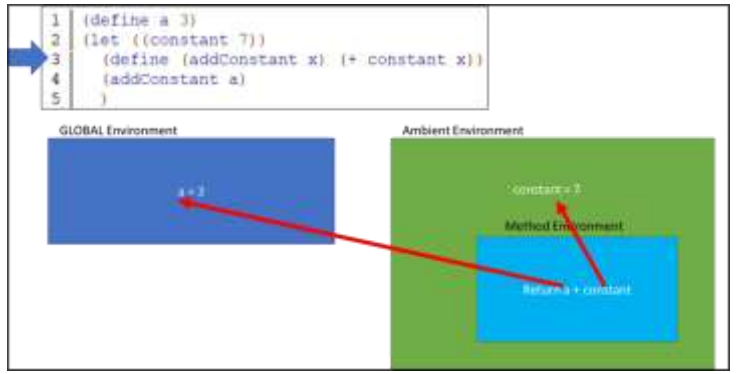
CSE 1729:Principles of Programming

Lecture 6: Let There Be Scheme



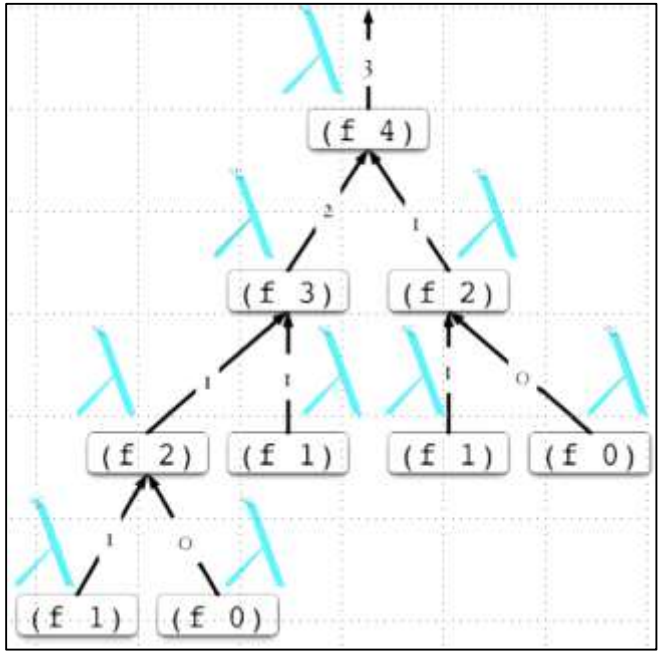
Kaleel Mahmood

Department of Computer Science and Engineering

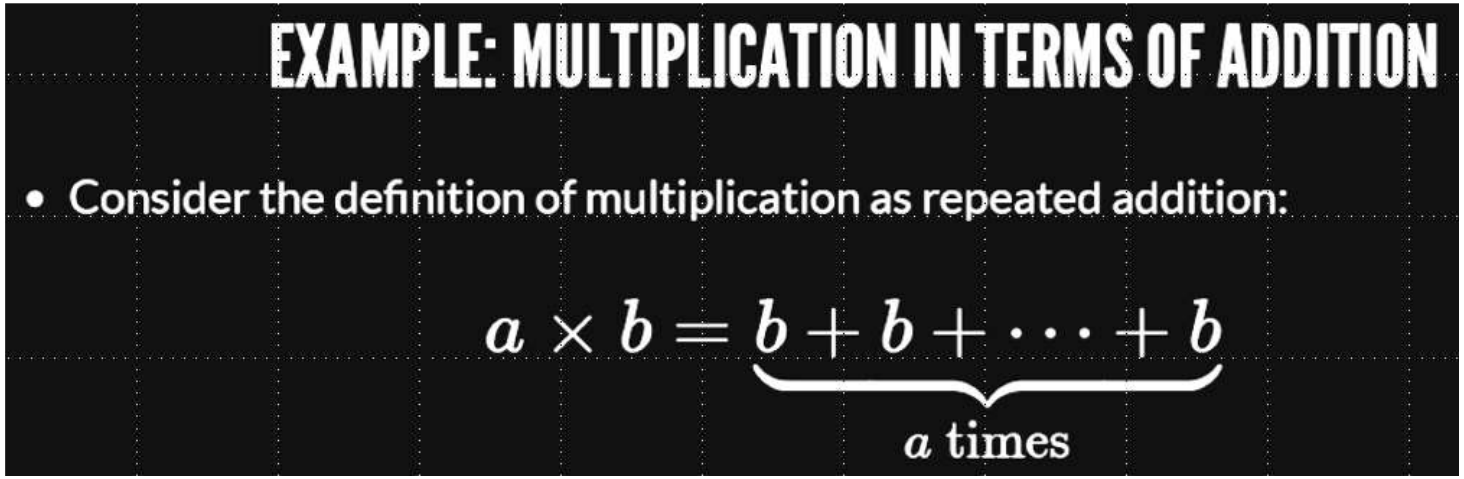
University of Connecticut

Previously in CSE 1729… *So many examples of recursion!!!!!*

1. Fibonacci Function



2. Multiplication through addition



2. Guessing a number using binary search

COMPUTING SQUARE ROOTS BY AVERAGING

- One simple way to compute an approximation to the square root of a number x is to
  - Start with two guesses, a and b, with the property that

$$a < \sqrt{x} < b$$

- (For example, if $x > 1$, we could start with $a = 1, b = x$.) Thus we know that the actual square root is between a and b.
- If $\frac{(a+b)}{2}$ is larger than the square root (which we can check by comparing $[\frac{(a+b)}{2}]^2$ with x) we know the real square root lies between a and $\frac{(a+b)}{2}$.
- Otherwise, the real square root lies between $\frac{(a+b)}{2}$ and b.

