CSE 1729:Principles of Programming

Lecture 7: Lexical Scope in Scheme

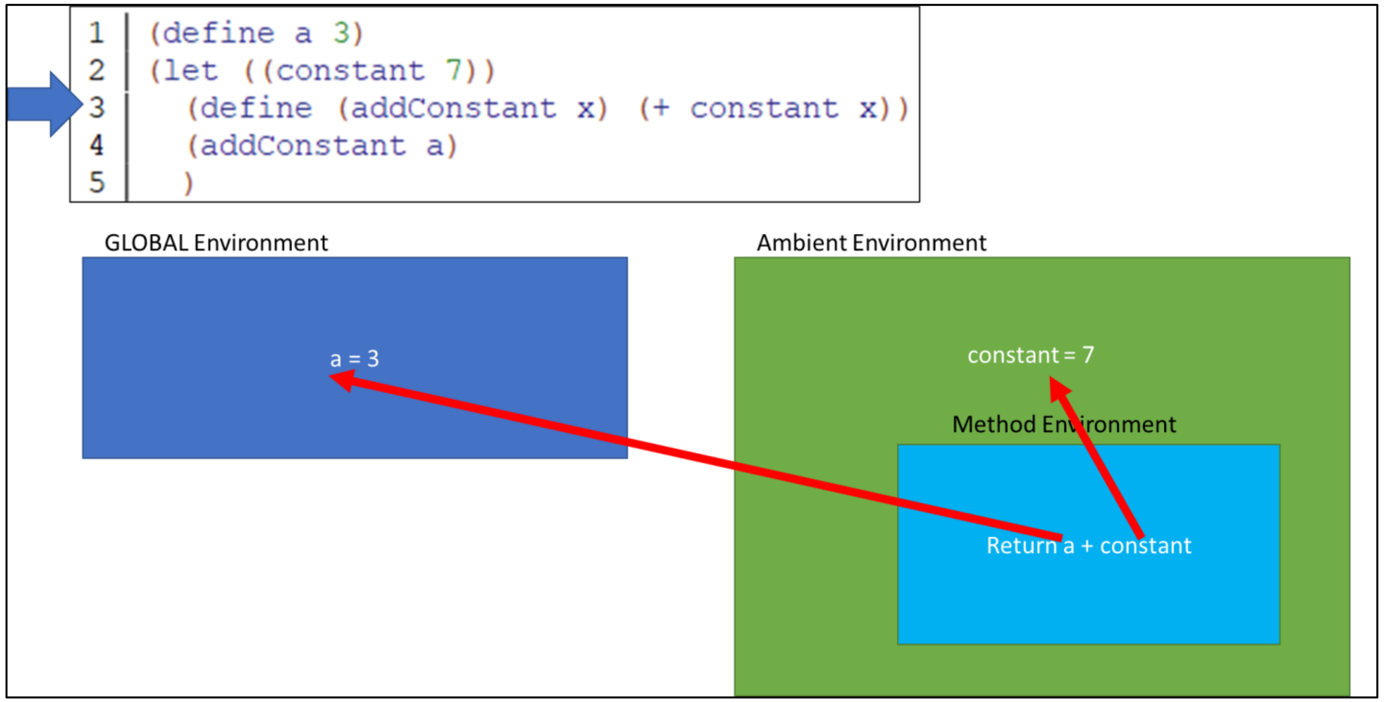
Kaleel Mahmood

Department of Computer Science and Engineering

University of Connecticut

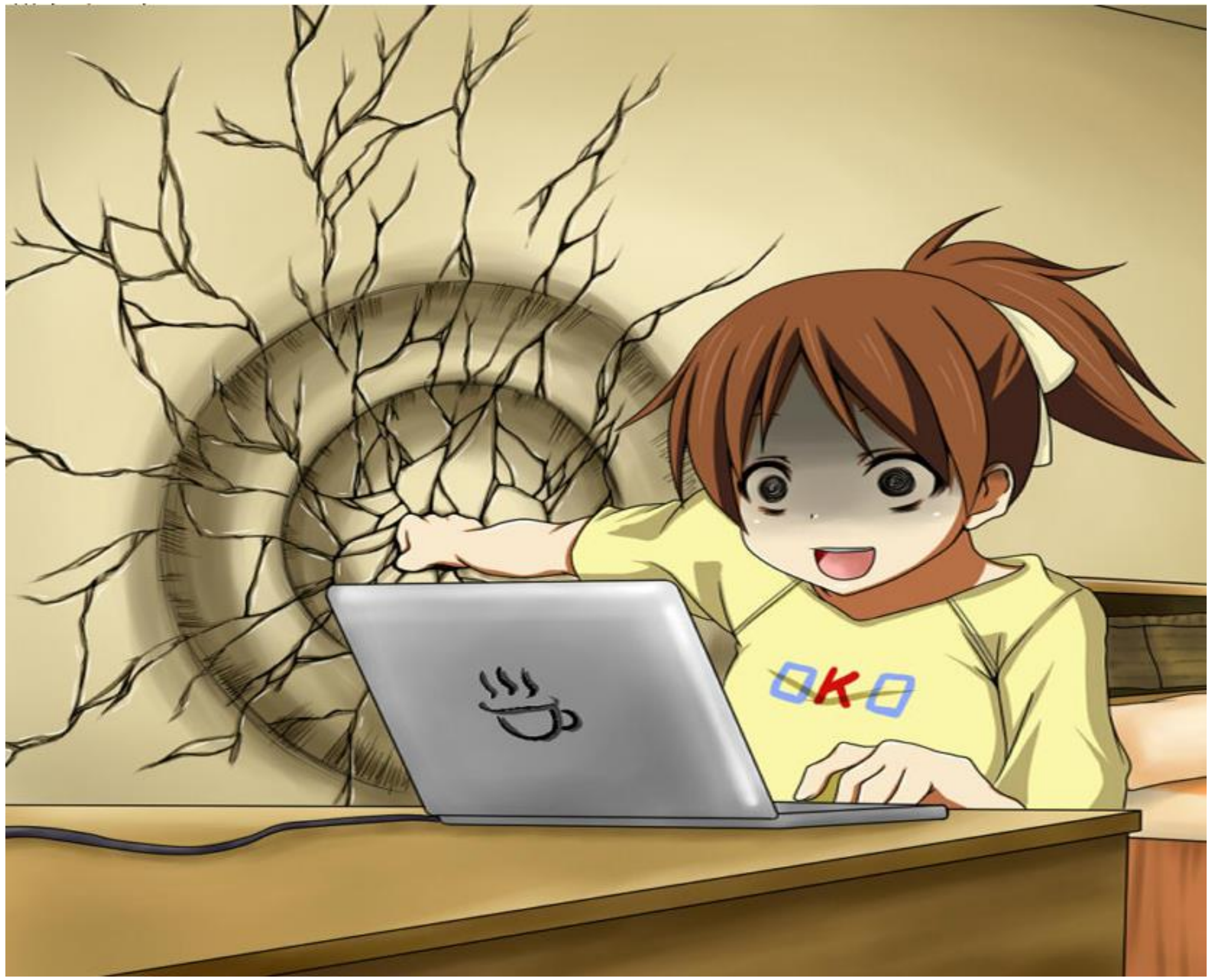
Previously on CSE 1729….

We introduced the "let" statement and talked about how they create "Ambient Environments"



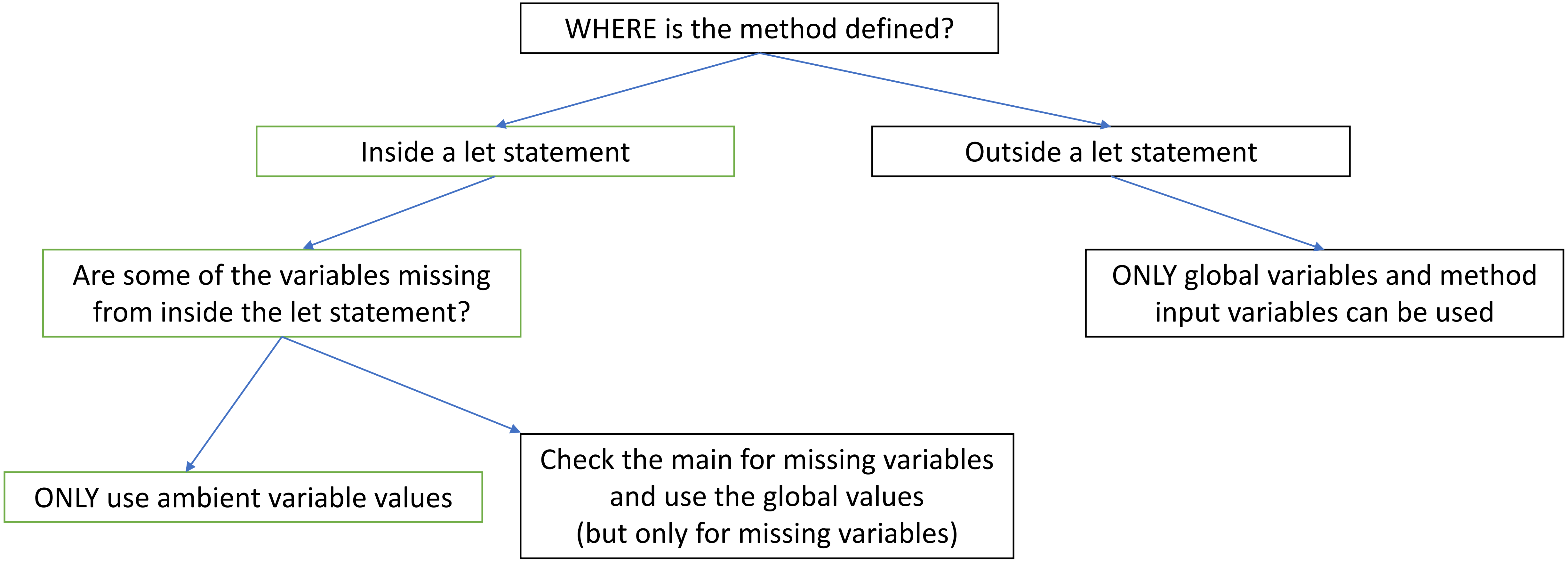
```
(define a 3)
(let ((constant 7))
  (define (addConstant x) (+ constant x))
  (addConstant a)
  )
```

Pretty much everyone's reaction to Scheme at this point…

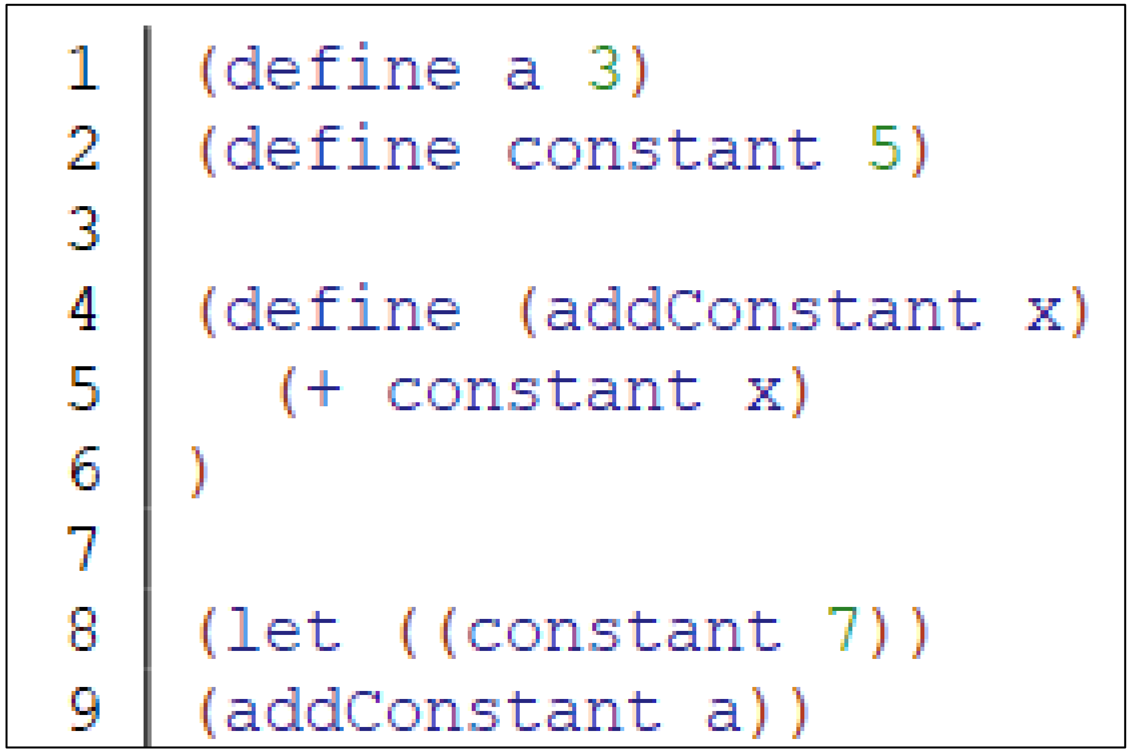


Question: WHY do we want to use the let statement?

