mutable data

`define`

versus

`set!`

An **identifier**
is a name used in the code.

A **binding**
is a connection to a value.

A **variable**
is an identifier + binding.

.

How do functions retain access to objects that existed when the function was created?

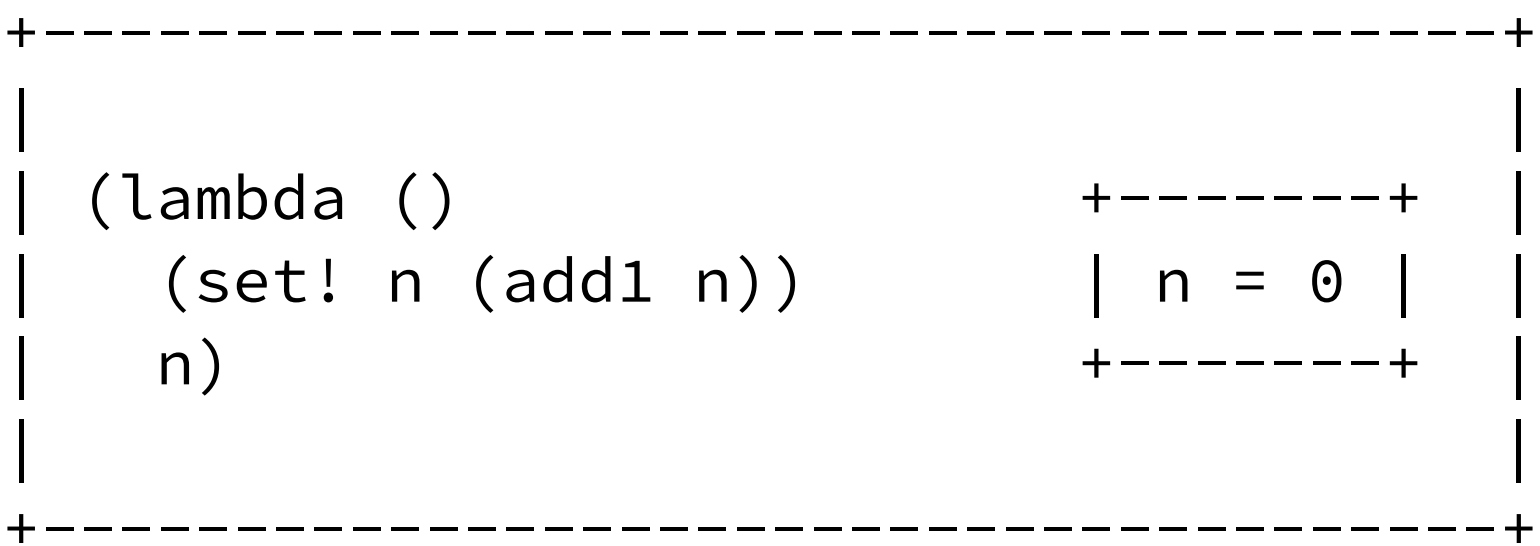
The interpreter creates a **closure**.

.

A closure is a data structure:

.

```
(define counter
  (let ((n 0))
    (lambda ()
      (set! n (add1 n))
      n)))
```



.

Now we can understand how the **region** of a variable is ***not*** the same as the **scope** of the variable.

.

"The American embassy in Paris occupies a very nice building on the Place de la Concorde. Certainly, the embassy is physically within the boundaries of France. But when you step inside the embassy, what country are you in? You're no longer in France. You're in the United States, and US law applies."

.

```
(define make-counter
  (lambda ()
    (let ((n 0))
      (lambda ()
        (set! n (add1 n))
        n))))
```

[demo in Dr. Racket]

```
(let ((n 42))
  (let ((clock-tick (make-counter)))
    ...
    (clock-tick)
    ... ))
```

.

One approach is to use

message-passing style

Create a function that receives a symbol as its argument and uses the symbol to choose which procedure to run.

.

```
(case transaction
  ('withdraw ...)
  ('deposit  ...))
```

is equivalent to

```
(cond ((eq? transaction 'withdraw) ...)
      ((eq? transaction 'deposit ) ...))
```

.