IN SCHEME

```
(define (average a b) (/ (+ a b) 2))
(define (square a) (* a a))

(define (sqrt-converge x a b)
  (if (< (abs (- a b)) .000001)
      a
      (if (> (square (average a b)) x)
          (sqrt-converge x a (average a b))
          (sqrt-converge x (average a b) b))))
```

Now, we might like to define a more attractive square root function that does not require choosing a and b:

```
(define (new-sqrt x) (sqrt-converge x 1 x))
```

LOCAL VARIABLES

- (average a b) is referred to several times in sqrt-converge.

Wouldn't it be nice if we could temporarily bind a "local" variable to this value?

- The let construct does exactly this:
- **Semantics**:
  - Evaluate each <expr$_i$>, yielding a value v_i.
  - Create a new environment by starting with the current one and binding each x_i to v_i.
  - Then return the value of <body-expr> in this environment.

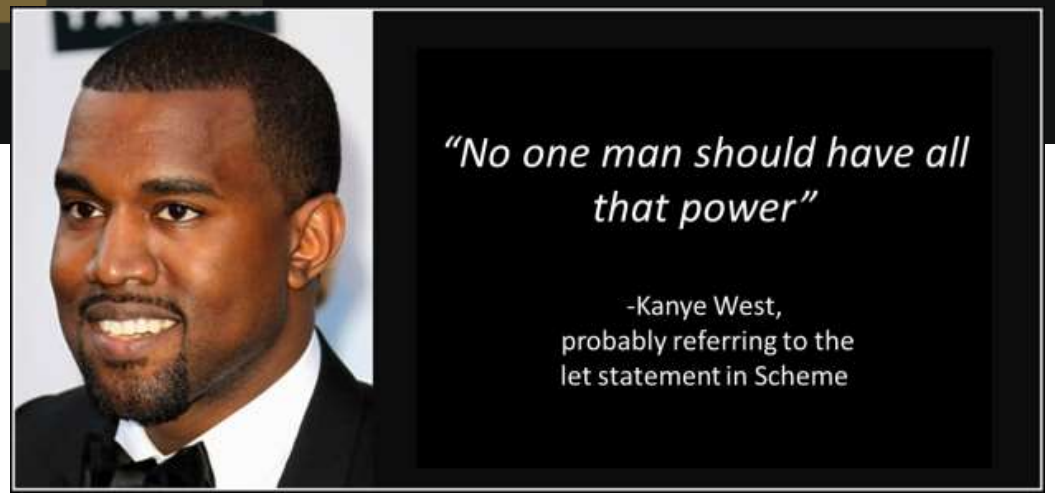
```
(let ((x1 <expr1>)
      (x2 <expr2>)
...
      (xk <exprk>))
<body-expr>)
```

Now I know what everyone is thinking.
This new “let” statement is so much cooler than “define”.

LOCAL VARIABLES

```
(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)))))
```

The let statement binds avg to $\frac{(a+b)}{2}$ for the shaded block of code



How about we try using the let statement outside of a method call?

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

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

(addConstant a)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
8
>
```

Can we rewrite this statement using "let"?

Rewriting the “add constant” method with the let statement.

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

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

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

What should the output of this code be?

10 right?

Rewriting the “add constant” method with the let statement.

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

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

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

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
8
>
```

We need to understand the rules of scoping in Scheme

- The “let” statement behaves differently depending on WHERE in the code it is called.
- We will now trace through that example pictorial to understand.

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

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

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

```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```

GLOBAL Environment

a = 3

```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```

GLOBAL Environment

a = 3
constant = 5

```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```

GLOBAL Environment

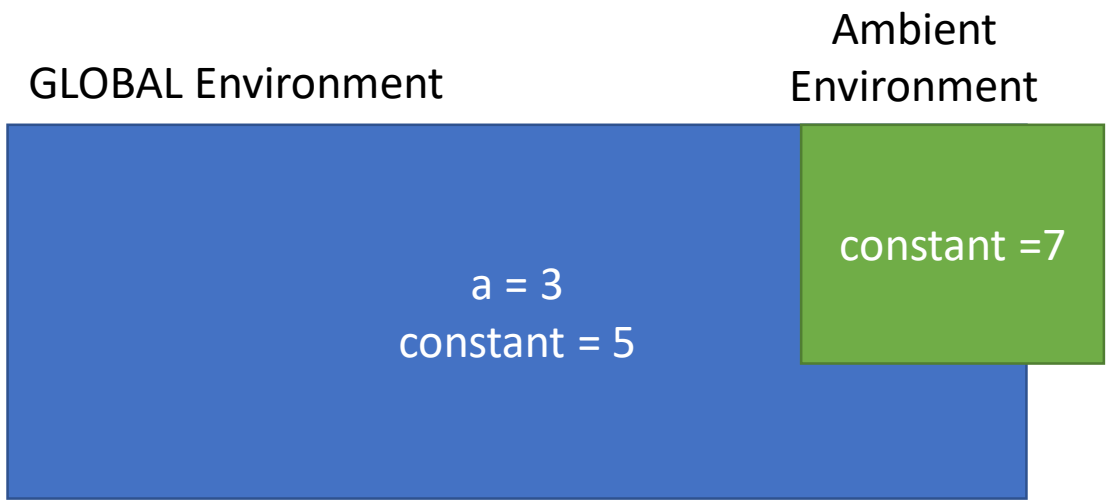
a = 3
constant = 5