A Guide to figuring out Method and Environment Variables


WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Tracing through an example…



```
(define a 3)
(define constant 5)

(define (addConstant x)
  (+ constant x)
)

(let ((constant 7))
(addConstant a))
```

Step 1: Write where the variables live

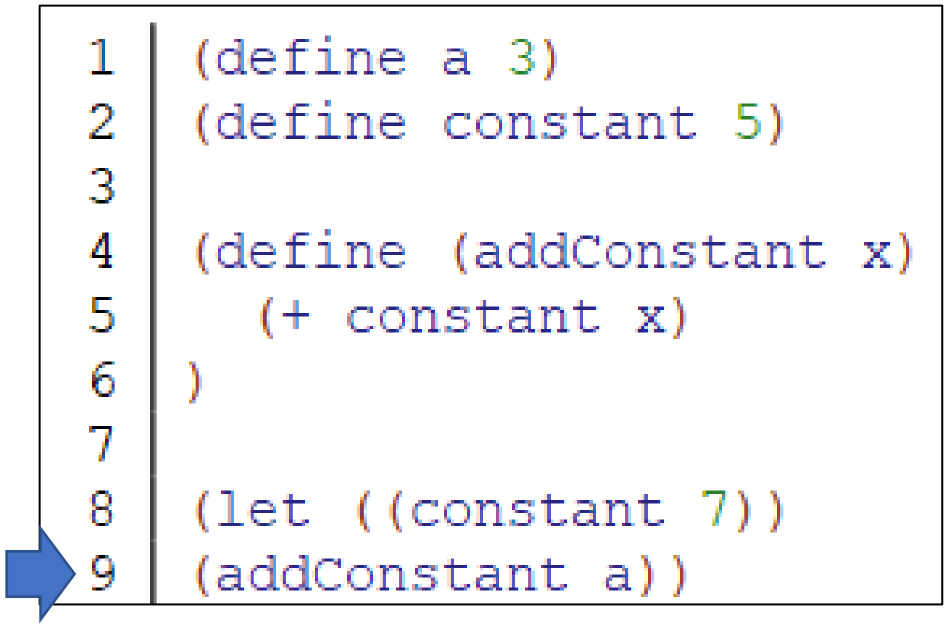
GLOBAL Environment

a= 3
constant = 5

Ambient Environment

constant = 7

Step 2: Use the Guide:



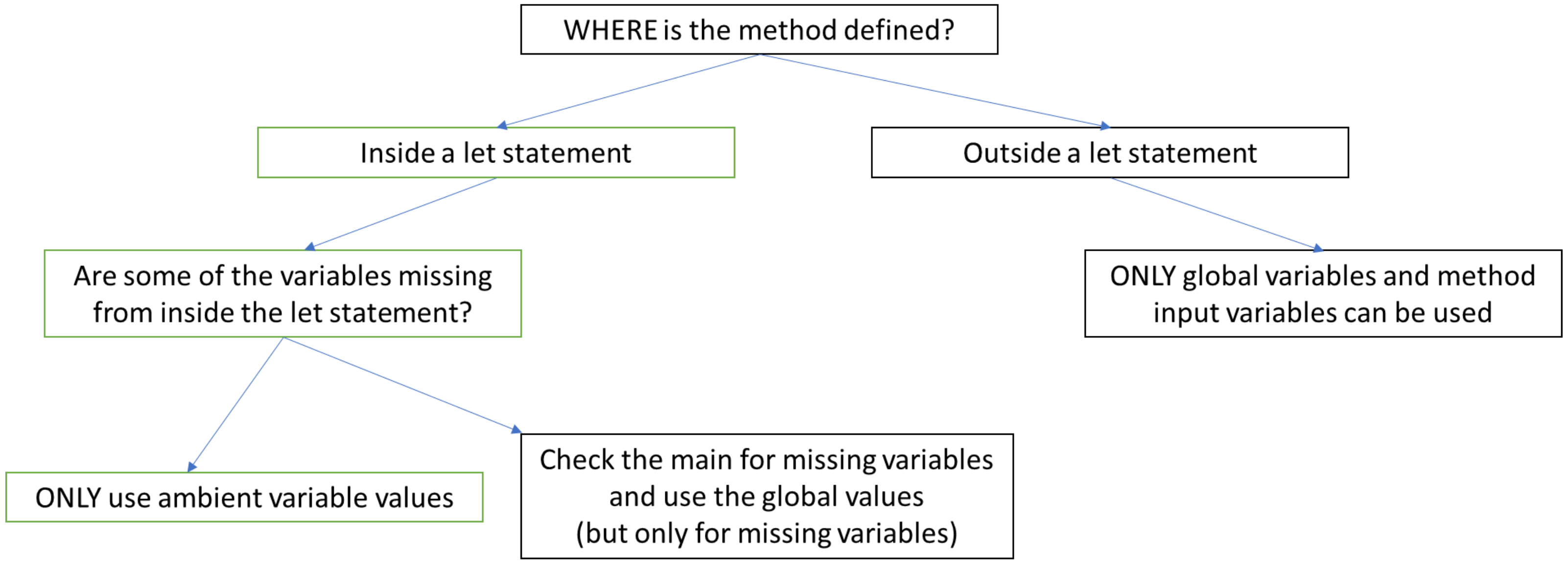
```
(define a 3)
(define constant 5)

(define (addConstant x)
  (+ constant x)
)

(let ((constant 7))
(addConstant a))
```

GLOBAL Environment

a= 3
constant = 5

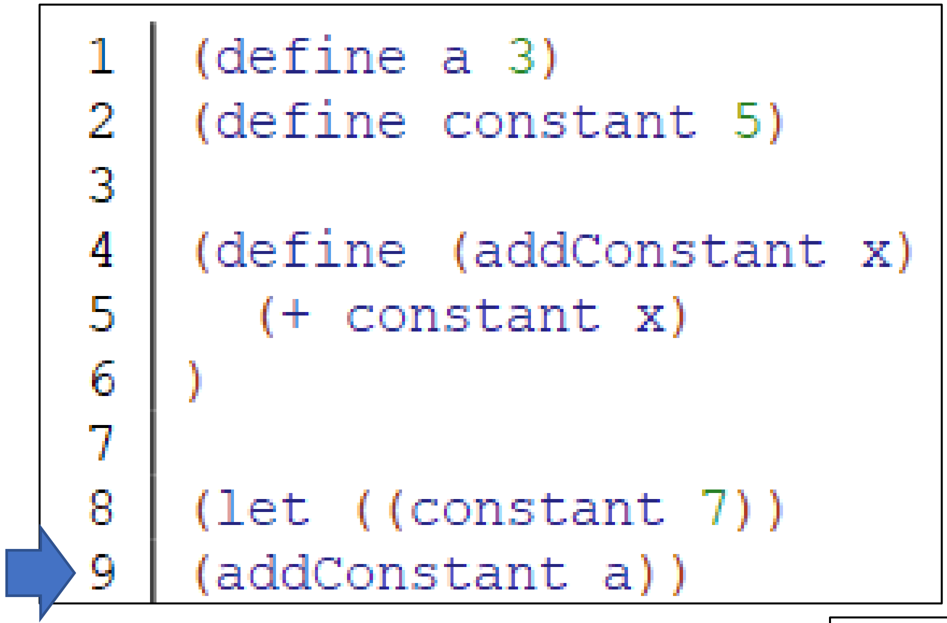
Ambient Environment

constant = 7



WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Step 2: Use the Guide:



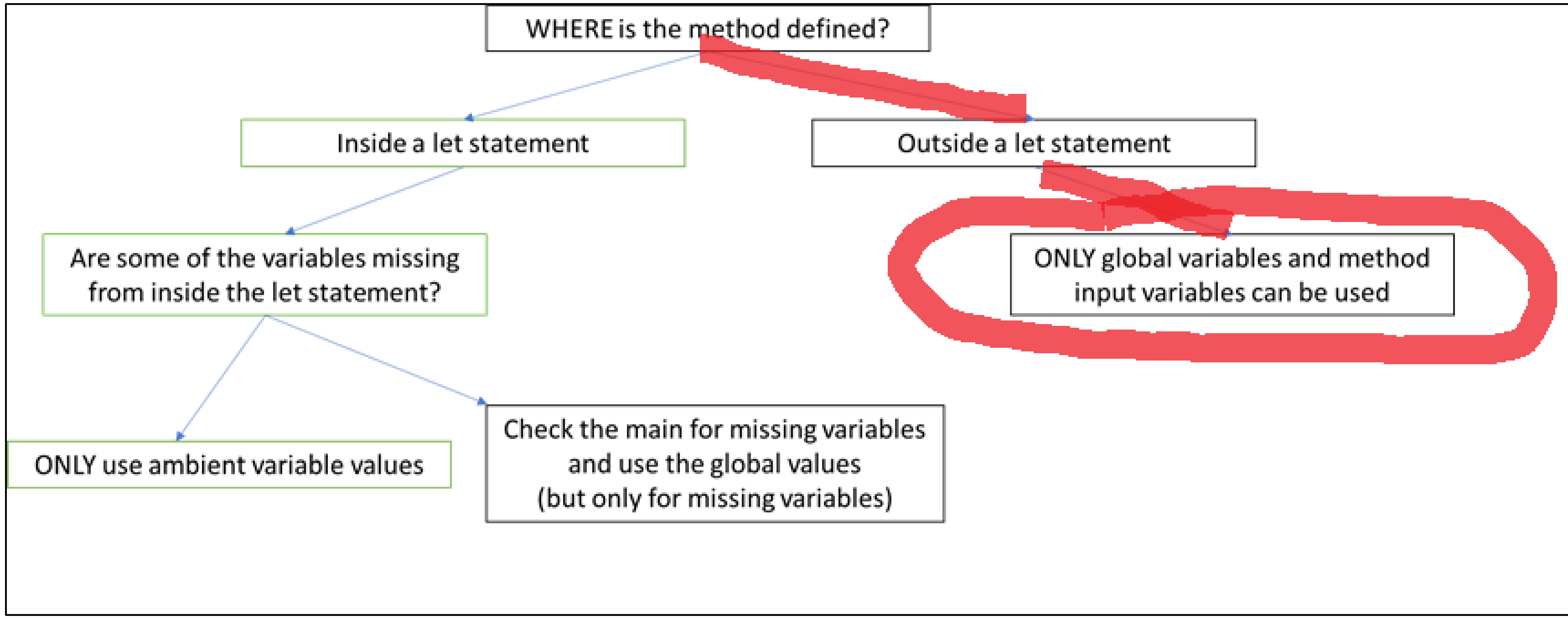
```
(define a 3)
(define constant 5)

(define (addConstant x)
  (+ constant x)
)

(let ((constant 7))
(addConstant a))
```

GLOBAL Environment

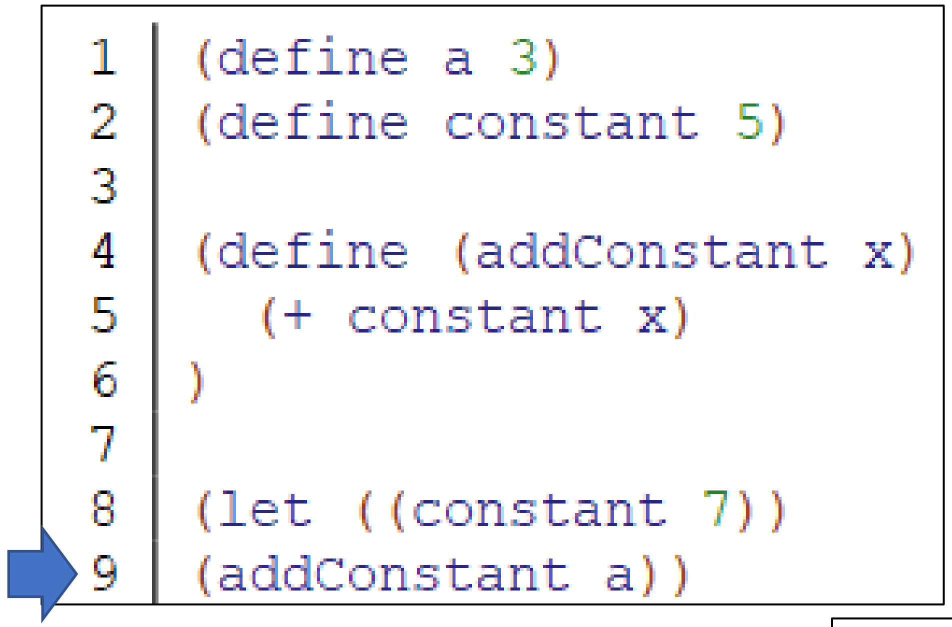
a= 3
constant = 5

Ambient Environment

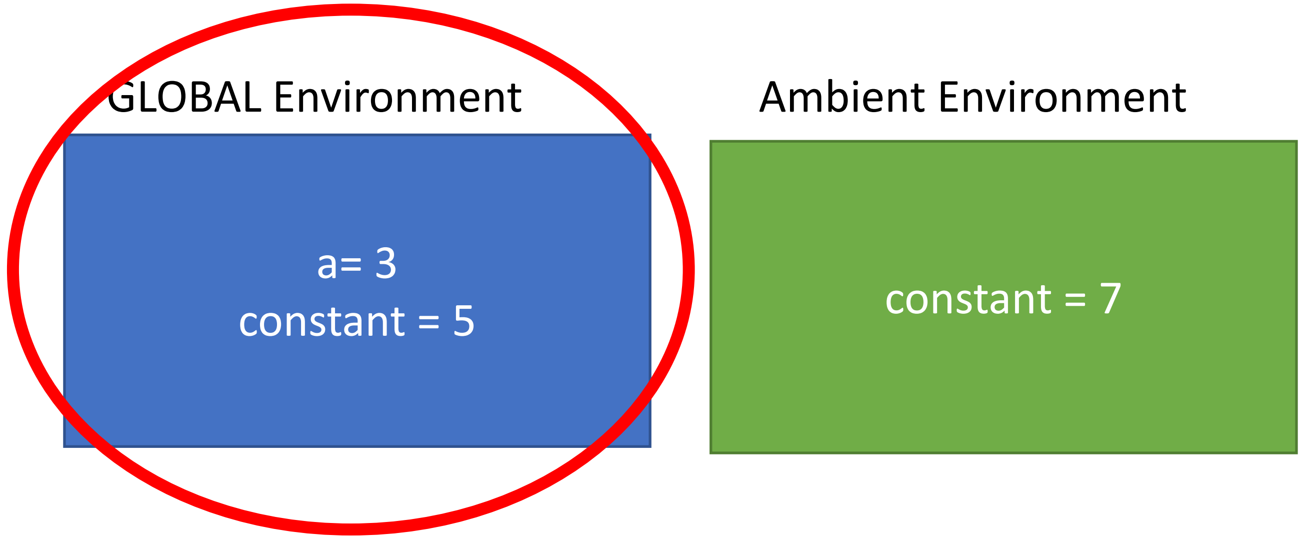
constant = 7

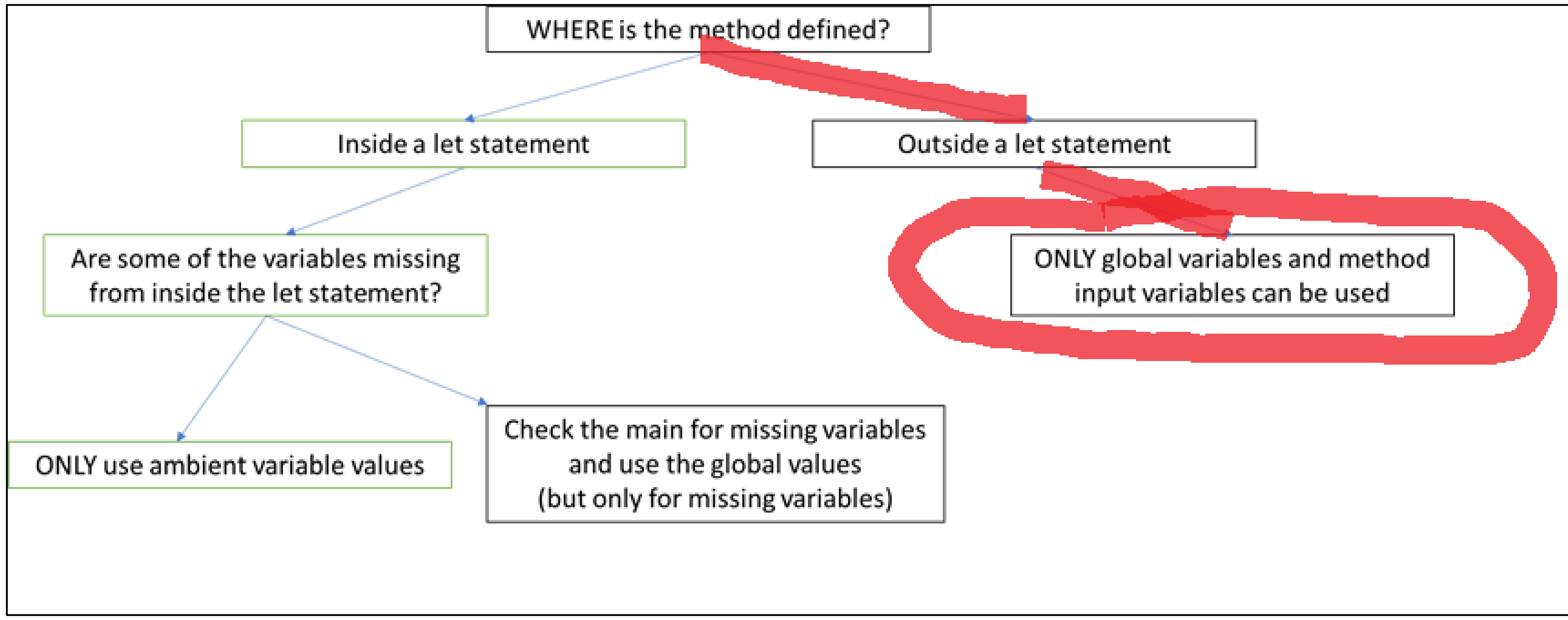

WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Step 2: Use the Guide:



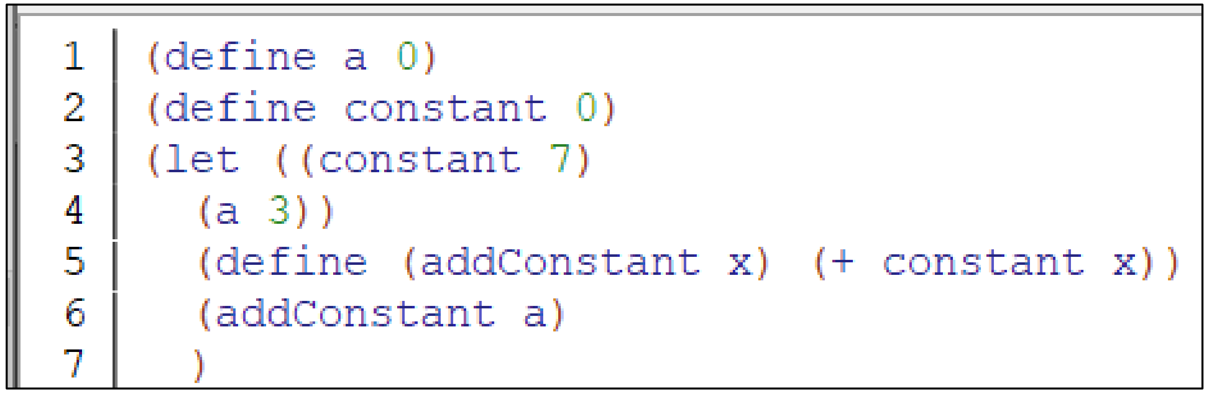
```
(define a 3)
(define constant 5)

(define (addConstant x)
  (+ constant x)
)

(let ((constant 7))
(addConstant a))
```





WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Tracing through an example 2…



```scheme
(define a 0)
(define constant 0)
(let ((constant 7)
  (a 3))
  (define (addConstant x) (+ constant x))
  (addConstant a)
  )
```

Step 1: Write where the variables live

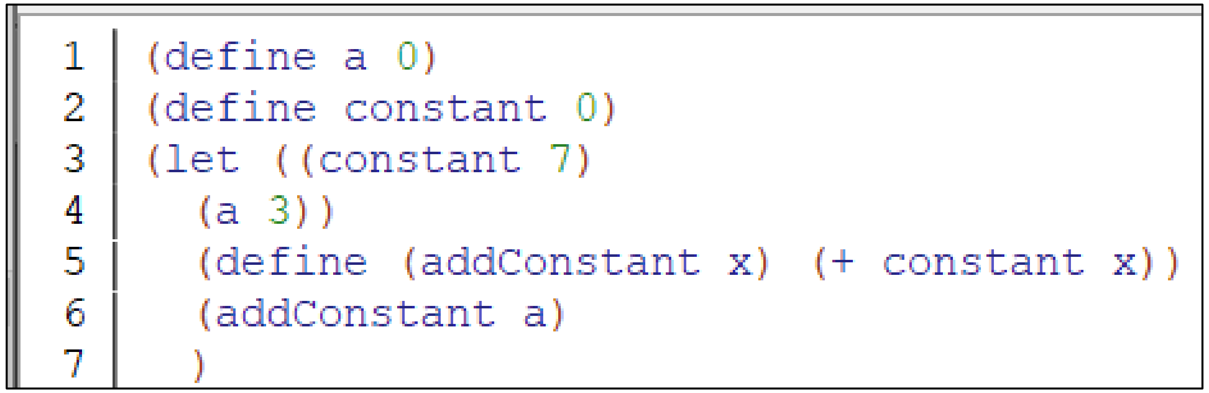
GLOBAL Environment

a= 0
constant = 0

Ambient Environment

constant = 7
a = 3

Tracing through an example 2…



```
(define a 0)
(define constant 0)
(let ((constant 7)
  (a 3))
  (define (addConstant x) (+ constant x))
  (addConstant a)
  )
```

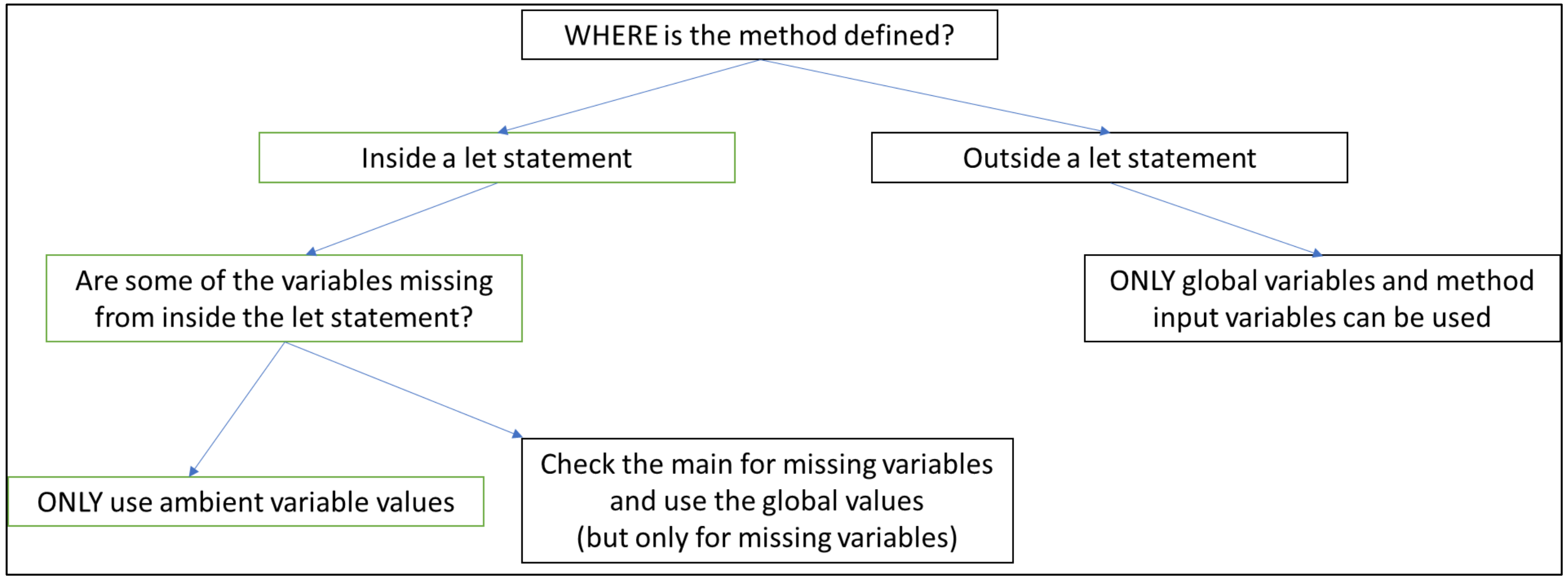
Step 1: Write where the variables live

GLOBAL Environment

a= 0
constant = 0

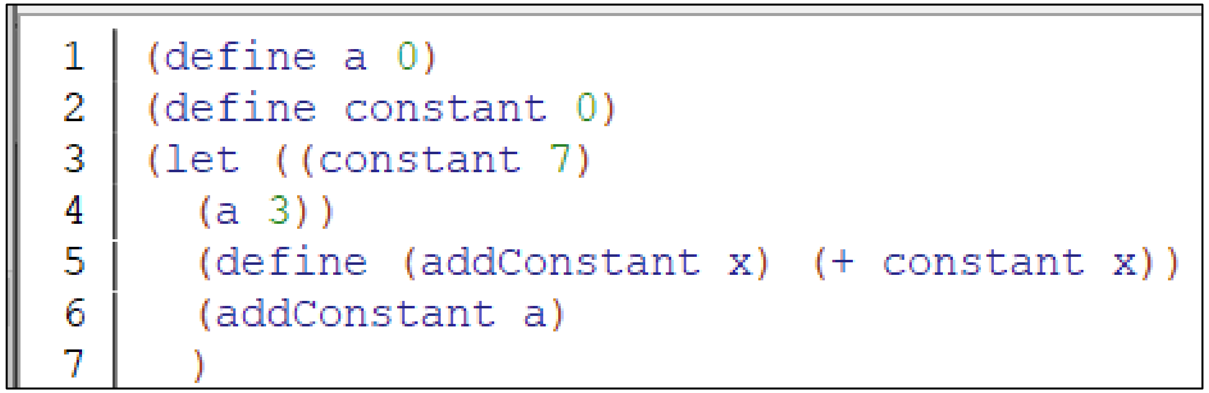
Ambient Environment

constant = 7
a = 3



WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Tracing through an example 2…



```
(define a 0)
(define constant 0)
(let ((constant 7)
  (a 3))
  (define (addConstant x) (+ constant x))
  (addConstant a)
  )
```

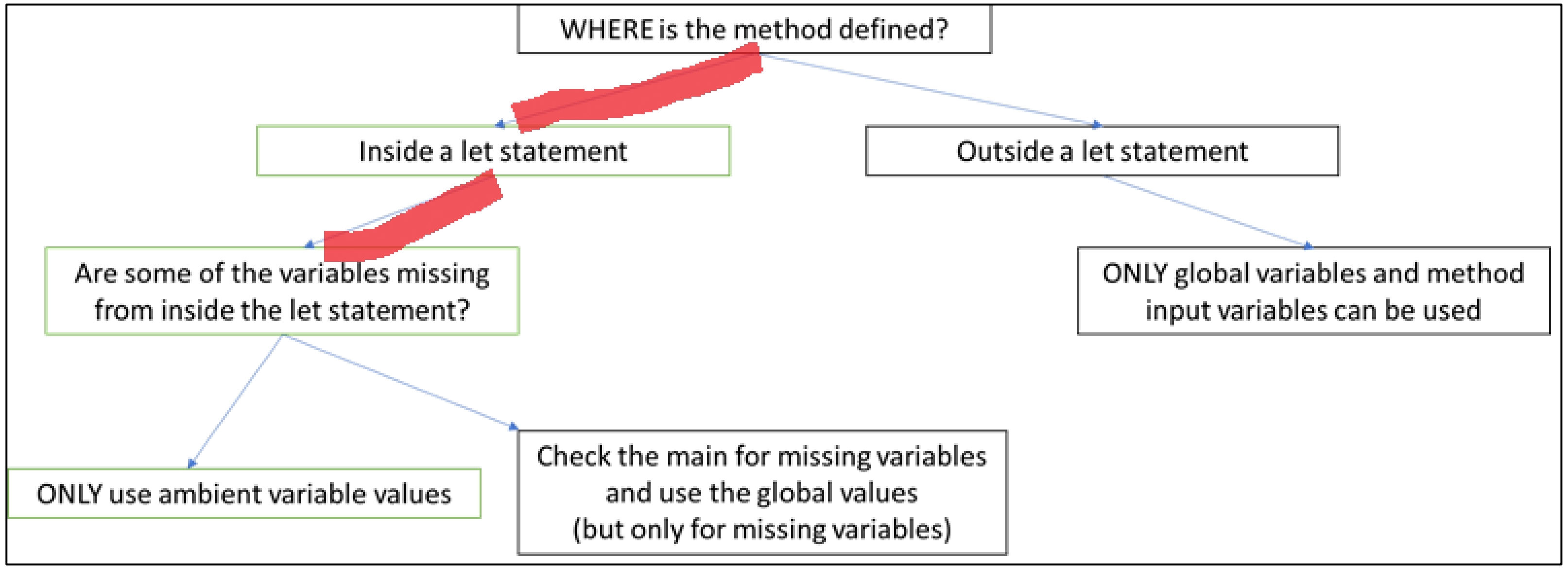
Step 1: Write where the variables live

GLOBAL Environment

a= 0
constant = 0

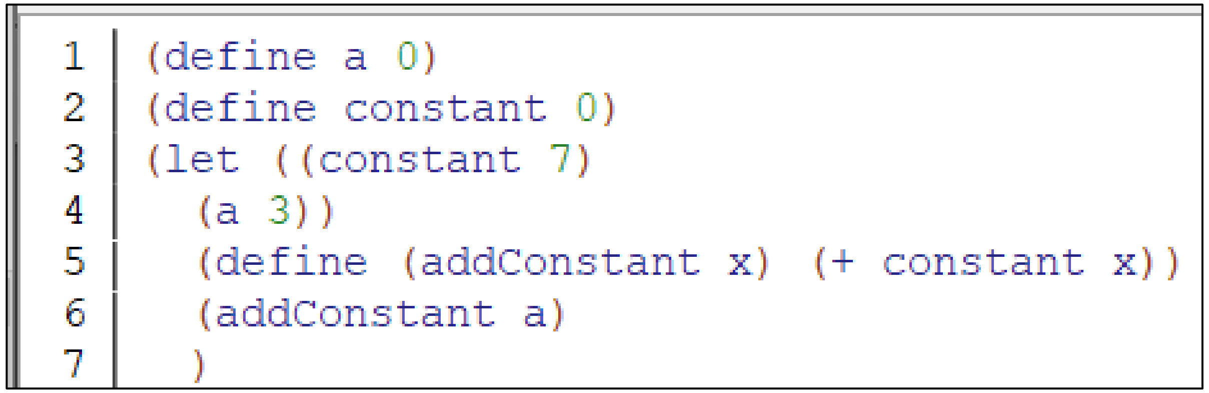
Ambient Environment

constant = 7
a = 3



WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Tracing through an example 2...



```
(define a 0)
(define constant 0)
(let ((constant 7)
  (a 3))
  (define (addConstant x) (+ constant x))
  (addConstant a)
  )
```

