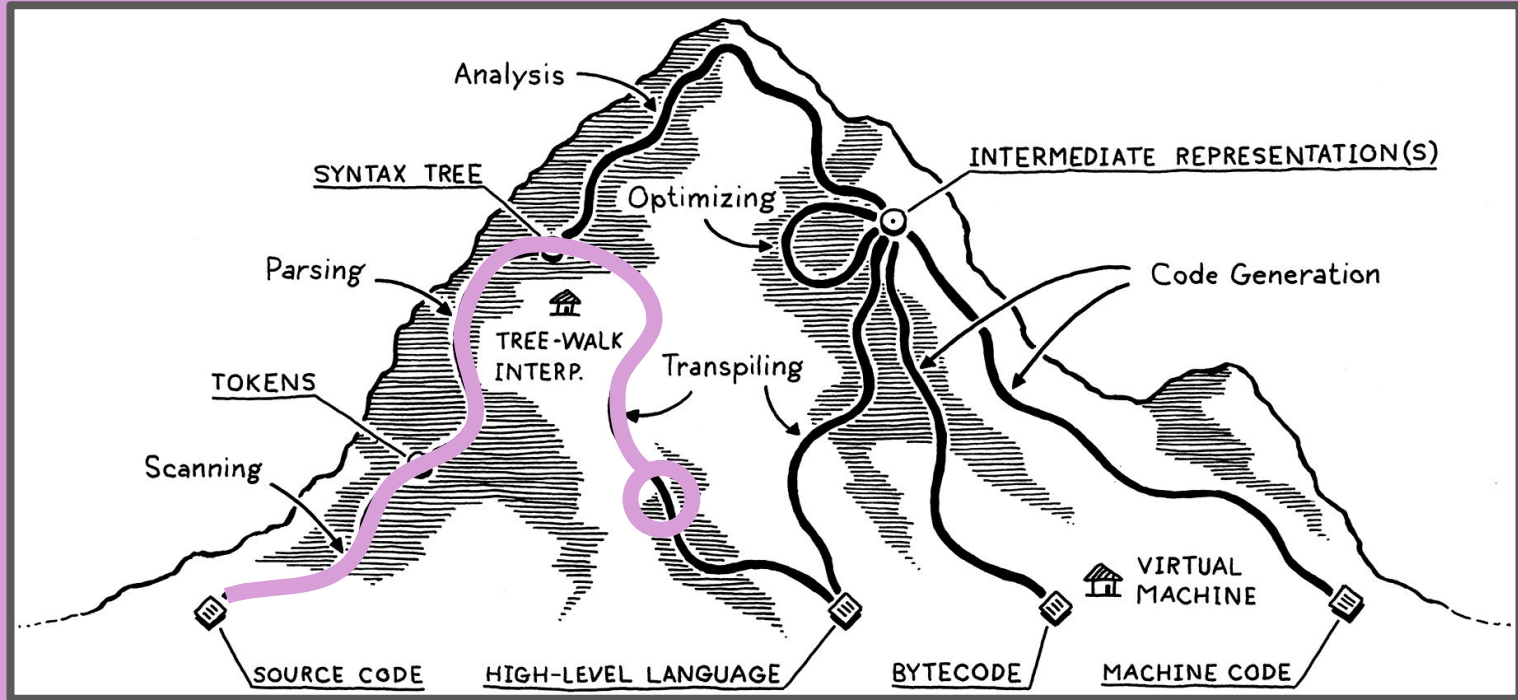




CMPSC 201: PROGRAMMING LANGUAGES



Defining the scope of "scope"

A **variable usage** refers to the **preceding** declaration with the same name in the **innermost scope** that encloses the expression where the variable is used.

```
var a = "outer";  
{  
  print a;  
  var a = "inner";  
}
```

```
var a = "outer";  
{  
  var a = "inner";  
  print a;  
}
```

Scoping out the problem

```
var a = "global";
```

```
{
```

```
  fun showA() {
```

```
    print a;
```

```
  }
```

What's our output?



```
  showA();
```

```
  var a = "block";
```

```
  showA();
```

```
}
```

Scoping out the problem

```
var a = "global";
```

```
{
```

```
  fun showA() {
```

```
    print a;
```