```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```

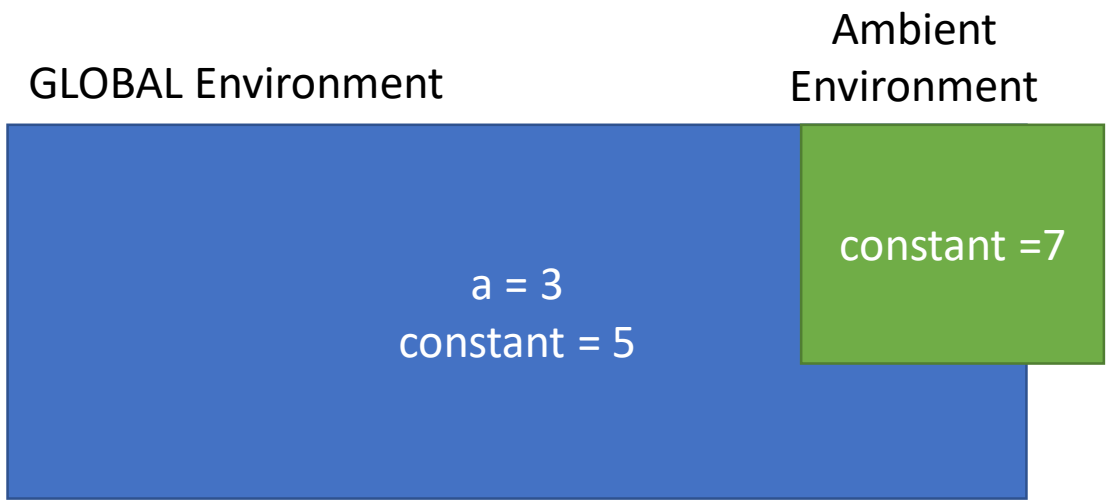
GLOBAL Environment

a = 3
constant = 5

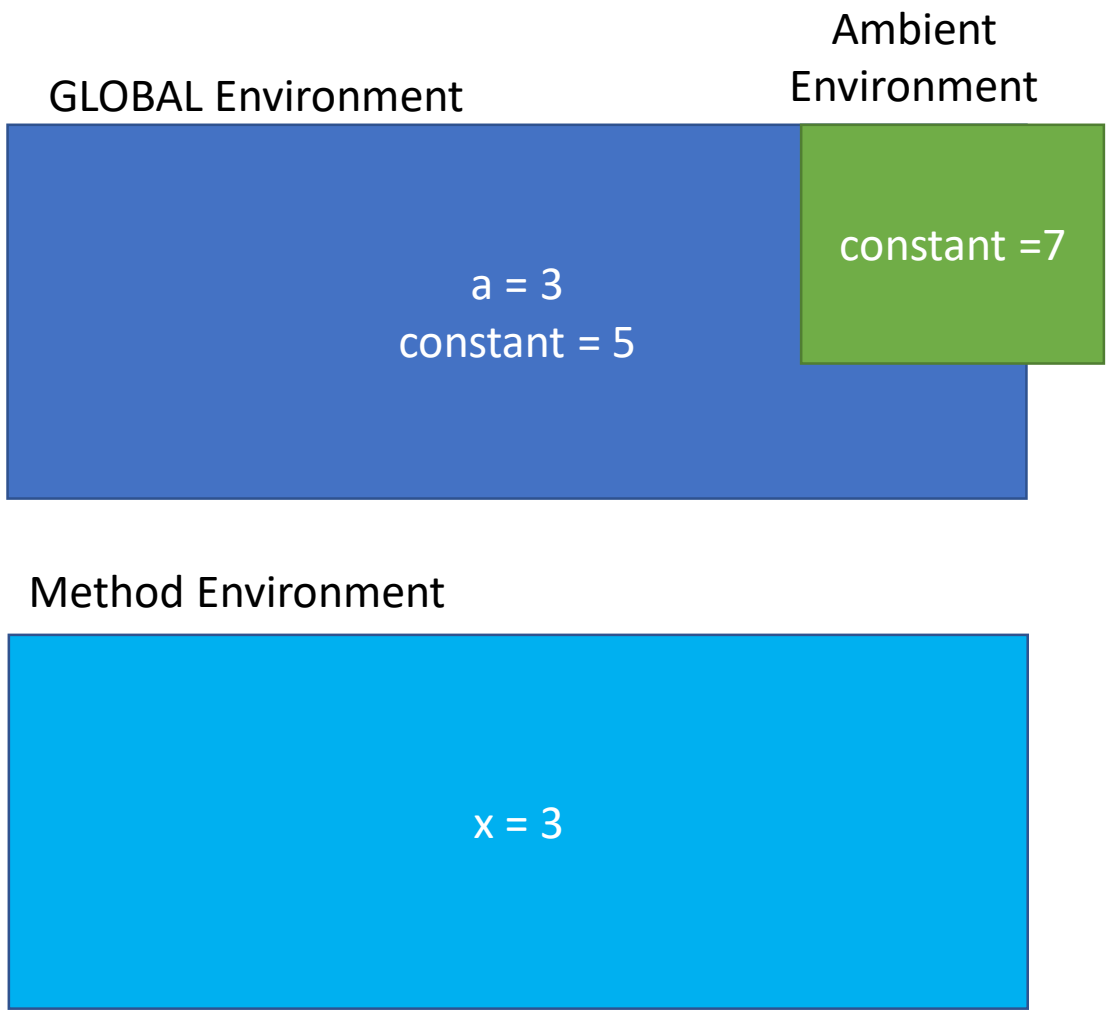
```
(define a 3)
(define constant 5)

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