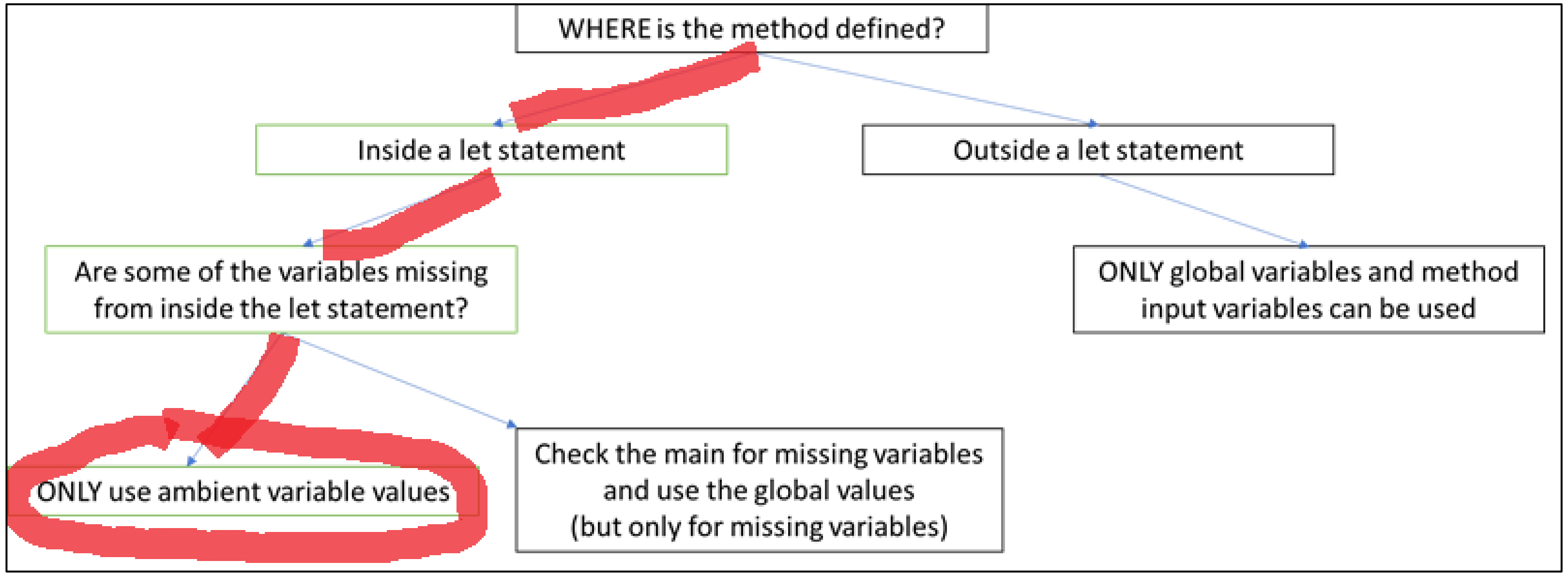
Step 1: Write where the variables live

GLOBAL Environment

a= 0
constant = 0

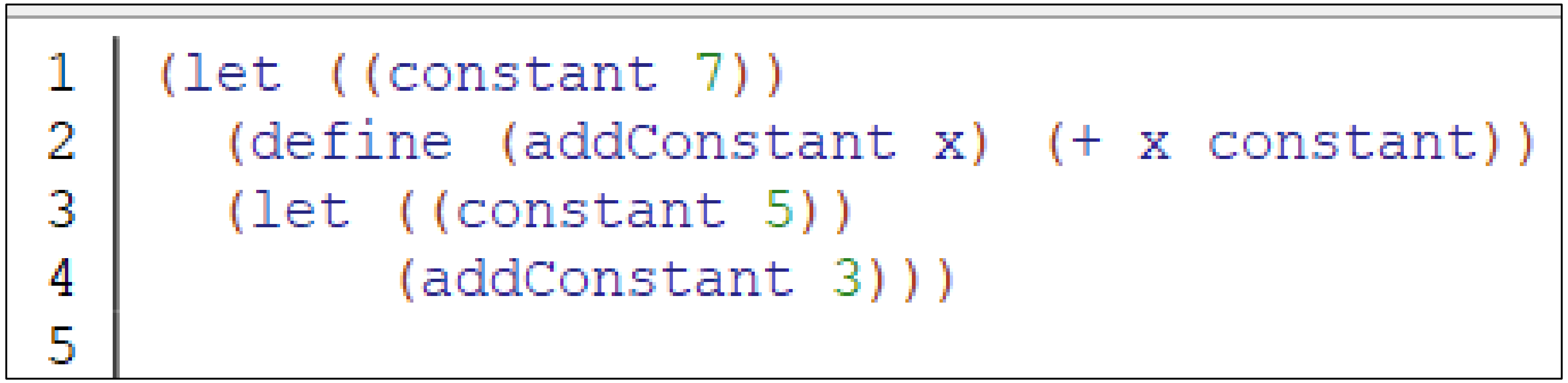
Ambient Environment

constant = 7
a = 3



WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

Think you are getting the trick? Here is a tough one…

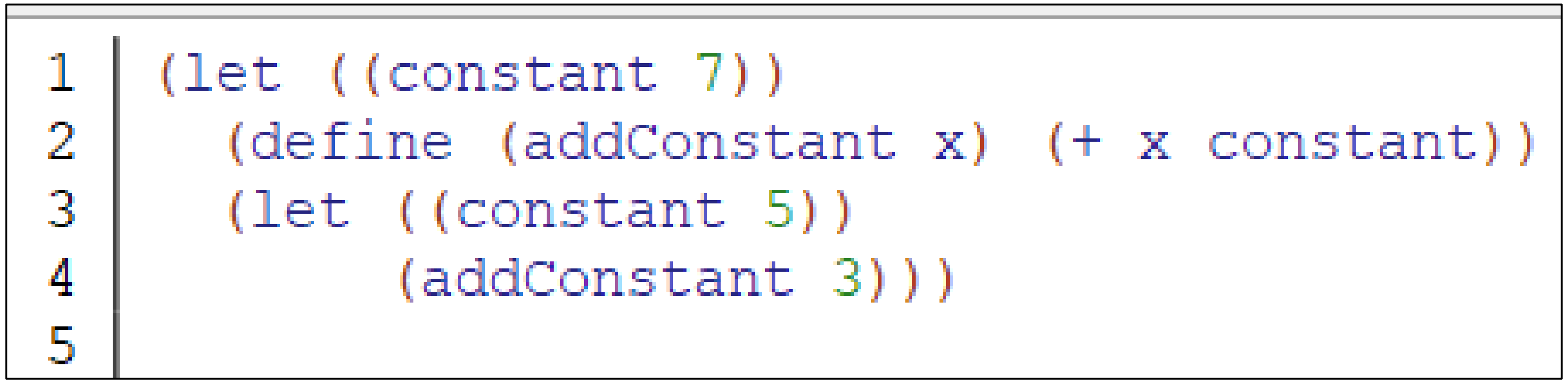


```
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant 3)))
```

Hint: Let statements do NOT share the same ambient environment.

So should the answer be 8 or 10?

Think you are getting the trick? Here is a tough one…

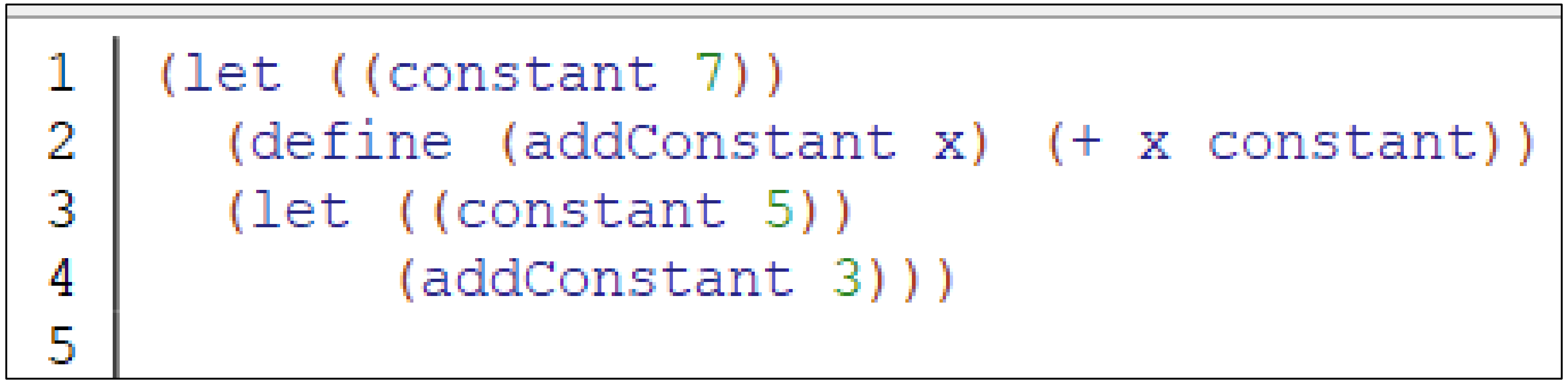


```
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant 3)))

```

<u>Step 1: Write where the variables live</u>

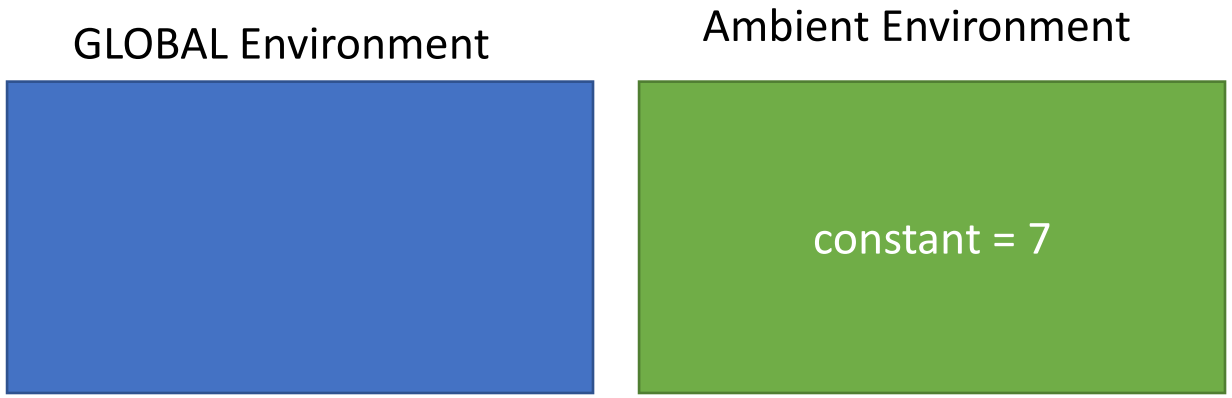
GLOBAL Environment

Think you are getting the trick? Here is a tough one…

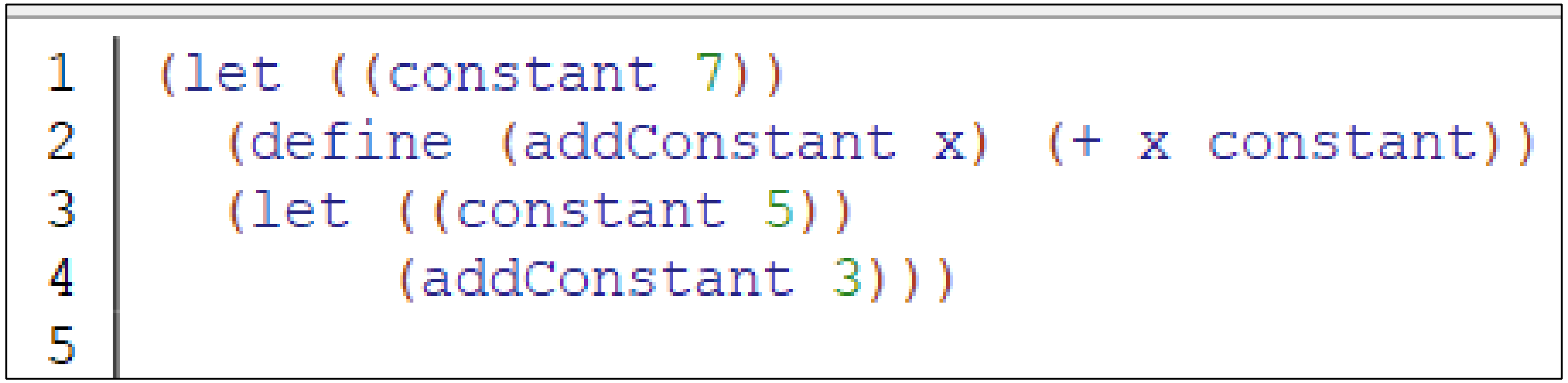


```
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant 3)))
```