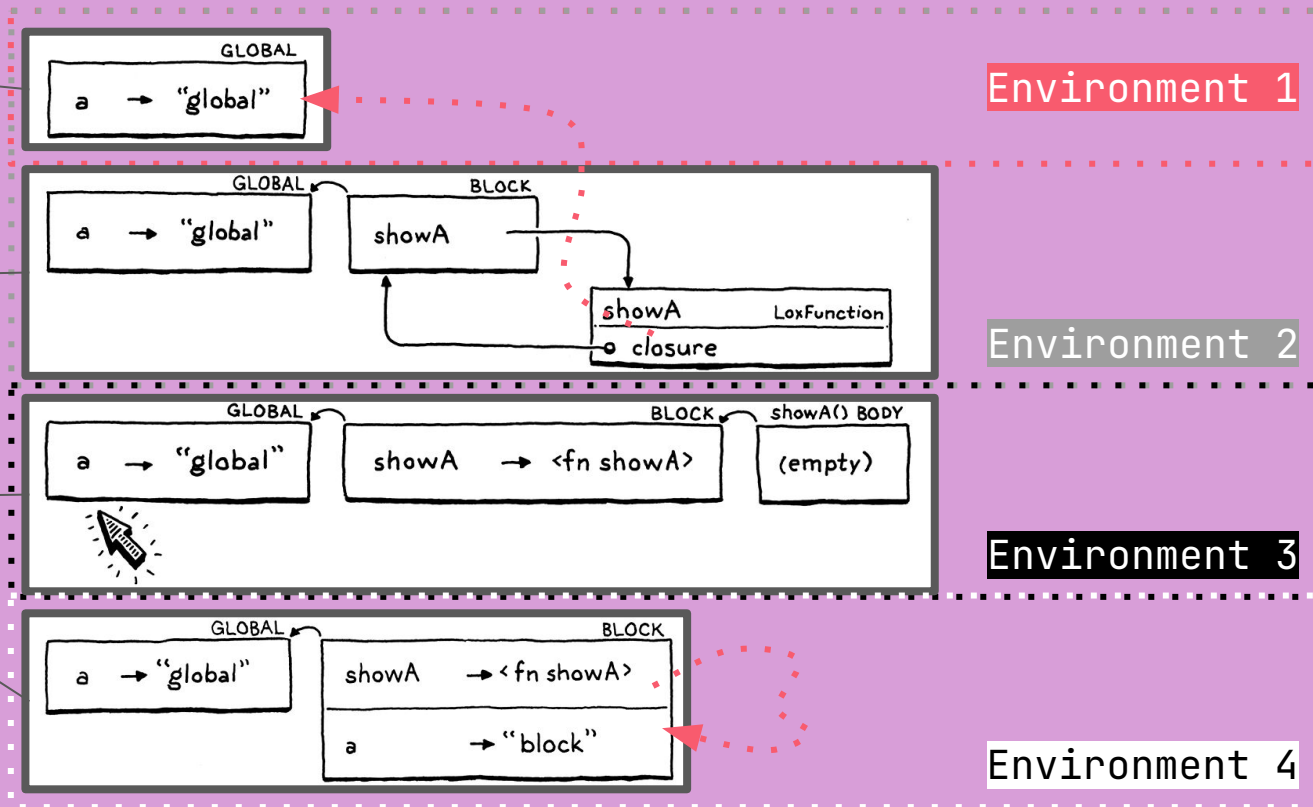
```
  }
```

```
showA();
```

```
var a = "block";
```

```
showA();
```

```
}
```



Lox's rule


But in our implementation, environments do act like the entire block is one scope, just a scope that changes over time. Closures do not like that...function[s] should capture a frozen snapshot of the environment as it existed at the moment the function was declared.

Static resolution

...a variable usage always resolves to the same declaration, which can be determined just by looking at the text.

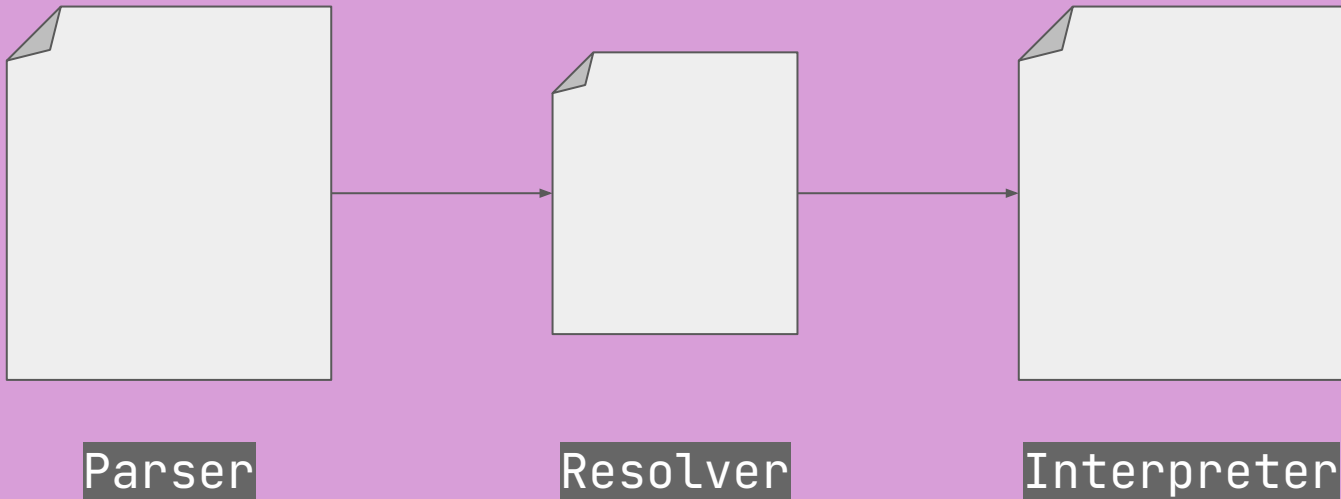
The solution to our leaky scope:

- Write code that looks at every variable *once*
- Figures out which declarations apply



Semantic
Analysis

Where it's at



Your Task

- Scoping has been implemented for you; your task is to check up on *unused* variables
- We've made several improvements to the language that aren't accounted for
 - We need to include those, not limited to:
 - `break`
 - `ternaries`
 - `New function definitions`

Programming corner: Stacks

Each environment contains its own “stack,” organized by *environment name*.

- Stacks always grow “down”
- Stacks grow by pushing values “onto” them
- Stacks shrink by popping values “off of” them

Push 5

5

Push 10

10

Push 9

9

Push 20

20

20

9

10

5

Pop 20

Pop 9

Pop 10

Pop 5

“Last In,
First Out”

Programming corner: HashMap

Java HashMaps operate like tables in memory.*

```
new HashMap<String, Boolean>()
```

↑
"Key" type

↖
"Value" type

* Kinda. Not really?