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



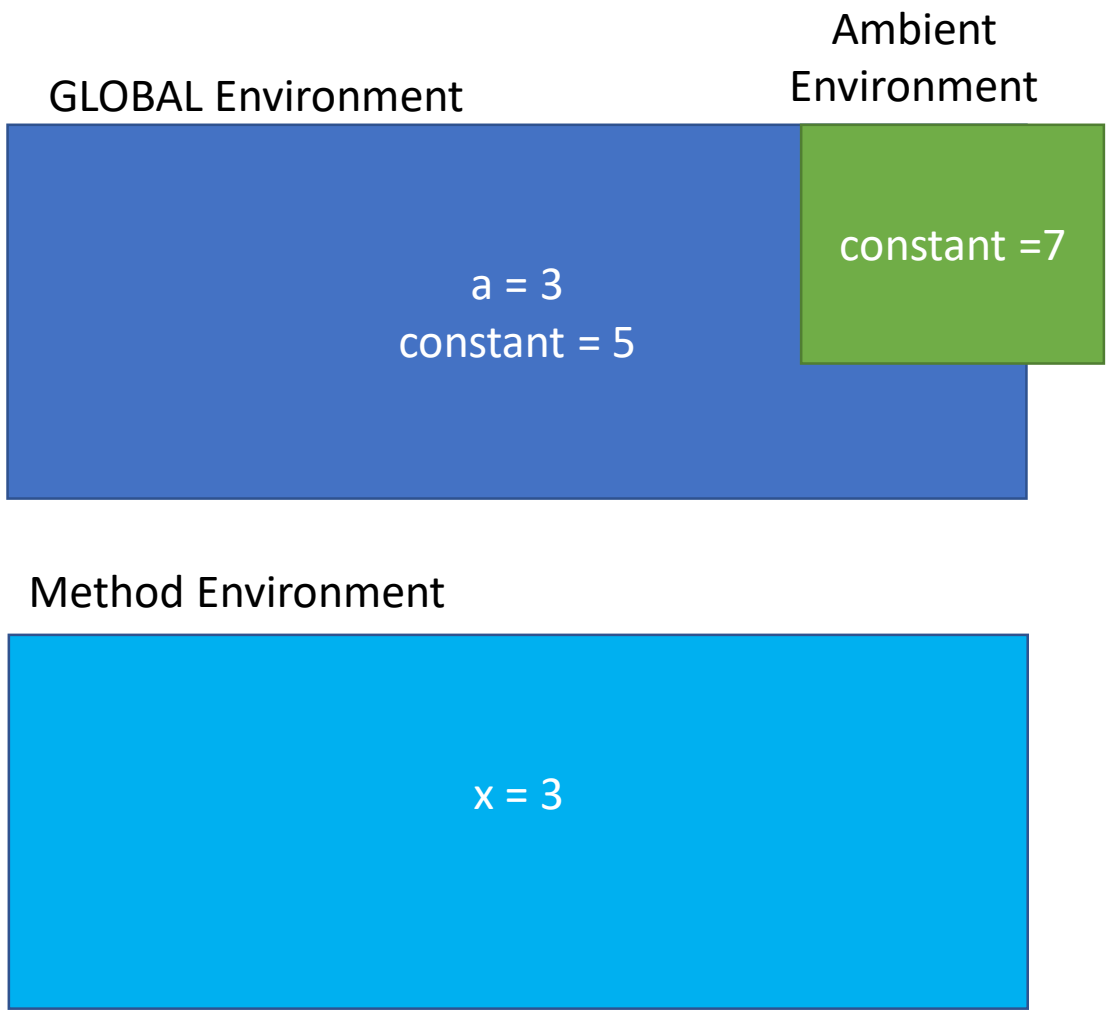
```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```



```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```

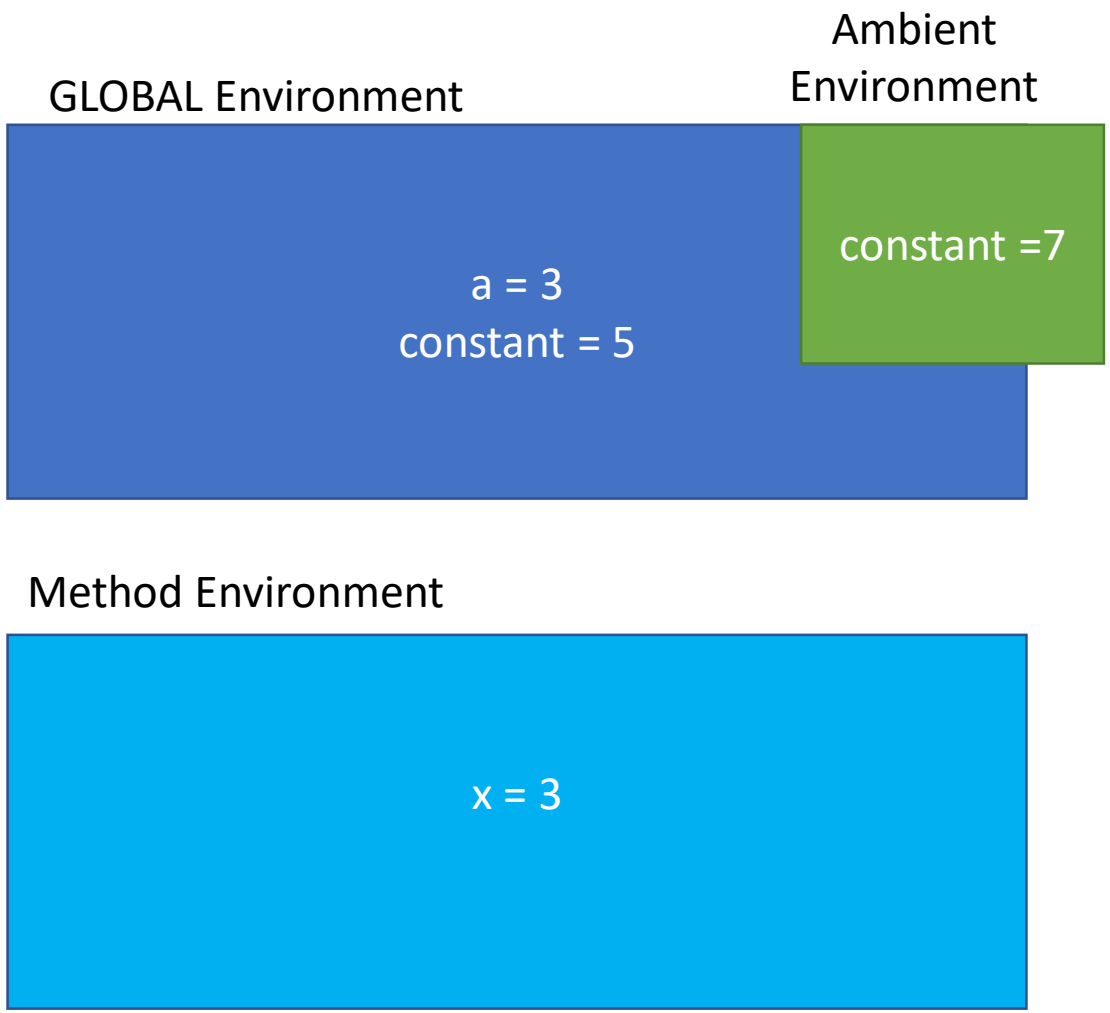