Step 1: Write where the variables live

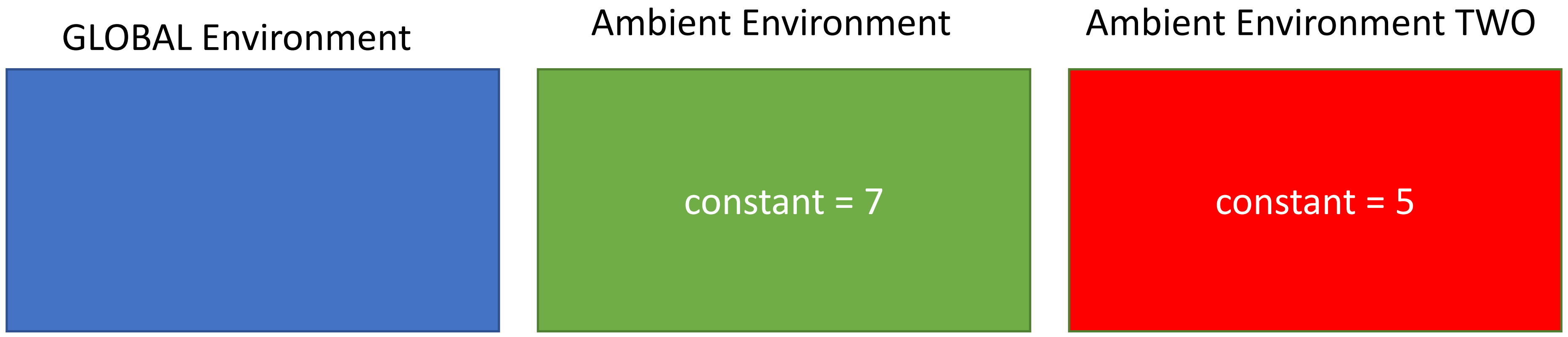


Think you are getting the trick? Here is a tough one…

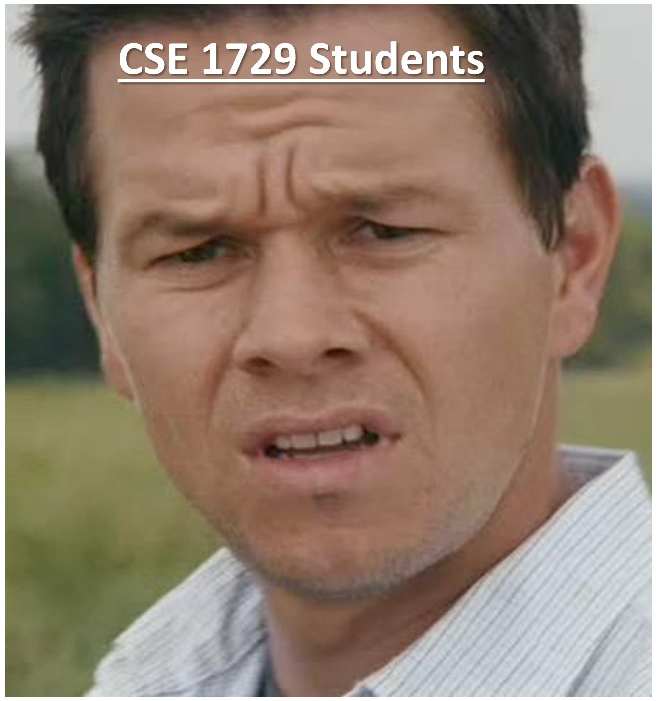


```
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant 3)))
```

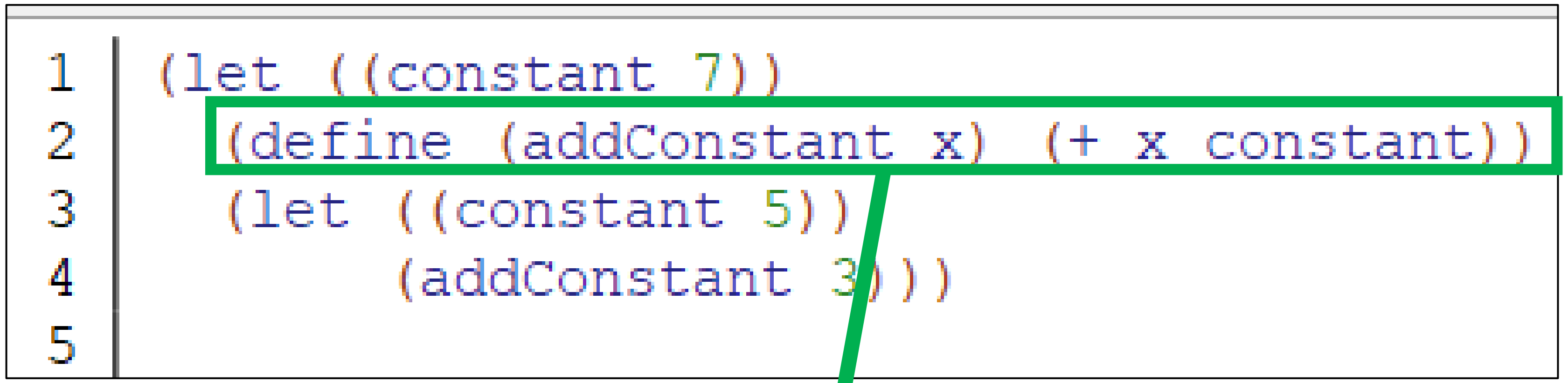
Step 1: Write where the variables live



Which constant do we use?!?

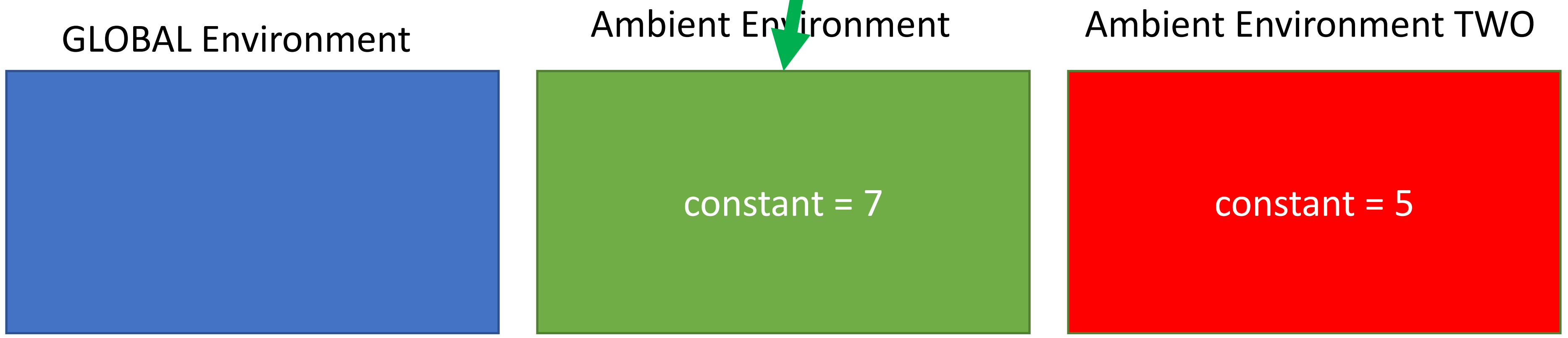


To answer that question: *We need to ask WHERE was add constant method defined?*

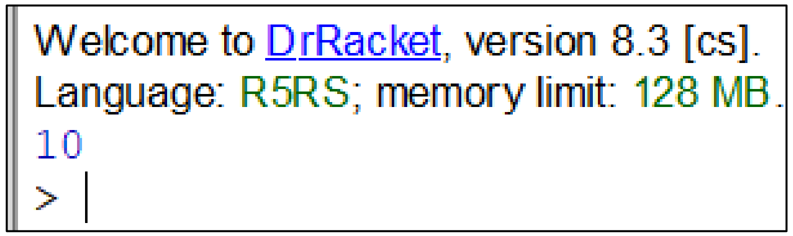


```
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant 3)))
```

Step 1: Write where the variables live



Solution:

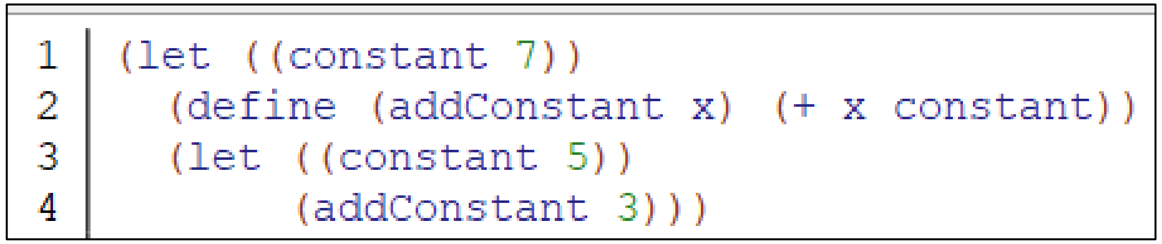


```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
10
> 
```

What can we conclude from this example?

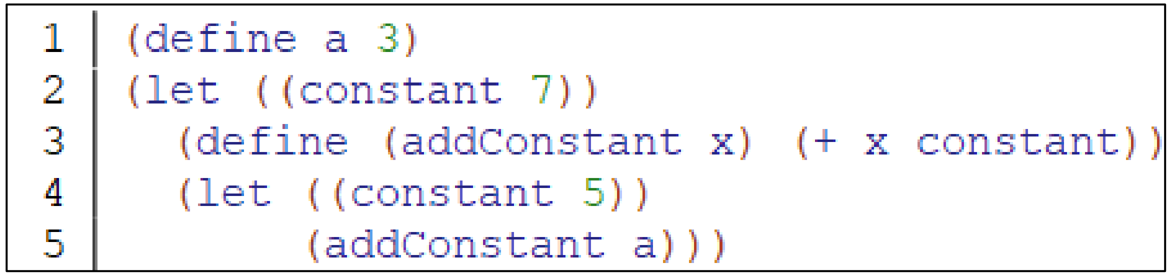
- Ambient environments CANNOT share variables between one another freely. But would this work?

<u>Old code:</u>



```
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant 3)))
```

<u>New code:</u>

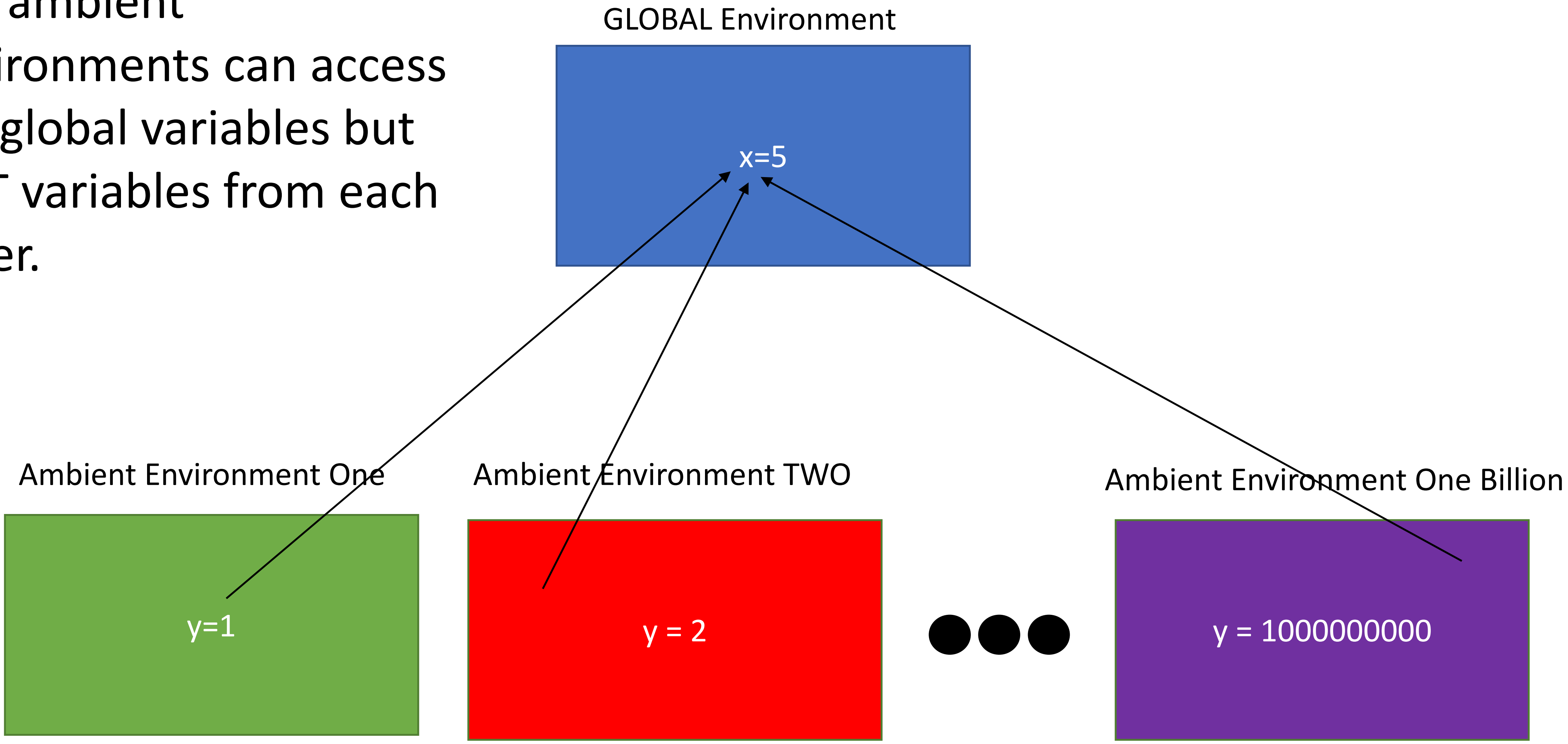


```
(define a 3)
(let ((constant 7))
  (define (addConstant x) (+ x constant))
  (let ((constant 5))
       (addConstant a)))
```

Yes. Why? Because the global variables are accessible by all environments.

A picture to draw in your head

The ambient environments can access the global variables but NOT variables from each other.



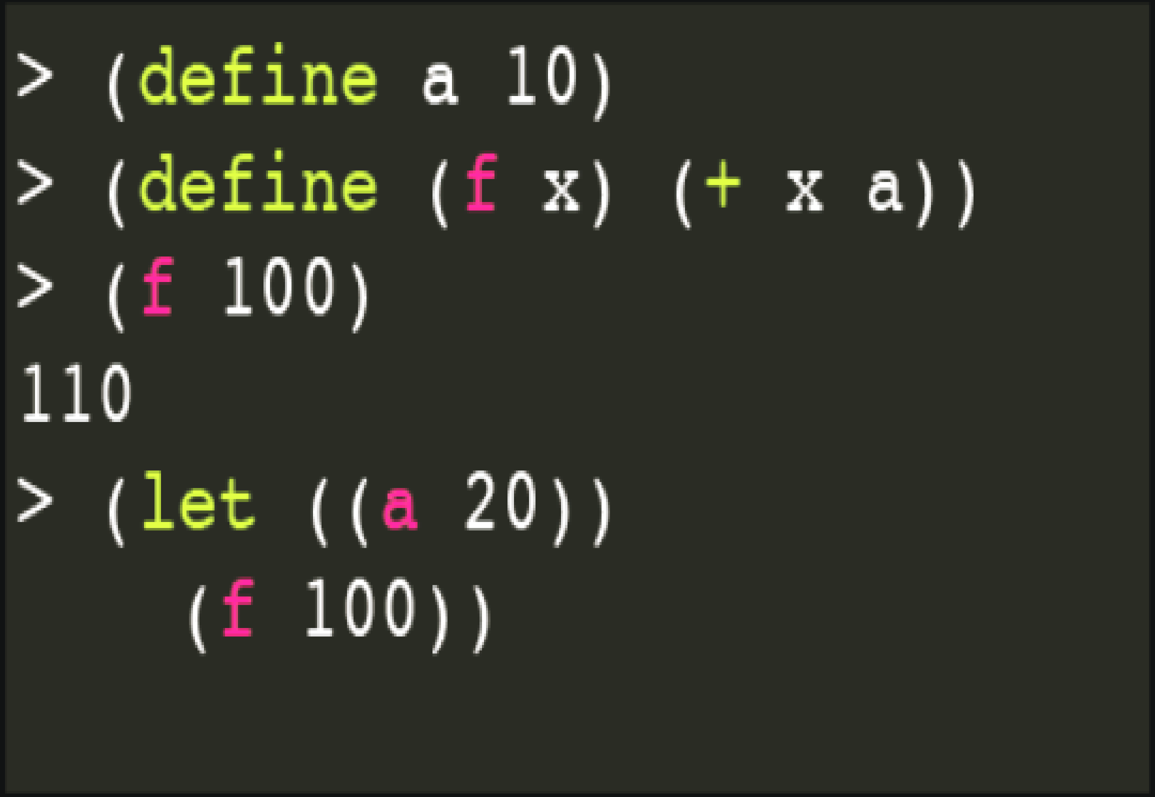
LEXICAL SCOPE AND VARIABLE CLASHES

- Scheme uses a precise set of rules to determine the binding of a variable.
- These conditions are known as *scoping rules* for the binding.
- Scheme uses *lexical scope*.
- The other natural choice is *dynamic scope*.
- Example:

```
> (let ((x 10))
    (define (g y) (+ x y))
    (let ((x 100))
         (g 1000)))

1010
```

- Two potentially relevant environments:
  - The environment at the time of *definition* (in which x = 10).
  - The environment at time of *invocation* (in which x = 100).
- Lexical scoping rules (which Scheme uses) always rely on the environment at *definition time*.

