
```

Have we seen something like this before?

Yes, we just saw a (**let** ...) with two expressions in the value part at the end of the previous chapter.

So what do you think is the value of (*gourmand* (**quote** potato))

It is probably the value of the second expression, just as in a (**let** ...) with two expressions.

And that is?

A good guess is (potato potato).

That is correct!

It also means that the value of x is potato.

Yes! And how did that happen?

The first expression
(**set!** *x food*)
means that the definition of *x* changed. It is as if we had written:

```
(define x (quote potato))
```

and as if we had never had any definition of *x* before.

Why?

Because *food* is potato.

What is the value of *x* now?

It is still potato.

What is the value of (*gourmand w*)
where *w* is rice

Now it is easy: (rice rice).

And what is the value of *x* now?

rice, of course.

Does *gourmand* remember what food it saw last?

Yes, *x* always refers to the last food that *gourmand* ate.

Can you write *dinerR* which is like *diner* but also remembers which food it ate last?

```
(define diner
  (lambda (food)
    (cons (quote milkshake)
      (cons food
        (quote ())))))
```

No problem. We can use the same trick.

```
(define dinerR
  (lambda (food)
    (set! x food)
    (cons (quote milkshake)
      (cons food
        (quote ())))))
```

What is the value of (*dinerR* (**quote** onion))

(milkshake onion).

What does *x* refer to now?

onion.

What is the value of
(*dinerR* (**quote** pecanpie))

(milkshake pecanpie).

And now what does x refer to?

pecanpie.

Which do you prefer?

Milkshake and pecan pie.

What is the value of
(*gourmand* (**quote** onion))

We have done this before:
(onion onion).

But, what happened to x

It now refers to onion.

What food did *dinerR* eat last?

Not onion.

How did that happen?

Both *dinerR* and *gourmand* use x to remember the food they saw last.

Should we have chosen a different name when we wrote *dinerR*

Yes, we should have chosen a new name.

Like what?

y.

But what would have happened if *gourmand* had used y to remember the food it saw last?

Well, wouldn't we have the same problem again?

Yes, but don't worry: there is a way to avoid this conflict of names.

There must be, because we should be able to get around such coincidences!

Here is a new function:

```
(define omnivore
  (let ((x (quote minestrone)))
    (lambda (food)
      (set! x food)
      (cons food
        (cons x
          (quote ()))))))
```

It looks like *gourmand*.

True, but not quite. What is the big difference?

Didn't you see the (**let** ...) that surrounds the (**lambda** ...)? Here it is:

```
(let ((x (quote minestrone)))
  (lambda (food)
    ...)).
```

What is the little difference?

The names.

What is the value of

```
(define omnivore
  (let ((x (quote minestrone)))
    (lambda (food)
      (set! x food)
      (cons food
        (cons x
          (quote ()))))))
```

We learned that (**let** ...) names the value of expressions.

What is the value of (**quote** minestrone)

minestrone.

And what is the value part of the (**let** ...)

The value part of this (**let** ...) is a function.

What value does *omnivore* stand for?

We do not know.

That is correct. We need to determine its value.

We have never done this before.

So the definition of *omnivore* is almost like writing two definitions:

```
(define x (quote minestrone))
```

```
(define omnivore
  (lambda (food)
    (set! x food)
    (cons food
      (cons x
        (quote ())))))
```

But it really is this:

```
(define x1 (quote minestrone))
```

```
(define omnivore
  (lambda (food)
    (set! x1 food)
    (cons food
      (cons x1
        (quote ())))))
```

Did you notice that **define** is underlined?	Yes, that's old hat by now.
Did you see the underlined name?	Yes, and that is something new.
What is $\underline{x}_1$	$\underline{x}_1$ is an imaginary name.
Has $\underline{x}_1$ ever been used before with (**define** ...)	No, it has not. And it never, ever will be used with (**define** ...) again.
What does $\underline{x}_1$ refer to?	It stands for minestrone.
So, what is $\underline{x}_1$'s value?	No answer; it is imaginary.
What is the value of *omnivore*	Now it is a function.
What is the value of (*omnivore* z) where z is bouillabaisse	It looks like it is (bouillabaisse bouillabaisse).
What is $\underline{x}_1$'s value?	No answer.
Right?	Always no answer for imaginary names. We just keep in mind what they represent.
What does $\underline{x}_1$ refer to?	It now stands for bouillabaisse.
And why?	After determining the value of (*omnivore* z) where z is bouillabaisse, $\underline{x}_1$ has changed. It is as if we had written: (**define** $\underline{x}_1$ (**quote** bouillabaisse)) and as if we had never had a definition of $\underline{x}_1$ before.

Determining the value of (*omnivore* z) is just like finding the value of (*gourmand* z)

What is the difference?

There is no answer for $\underline{x}_1$

Unlike x, $\underline{x}_1$ is an imaginary name. We must remember what value it represents, because we cannot find out!

The Sixteenth Commandment

Use (set! ...) only with names defined in (let ...)s.

Take a really close look at this:

```
(define gobbler
  (let ((x (quote minestrone)))
    (lambda (food)
      (set! x food)
      (cons food
        (cons x
          (quote ()))))))
```

This looks like *omnivore*.

Not quite. What is the little difference?

The names.

Is there a big difference?

No!

What is the value of