```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```



Where is constant?

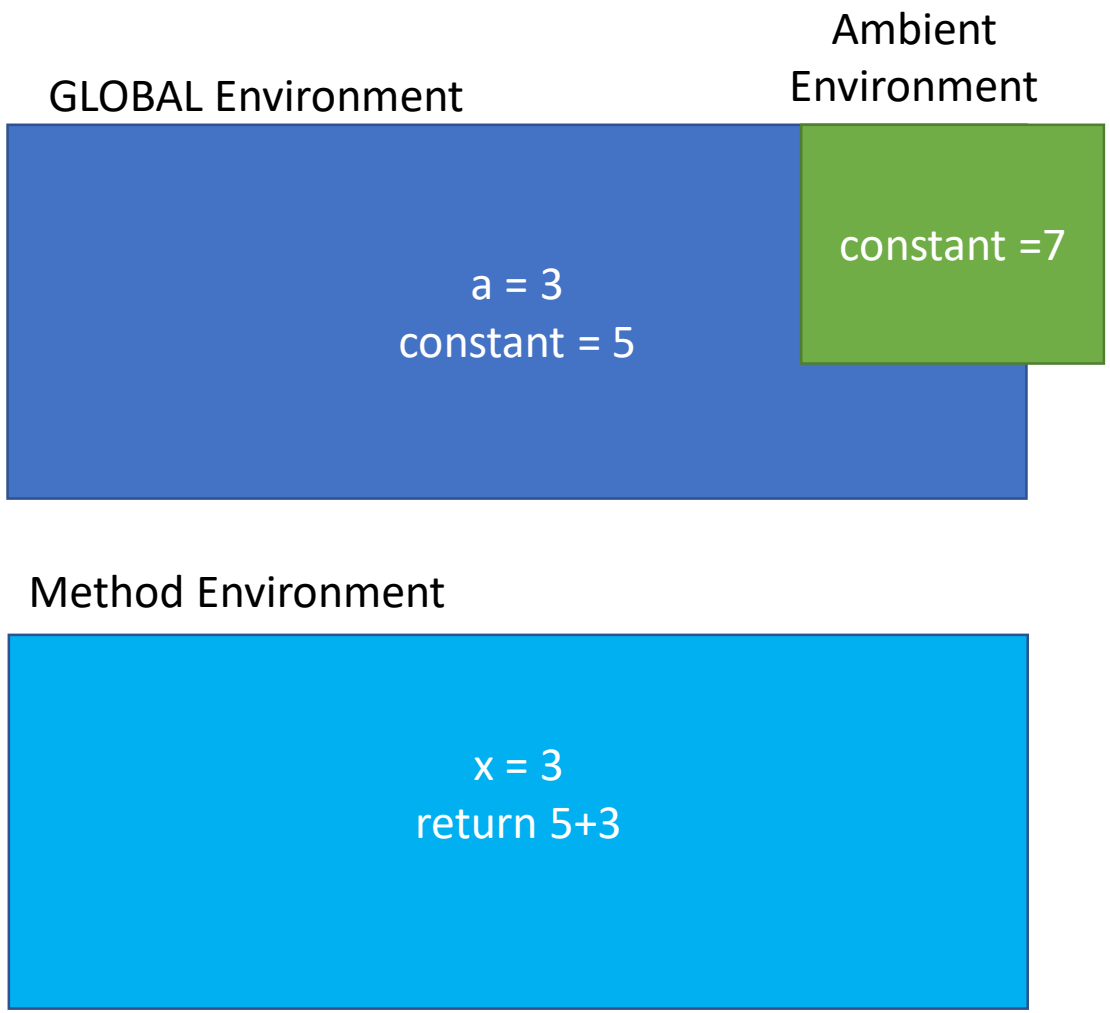
```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```



Where is constant?

First look in the method environment (missing)