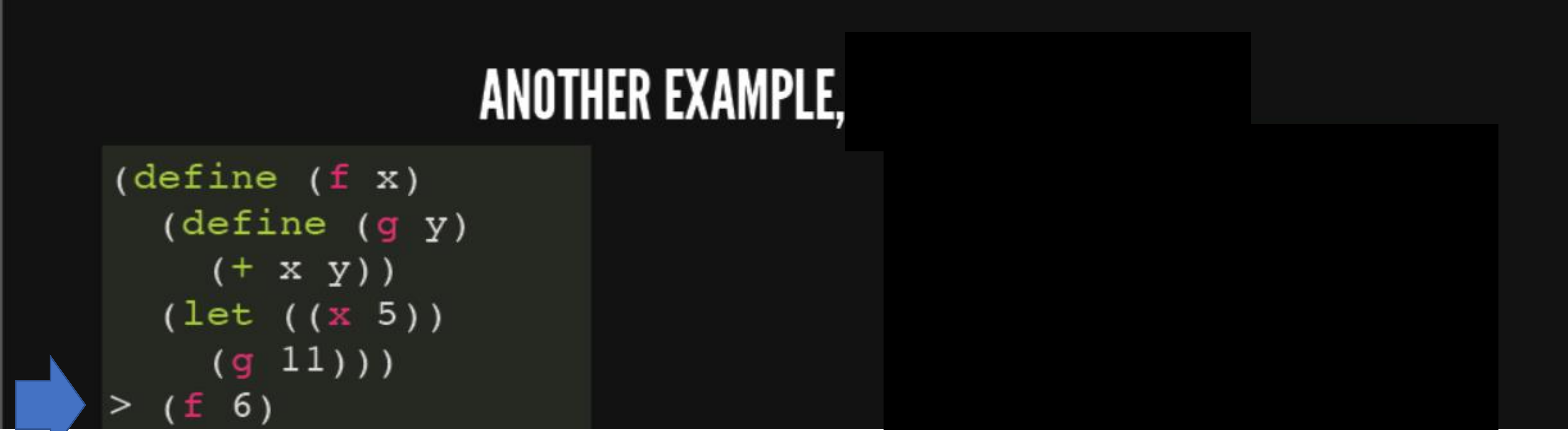
OH NO!
SUBSTITUTION SEMANTICS IS WRONG



```
> (define a 10)
> (define (f x) (+ x a))
> (f 100)
110
> (let ((a 20))
    (f 100))
```

Substitution here would have given…

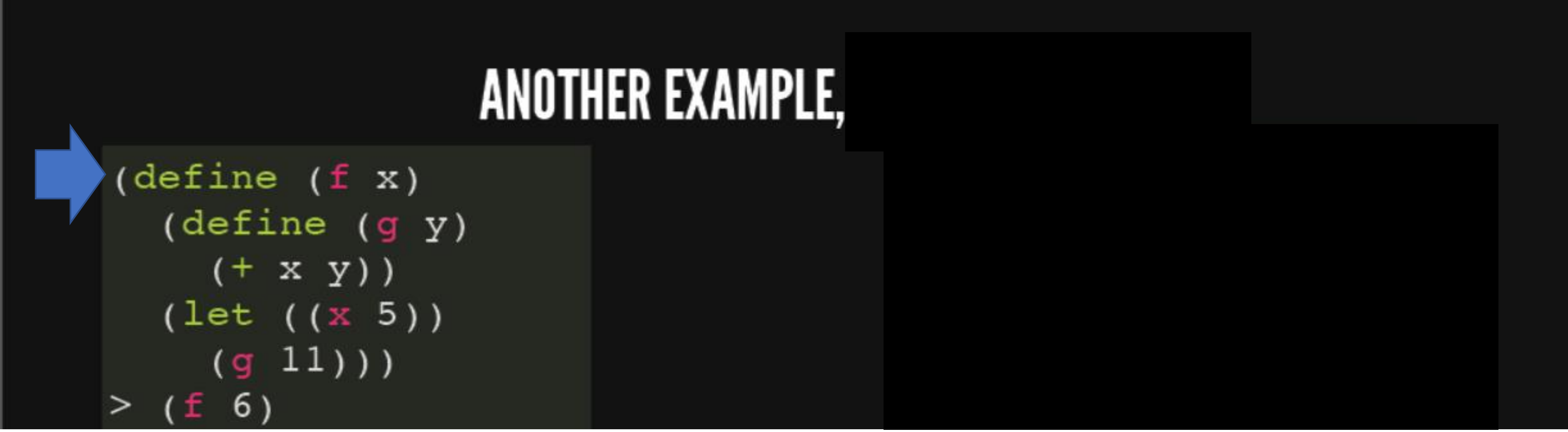
120



ANOTHER EXAMPLE, [redacted]

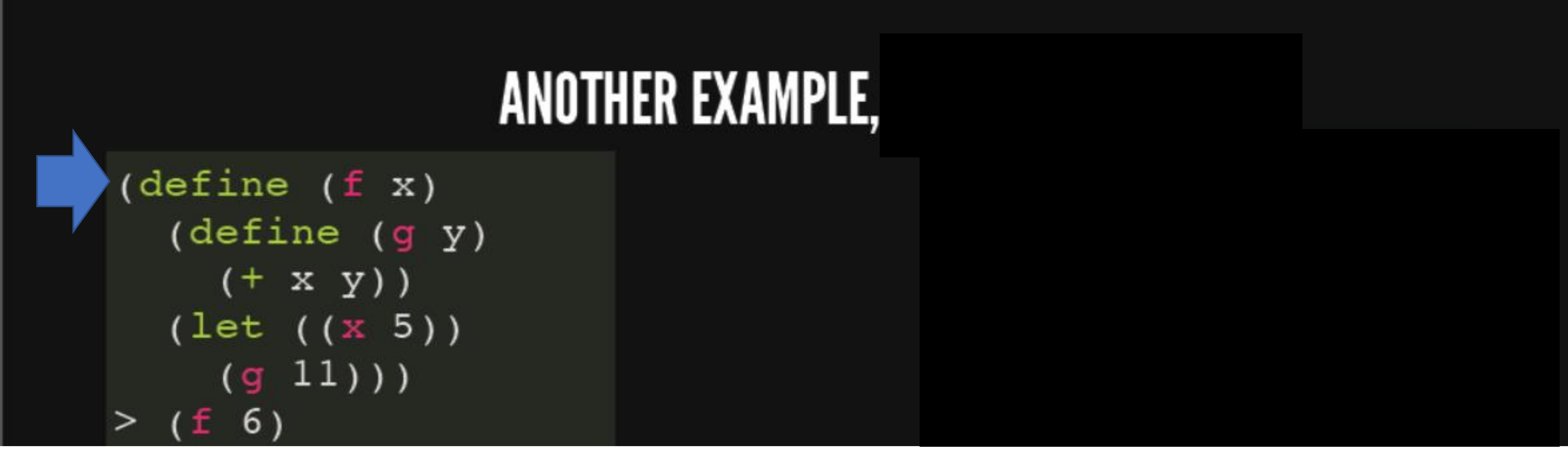
```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

GLOBAL Environment



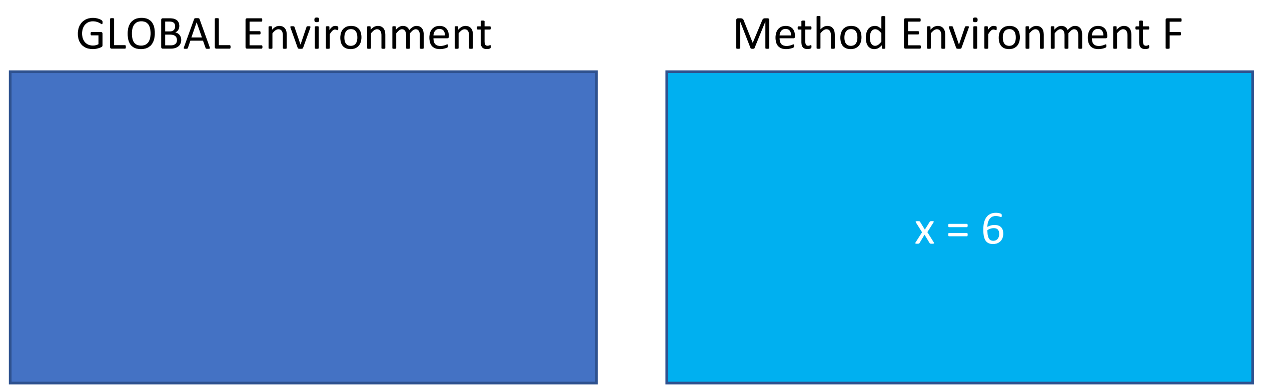
ANOTHER EXAMPLE, [redacted]

```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

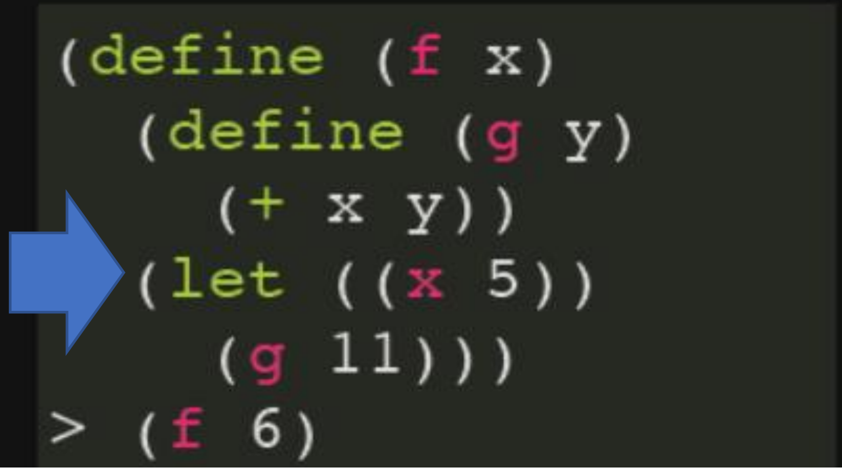


ANOTHER EXAMPLE, [redacted]

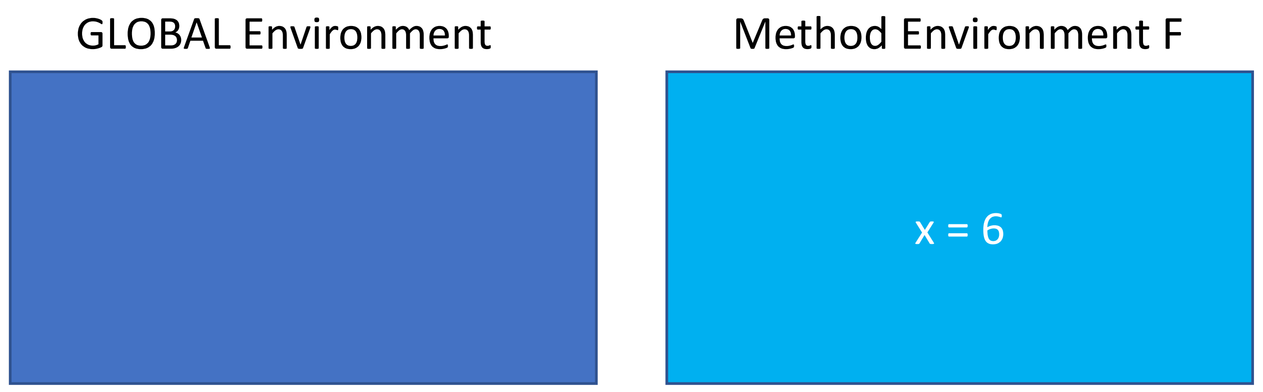
```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

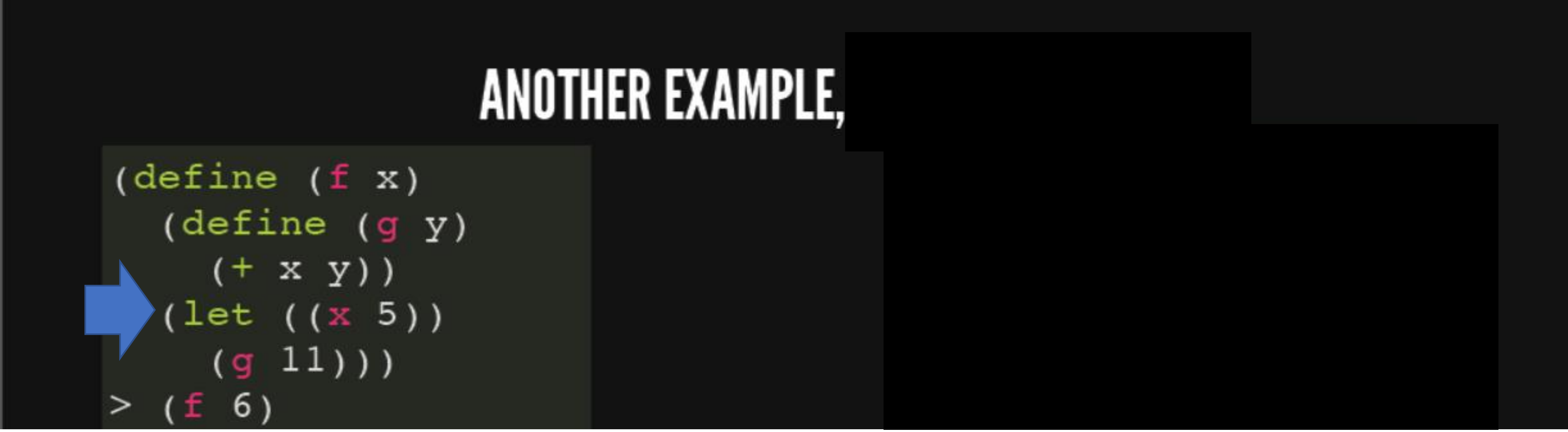


ANOTHER EXAMPLE, [redacted]



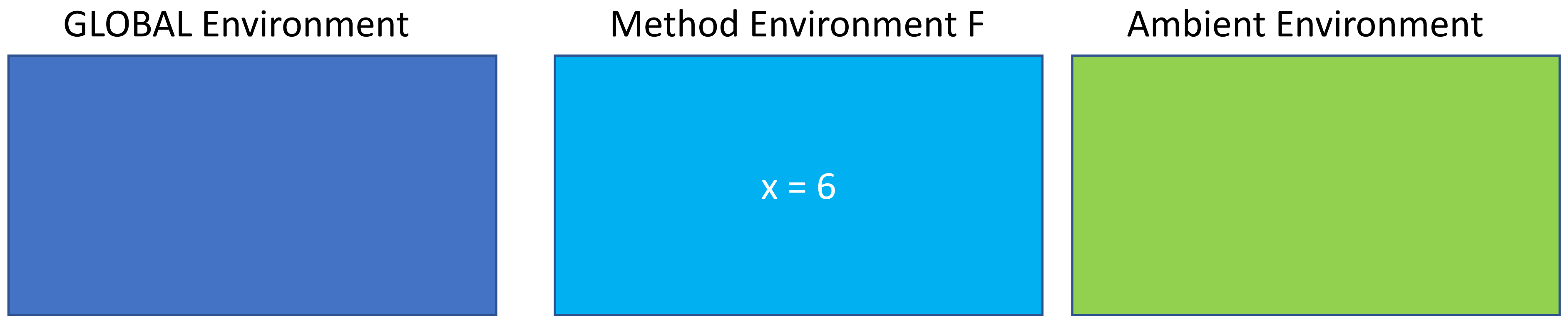
```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```



