What is Lexical scoping??
-First step will be to look at gpt for some understanding.

Steps:
1.When a lambda function is created, the current env is attached to it forming a closure(Have to research to)

2.On a function called a new frame binds its parameters. Its parent pointer is the frame captured in 1.

3. While evaluating the body every identifier is followed up by following parent pointers outward until it is found or a top level failure occurs.

Example in scheme:

```
(define make-counter
(lambda()
        (let((n 0))        ;N lives in the LET fram
                (lambda()
                        (set! n(+ n 1))  ;Refers to the captured N
                        n))))

(define c1(make-counter))
(define c2(make counter))
```

Ex:So let's say we call c1 and c2

So if we do it like

```
 (c1)
(c1)
(c2)
```

The value of c1 would return 2 since we called it twice and it “remembers” its own N.
If we now call c2 the value would be 1 since the N for c2 has not been called yet.

That is what lexical scoping is.

Lexical vs dynamic
Dynamic scope and would be looked up in the caller's chain so each counter would have the same global n.

Since Scheme is weird here is a translation in JS

```
function makeCounter() {
  let n = 0;
  return function () {
    return ++n;
  };
}

const c1 = makeCounter();
const c2 = makeCounter();

console.log(c1()); // 1
console.log(c1()); // 2
console.log(c2()); // 1
```

Each c1 and c2 are referring to their own N and not a global one.

Lexical scoping is usually used when building environments so in our case we would need to prove that all our environments are lexically scoped.

Example:

```
(define-datatype env env?
  (empty-env)
  (extend-env (id symbol?) (val any?) (parent env?)))

(define apply-env
  (lambda (ρ sym)
    (cases env ρ
      (empty-env () (error "unbound variable" sym))
      (extend-env (id val parent)
        (if (eq? sym id) val (apply-env parent sym))))))
```

-Creating a closure stores P right inside the function value
-When evaluating the body use apply-env which never consults on dynamic scoping and only relies on the stored parent .

Bindings, Closures and their relationship to Lexical Scoping

What is a bound occurrence?
It sits inside the scope that introduces the name such as x in (lambda(x))

What is free occurrence?
-Not introduced locally
Ex: x in (lambda(y) (+ x y))

What is an unbound occurrence?
Lookup found nothing

Such as x in (lambda(y)(+ x y)
Where no surrounding x exists

Bound=has a binding right there
Free=free to look outside of local scope
Unbound=Owned by no one

What is a closure??
It's a two-part runtime value

1. Code pointer: the functions AST
2. Environment:pointer A snapshot of all the frames that satisfy its free variables at the moment when the function was created.

Ex:
```
(define make-adder
        (lambda (x)          ; x is bound here
                (lambda (y) (+ x y))))  ; inner lambda closes over x
```

Calling (make-adder 5) produces a closure

That environment turns the formerly free x inside the body into a bound run-time slot, while leaving y to be bound when the closure is invoked.

Lexical scope and its relationship with closure and Bound,free,unbound

Bound -> inside the same lexical scope:already in the innermost frame
Free-> elsewhere in the lexical nest:will be found by the closures parent pointer chain
Unbound-> outside entire program:raises error