```
1  (define a 3)
2  (define constant 5)
3
4  (define (addConstant x)
5    (+ constant x)
6  )
7
8  (let ((constant 7))
9  (addConstant a))
```

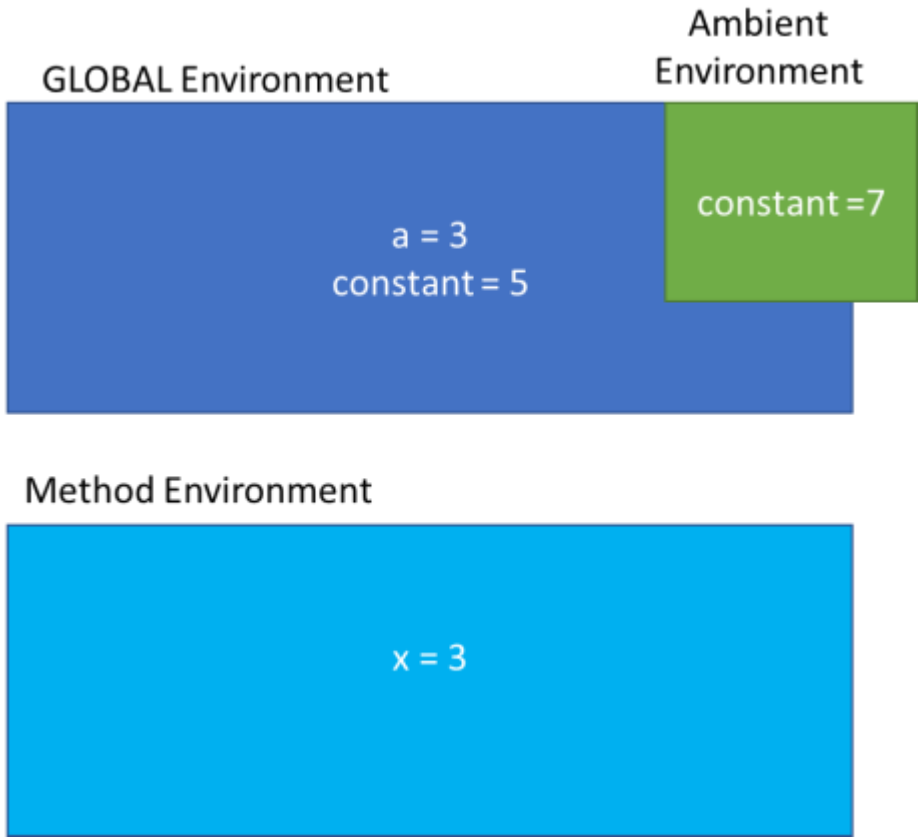


Where is constant?

Next look in the GLOBAL environment

Question: What happens if we don't make constant a global variable?

- Does that mean the interpreter will look in the "ambient environment" next?



Question: What happens if we don't make constant a global variable?

- Does that mean the interpreter will look in the "ambient" environment next?

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

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

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

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

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

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

Question: What happens if we don't make constant a global variable?

- Does that mean the interpreter will look in the "ambient" environment next?

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

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

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

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
constant: undefined;
 cannot reference an identifier before its definition
>
```

Answer: Nope!

How can ambient variables defined OUTSIDE of methods be used INSIDE of methods?

1. By passing them as METHOD PARAMETERS:

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

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

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

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

OR what if we put the method definition inside the ambient environment?