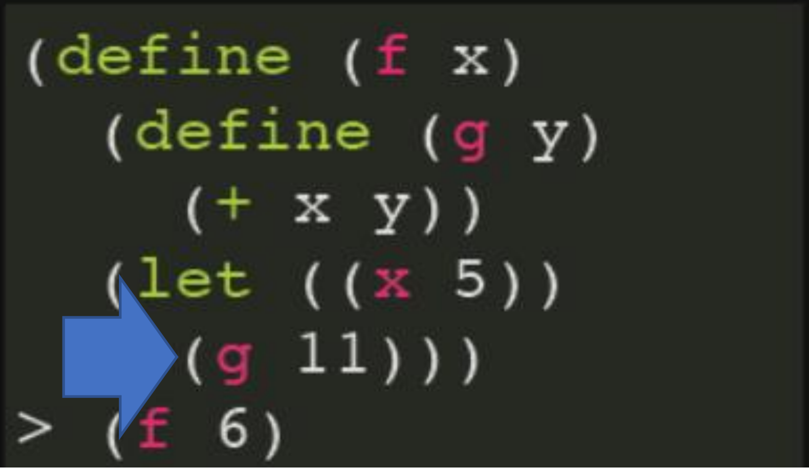


ANOTHER EXAMPLE, [redacted]

```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```



ANOTHER EXAMPLE, [redacted]



```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

GLOBAL Environment

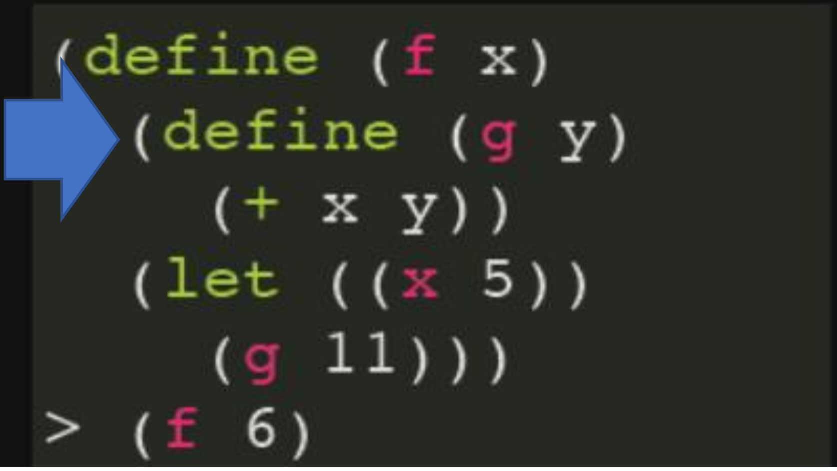
Method Environment

x = 6

Ambient Environment

x = 5

ANOTHER EXAMPLE, [redacted]



```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

GLOBAL Environment

Method Environment F

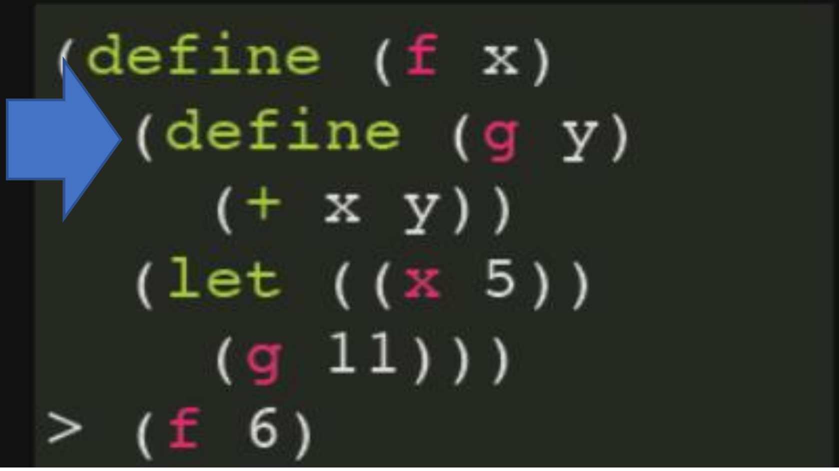
x = 6

Ambient Environment

x = 5

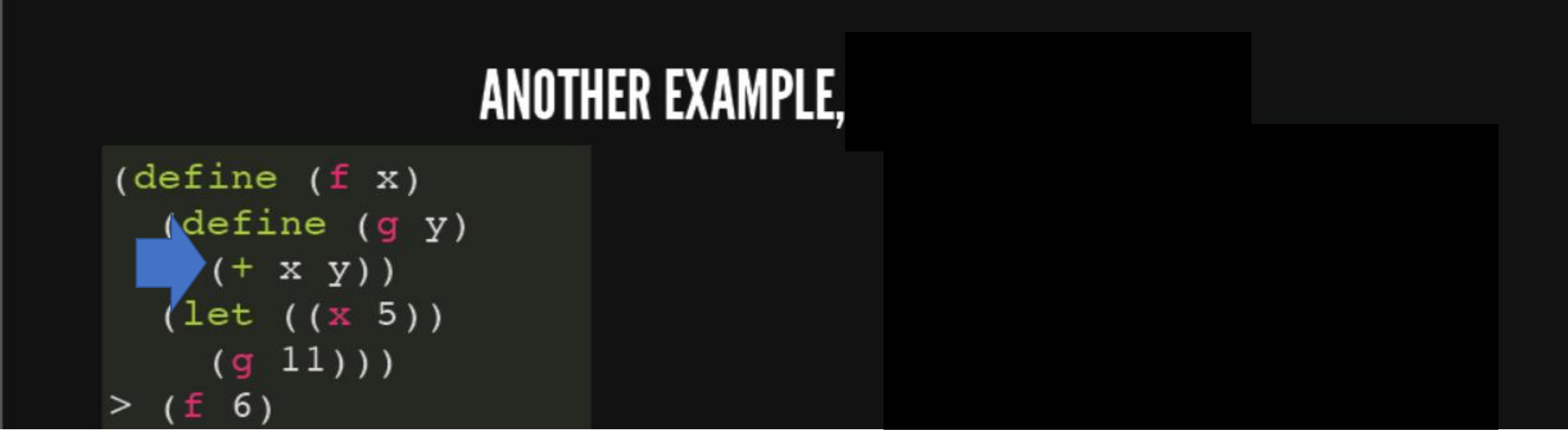
Method Environment G

ANOTHER EXAMPLE, [redacted]



```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

GLOBAL Environment	Method Environment F	Ambient Environment	Method Environment G
	x = 6	x = 5	y = 11

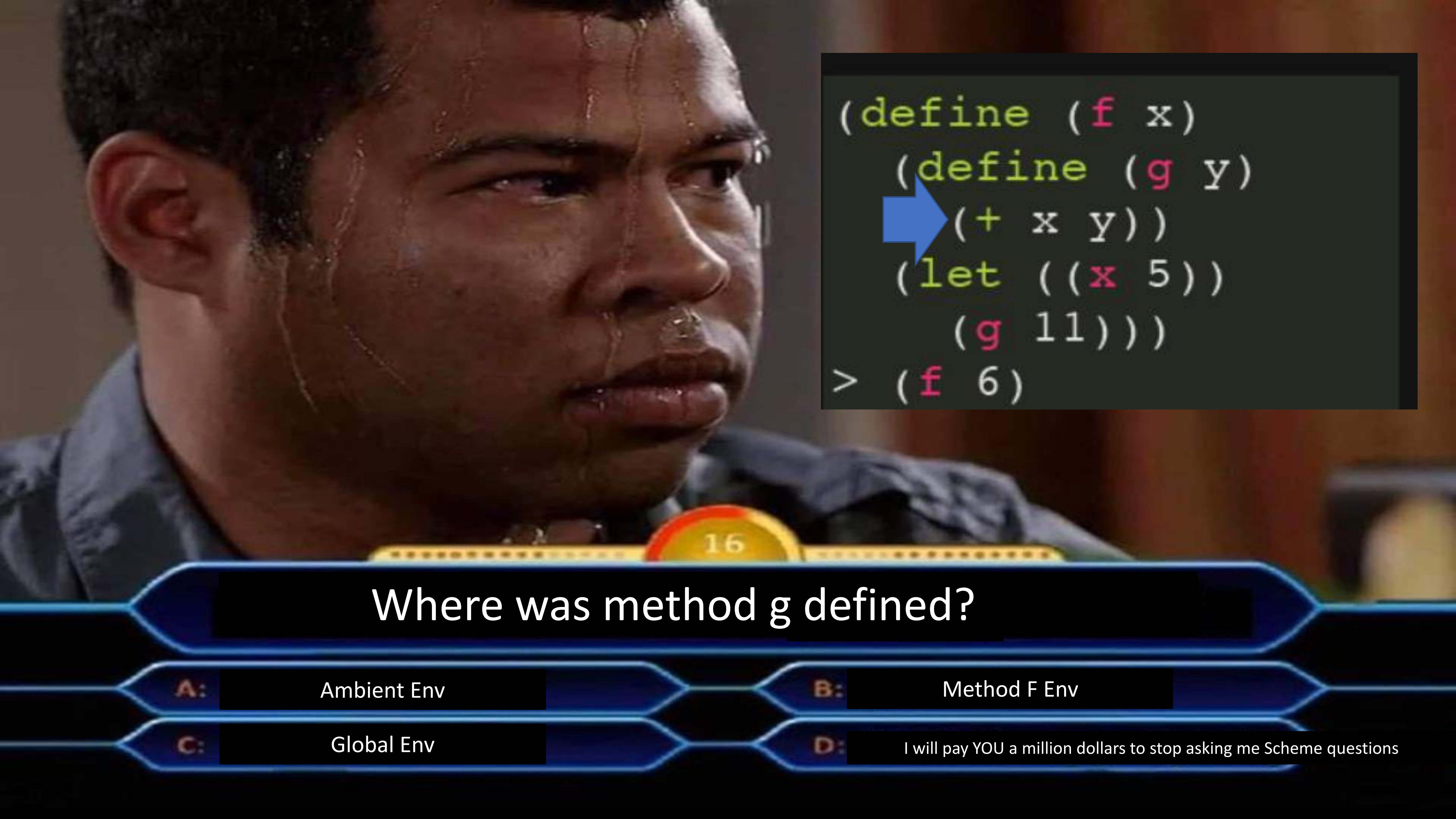


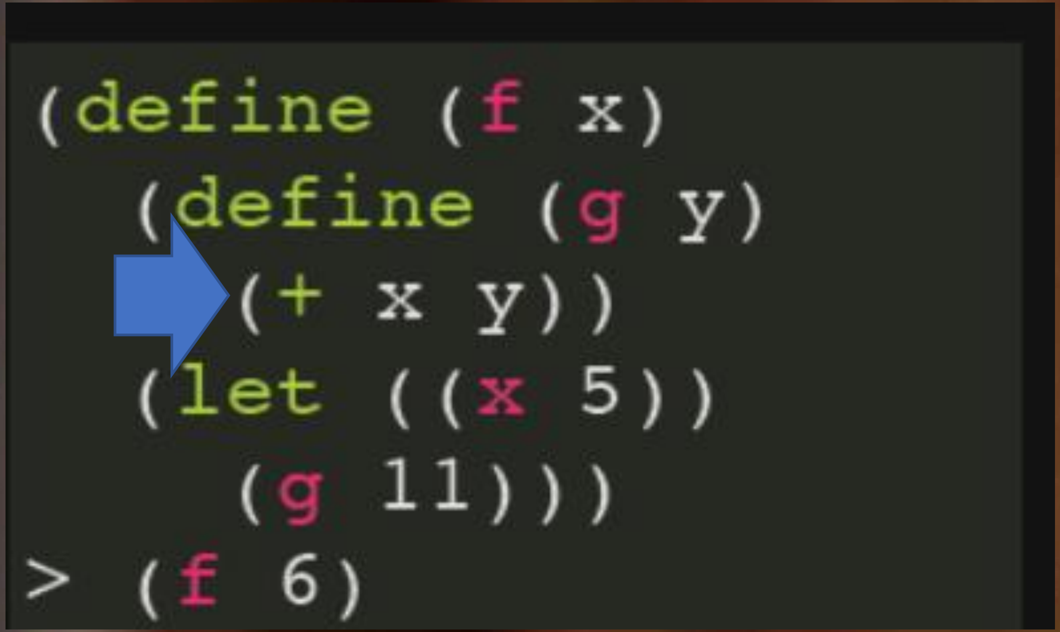
ANOTHER EXAMPLE, [redacted]

```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

GLOBAL Environment	Method Environment F	Ambient Environment	Method Environment G
	x = 6	x = 5	y = 11

Now time for the million dollar question…





```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

Where was method g defined?