```
(define gobbler
  (let ((x (quote minestrone)))
    (lambda (food)
      (set! x food)
      (cons food
        (cons x
          (quote ()))))))
```

```
(define x2 (quote minestrone))
```

```
(define gobbler
  (lambda (food)
    (set! x2 food)
    (cons food
      (cons x2
        (quote ())))))
```

What is $\underline{x}_2$	$\underline{x}_2$ is another imaginary name.
Has $\underline{x}_2$ ever been used before with (**define** ...)	No, and it never, ever will be used with (**define** ...) again.
What does $\underline{x}_2$ refer to?	It stands for minestrone.
What does $\underline{x}_1$ refer to?	It still stands for bouillabaisse.
So, what is $\underline{x}_2$'s value?	No answer, because $\underline{x}_2$ is imaginary.
What is the value of *gobbler*	It is a function.
What is the value of (*gobbler z*) where z is gumbo	It is (gumbo gumbo).
Now, what is $\underline{x}_2$'s value?	No answer. Ever!
What does $\underline{x}_2$ refer to?	It now stands for gumbo.
And why?	After determining the value of the definition, the definition of $\underline{x}_2$ has changed. It is as if we had written: (**define** $\underline{x}_2$ (**quote** gumbo)) and as if we had never had a value for $\underline{x}_2$ before.
Determining the value of (*gobbler z*) is just like finding the value of (*omnivore z*)	What is the difference?

Here is the function *glutton*

```
(define food (quote none))
```

```
(define glutton
  (lambda (x)
    (set! food x)
    (cons (quote more)
      (cons x
        (cons (quote more)
          (cons x
            (quote ()))))))))
```

Explain in your words what it does.

As you know, we use our words:
"When given a food item, say onion, it builds a list that demands a double portion of this item,
(more onion more onion)
in our example, and also remembers the food item in *food*."

Why does the definition of *glutton* disobey The Seventeenth Commandment?

Recall that we occasionally ignore commandments, because it helps to explain things.

What is the value of
(*glutton* (**quote** garlic))

(more garlic more garlic).

What does *food* refer to

garlic.

Do you remember what x refers to?

onion. In case you forgot, x refers to what *gourmand* or *dinerR* ate last.

Who saw the onion

gourmand.

Can you write the function *chez-nous*, which swaps what x and *food* refer to?

If so, have a snack and join us later for the main meal.

How can *chez-nous* change *food* to what x refers to?

(**set!** *food* x).

How can the function change x to what *food* refers to?

(**set!** x *food*).

How many arguments does *chez-nous* take?

None!

Is this the right way of putting it all together in one definition?

```
(define chez-nous
  (lambda ()
    (set! food x)
    (set! x food)))
```

It is worth a try, but we should check whether it works.

What does *food* refer to?

garlic.

What does x refer to?

onion.

What is the value of (*chez-nous*)

Now, what does *food* refer to

onion.

Now, what does x refer to?

onion.

Did you look closely at the last answer?

We hope so.

Why is the value of x still onion

After changing *food* to the value that x stands for, *chez-nous* changes x to what *food* refers to.

And what does *food* refer to?

onion.

The Eighteenth Commandment

Use (set! x ...) only when the value that x refers to is no longer needed.

How could we save the value in *food* so that it is still around when we need to change x

With (**let** ...).

Explain!

Here is our attempt:
"(**let** ...) names values. If *chez-nous* first names the value in *food*, we have two ways to refer to its value. And we can use the name in (**let** ...) to put this value into x."

Like this?

```
(define chez-nous
  (lambda ()
    (let ((a food))
      (set! food x)
      (set! x a))))
```

Yes, exactly like that.

What is the value of
(*glutton* (**quote** garlic))

(more garlic more garlic).

What does *food* refer to?

garlic.

What is the value of
(*gourmand* (**quote** potato))

(potato potato).

What does x refer to?

potato.

What is the value of (*chez-nous*)

And *food* refers to ...	potato.
But this time, x refers to ...	garlic.
See you later!	Bye for now.
Don't you want anything to eat?	No, that was enough garlic for one day.
If you want something full of garlic, try skordalia.	Perhaps someday.

SKORDALIA

To make 3 cups:

6 cloves to 1 head garlic, peeled

2 cups mashed potatoes (approximately 4 medium potatoes)

4 or more large slices of French- or Italian-type bread, crusts removed, soaked in water, and squeezed dry

1/2 to 3/4 cup olive oil

1/3 to 1/2 cup white vinegar

Pinch of salt

Pound the garlic cloves in a large wooden mortar with a pestle until thoroughly mashed. Continue pounding while adding the potatoes and bread very gradually, beating until the mixture resembles a paste. Slowly add the oil, alternating with the vinegar, beating thoroughly after each addition until well absorbed. Add salt, taste for seasoning, and beat until the sauce is very thick and smooth, adding more vinegar or soaked squeezed bread, if necessary. Then scoop into a serving bowl. Cover and refrigerate until ready to use. Use as a dip for beets, zucchini, and eggplant.

THE FOOD OF GREECE
Vilma Liacours Chentiles
Avenel Books, New York, 1975