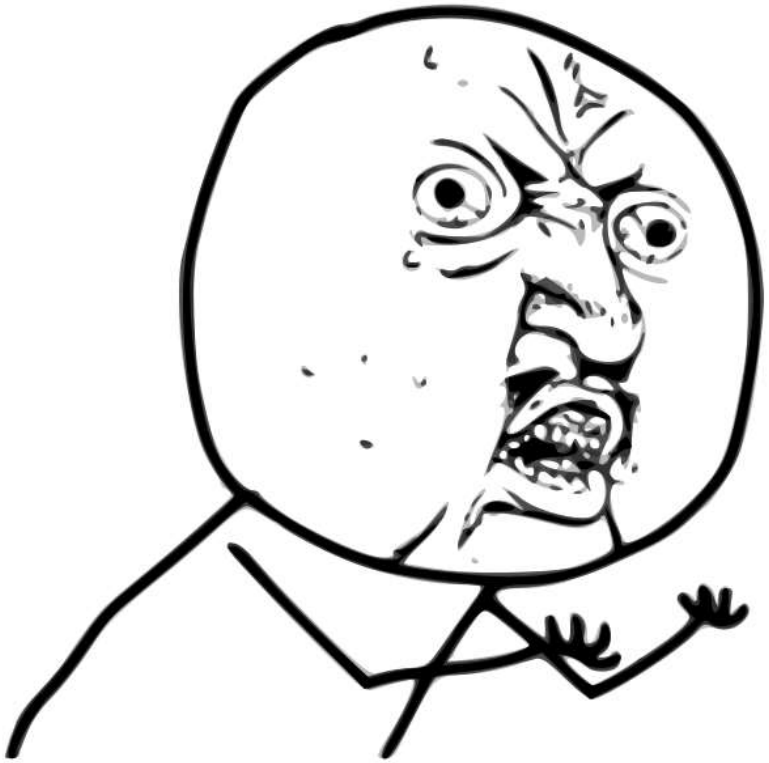
```
(define a 3)
;(define constant 5)

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

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

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

Result:

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

Why did that work?



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

Let’s draw a picture to understand…

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

GLOBAL Environment

a = 3

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

GLOBAL Environment

a = 3

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

GLOBAL Environment

a = 3

Ambient Environment

constant = 7

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

GLOBAL Environment

a = 3

Ambient Environment

constant = 7

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

GLOBAL Environment

a = 3

Ambient Environment

constant = 7

Method Environment

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

GLOBAL Environment

a = 3

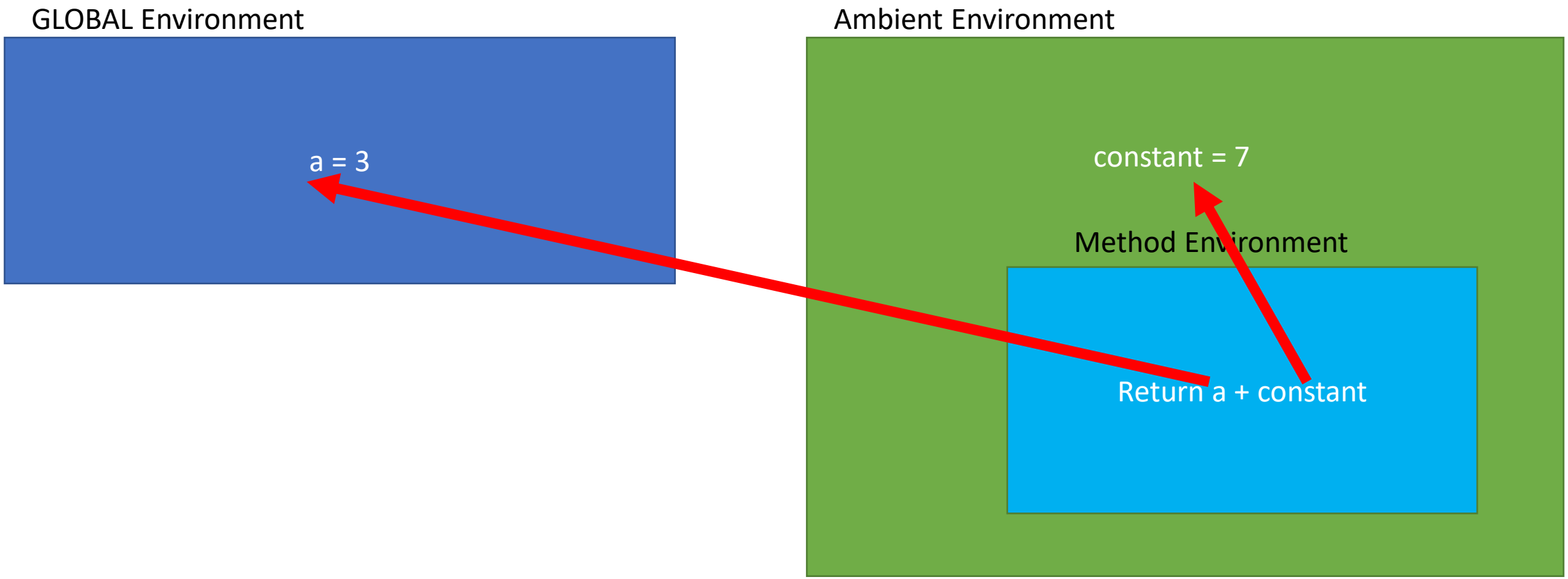
Ambient Environment

constant = 7

Method Environment

Return a + constant

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



Conclusion 1: How can methods access ambient variables?

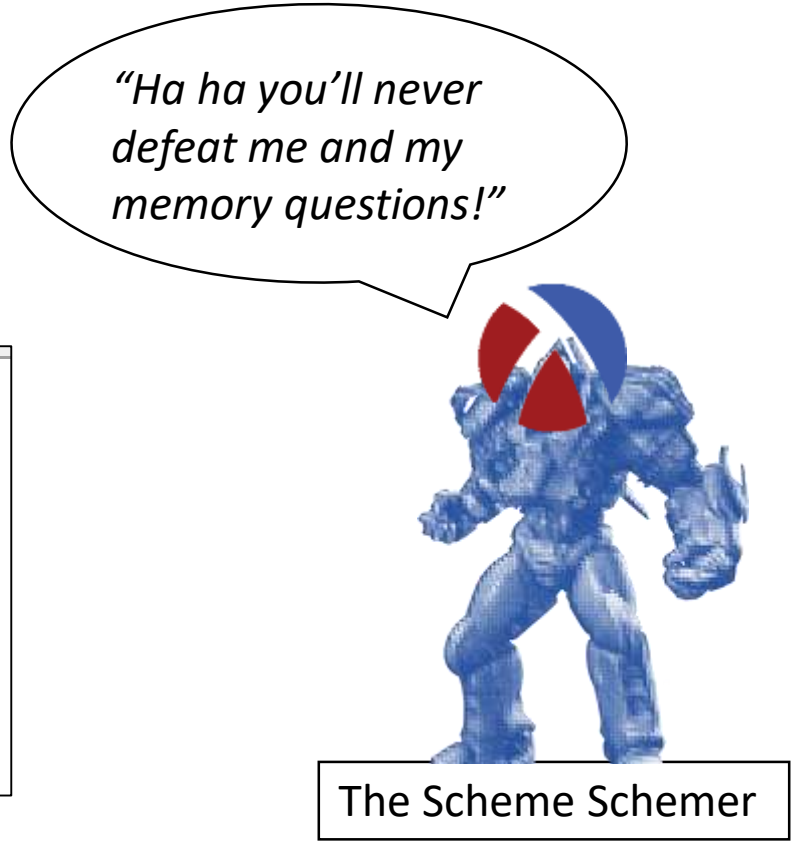
1. Pass the ambient variables as method input parameters.
2. Put the method declaration INSIDE of the ambient environment.

Other important points:

- The “let” statement creates an ambient environment.
- Methods can always access variables from the global environment.

One last twist…

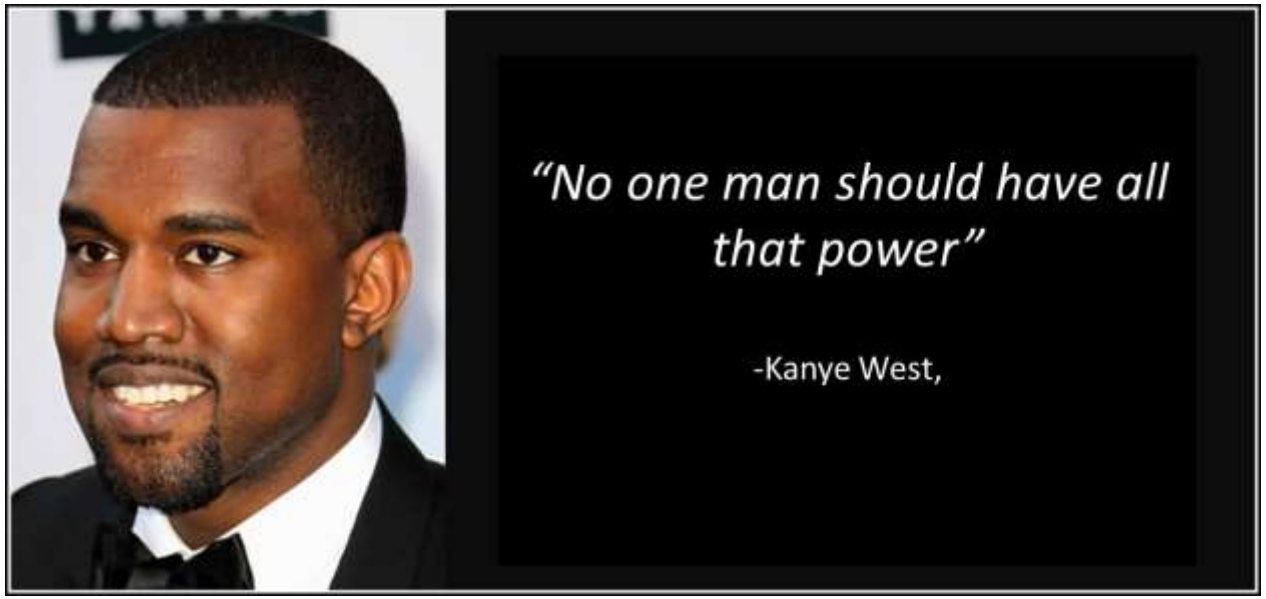
```
(define a 0)