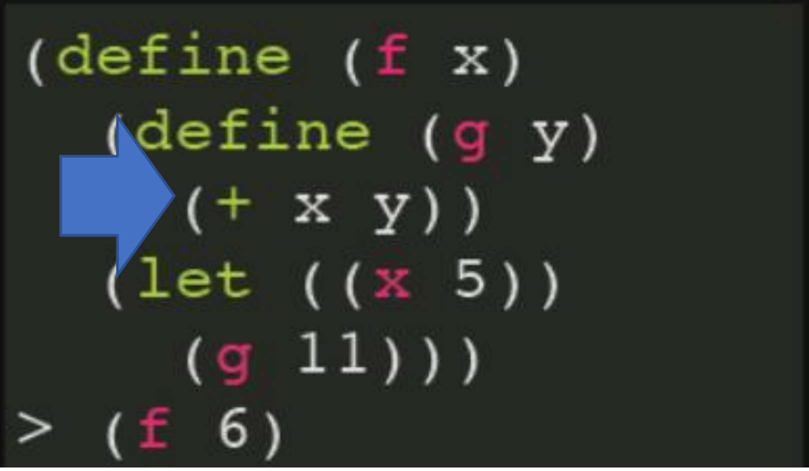
A: Ambient Env

B: Method F Env

C: Global Env

D: I will pay YOU a million dollars to stop asking me Scheme questions

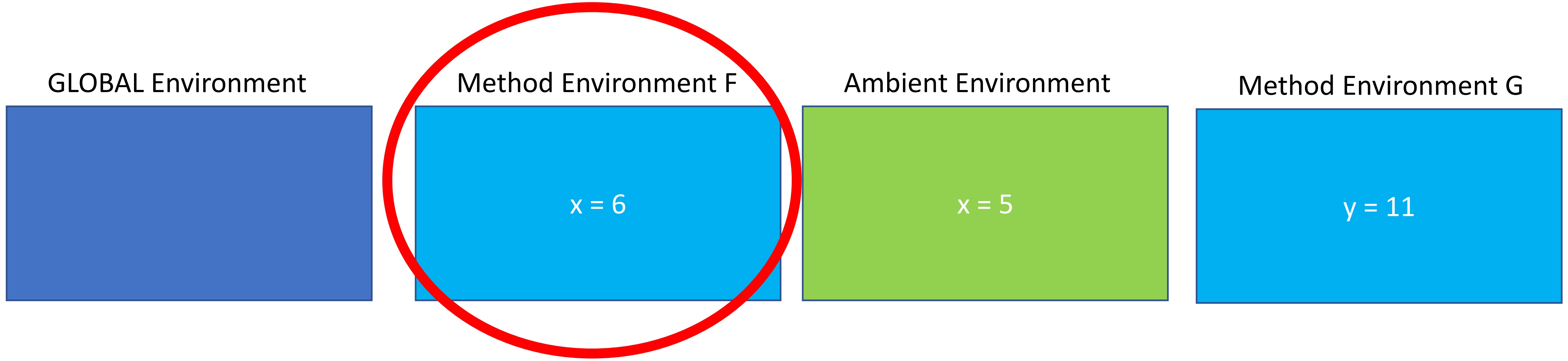
ANOTHER EXAMPLE,



```
(define (f x)
  (define (g y)
    (+ x y))
  (let ((x 5))
    (g 11)))
> (f 6)
```

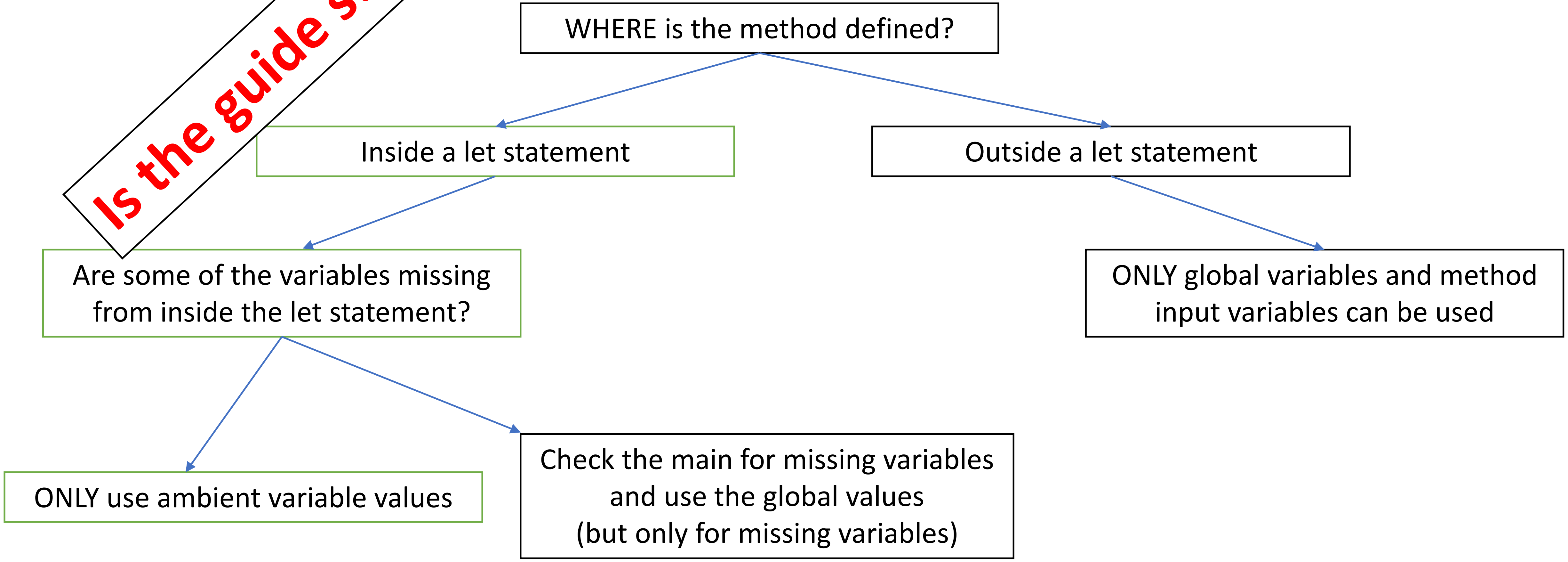
Solution:

17

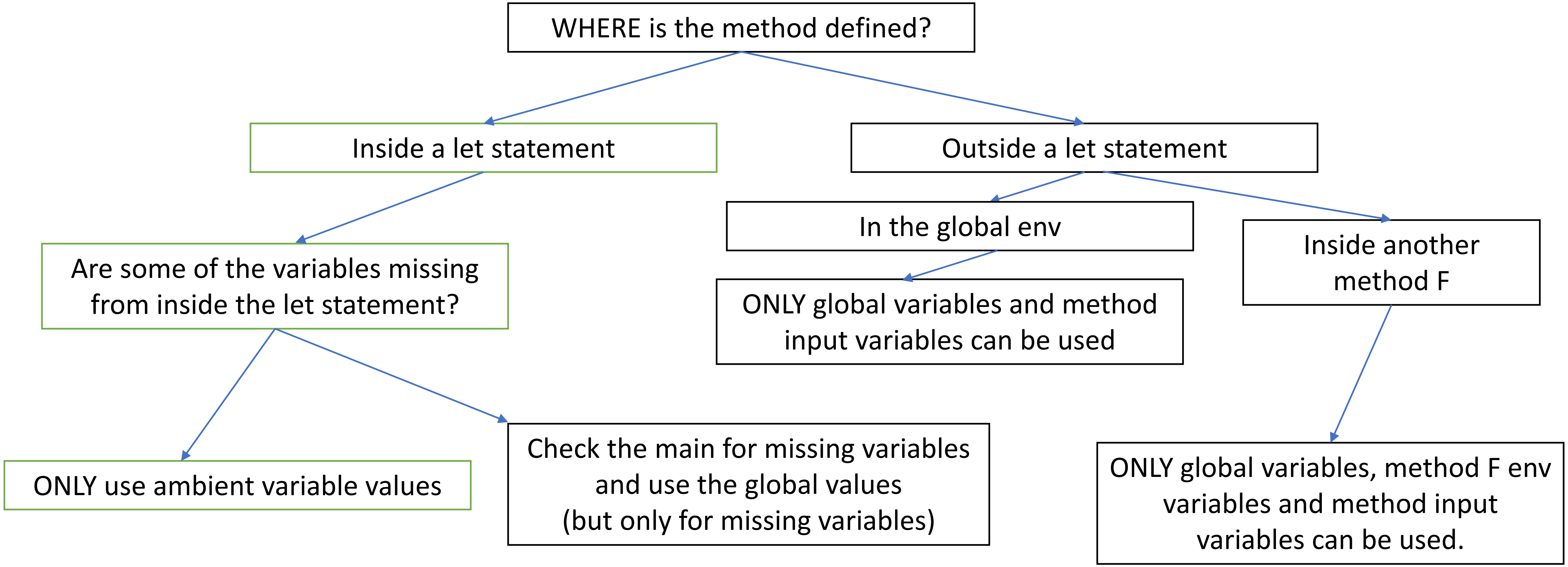


A Guide to figu[illegible]out Method and Environment Variables

Is the guide still right???


WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)

The Guide CORRECTED


WHERE is the method defined?
Inside a let statement
Outside a let statement
Are some of the variables missing from inside the let statement?
In the global env
Inside another method F
ONLY global variables and method input variables can be used
ONLY use ambient variable values
Check the main for missing variables and use the global values (but only for missing variables)
ONLY global variables, method F env variables and method input variables can be used.

ENVIRONMENT CLUTTER AND LOCAL FUNCTIONS

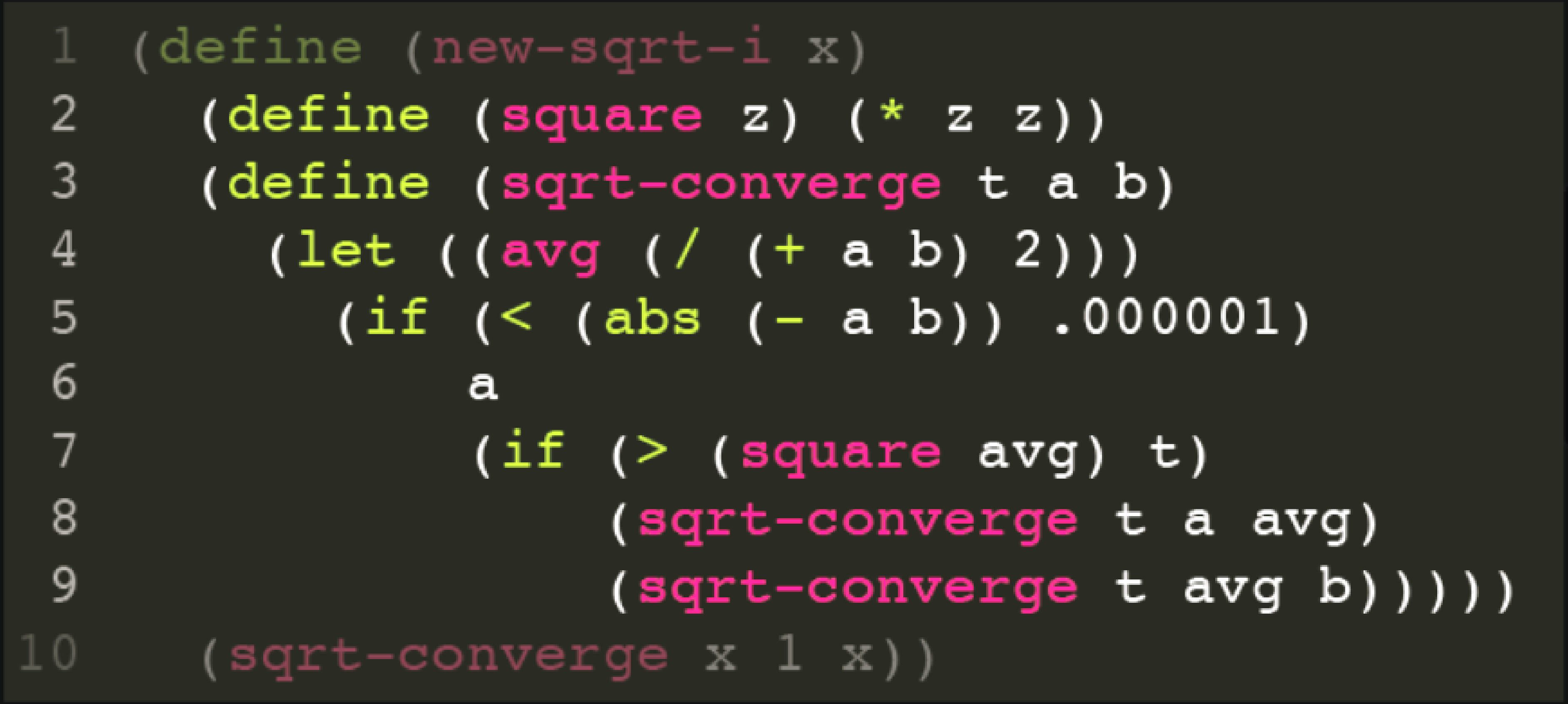
- Consider the definition

```
(define (square a)    (* a a))
(define (sqrt-converge x a b)
  (let ((avg (/ (+ a b) 2)))
    (if (< (abs (- a b)) .000001)
        a
        (if (> (square avg) x)
            (sqrt-converge x a avg)
            (sqrt-converge x avg b))))))
(define (new-sqrt x) (sqrt-converge x 1 x))
```

- We wished to define `new-sqrt`, but introduced many other functions into the environment.
- What if someone clobbers them or, in general, they clash with other functions?

ENVIRONMENTAL PROTECTION…LEAVE BEHIND ONLY WHAT YOU INTENDED

- Making internal structure (e.g., `sqrt-converge`) available to the user is dirty, provides opportunities for error.
- To avoid this, we can place the definitions inside new-sqrt:



```
(define (new-sqrt-i x)
  (define (square z) (* z z))
  (define (sqrt-converge t a b)
    (let ((avg (/ (+ a b) 2)))
      (if (< (abs (- a b)) .000001)
          a
          (if (> (square avg) t)
              (sqrt-converge t a avg)
              (sqrt-converge t avg b)))))
  (sqrt-converge x 1 x))
```

Figure Sources

- https://memegenerator.net/img/instances/69835463.jpg
- https://i.kym-cdn.com/entries/icons/original/000/023/397/C-658VsXoAo3ovC.jpg
- https://preview.redd.it/qdfmlzjx45e41.png?auto=webp&s=00b3e324fc7088664502b0935d433de6ddf7c5ea
- https://imgix.bustle.com/rehost/2016/9/13/809bfbc3-bc7a-41b7-b68c-fa126a8896c1.jpg?w=800&fit=crop&crop=faces&auto=format%2Ccompress
- https://i0.wp.com/trendingposts.net/wp-content/uploads/2018/10/Confused-man.jpg?resize=650%2C451&ssl=1