(define constant 0)
(let ((constant 7)
  (a 3))
  (define (addConstant x) (+ constant x))
  (addConstant a)
  )
```



What should the output of this code be?

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

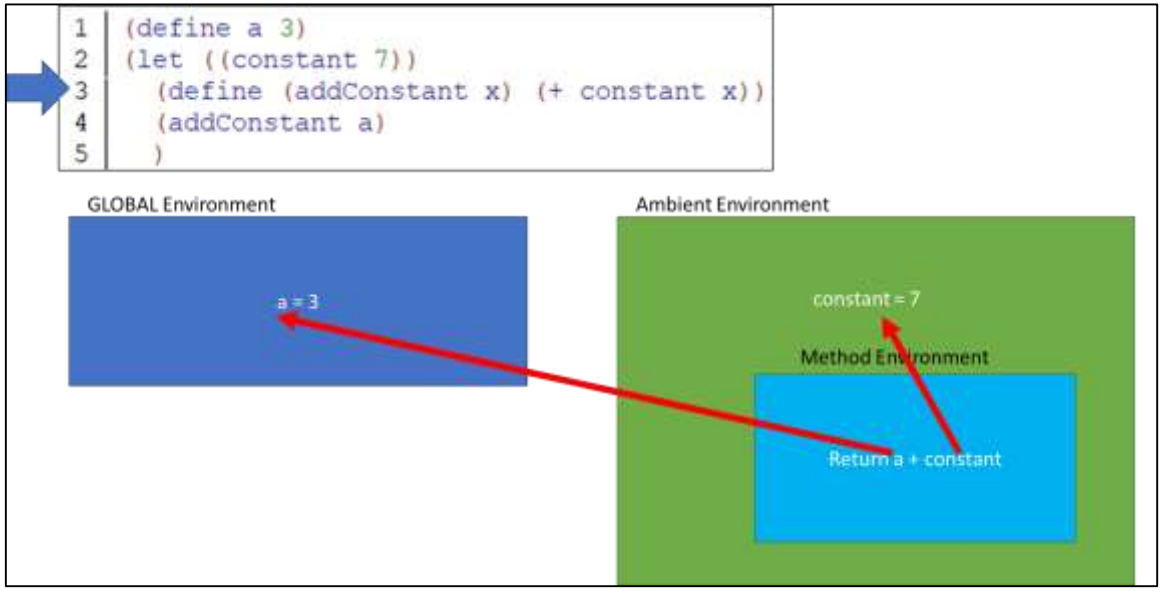
Conclusion 2: Why did Kanye West say this quote?



- Two possible answers:

1. He was rapping.
2. He was warning you to be very careful when using the let statement and variable scoping in Scheme.

Conclusion 3: *WHERE* Methods are defined MATTERS



- If the method is defined (notice here we say defined, not called) inside a let statement, it will use the ambient environment variables.
- If a method is called in the global environment it cannot use ambient variables unless they are passed as parameters to the method.

Figure Sources

- https://external-preview.redd.it/fudpTkeL0ltbeklZ6lRujiTeIPlye0823LKwAMkuyYY.jpg?auto=webp&s=4ac03fb1cc98a547d9b578938442686c117f0ca1
- https://uploads.dailydot.com/04c/00/a72e177e45dc0788.jpg?auto=compress%2Cformat&fit=scale&h=300&ixlib=php-3.3.0&w=600&wpsize=medium
- https://www.azquotes.com/picture-quotes/quote-no-one-man-should-have-all-that-power-kanye-west-85-46-66.jpg
- https://i.kym-cdn.com/entries/icons/facebook/000/038/208/Anime_Girl_Punching_Wall_banner.jpg
- https://datinginmiddleearth.files.wordpress.com/2015/01/reading.jpg
- https://external-preview.redd.it/fudpTkeL0ltbeklZ6lRujiTeIPlye0823LKwAMkuyYY.jpg?auto=webp&s=4ac03fb1cc98a547d9b578938442686c117f0ca1
- https://freesvg.org/img/1317604469.png
- https://static.wikia.nocookie.net/characterprofile/images/f/fe/Lex_Luthor_Gods_Among_Us_001.png/revision/latest?cb=20160104030401