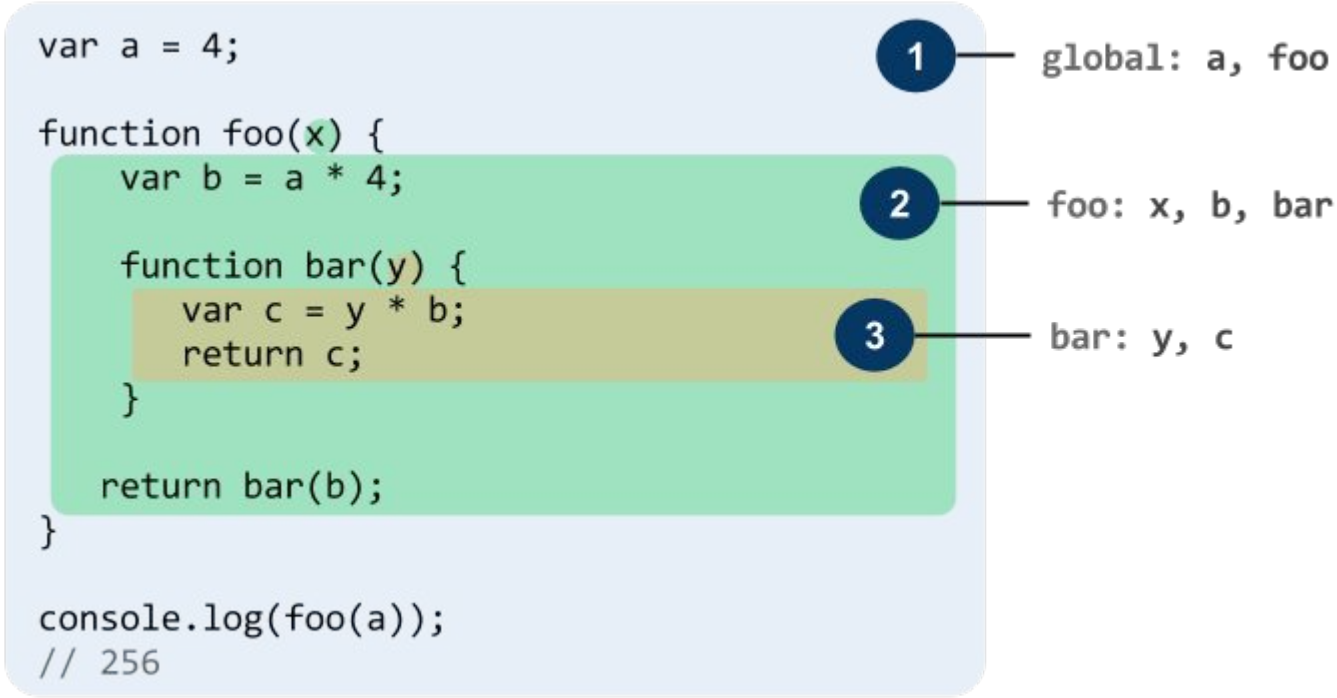
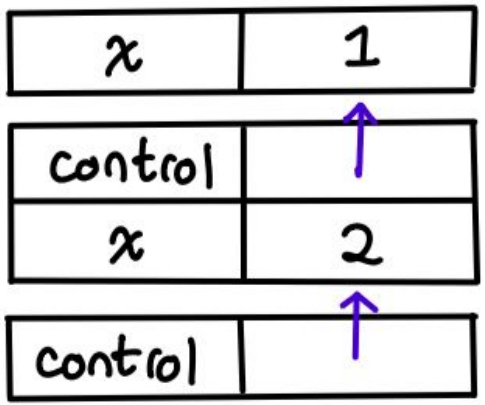
Dynamic Scoping

A Review of Lexical Scoping



(Dave Atchley - http://www.datchley.name/basic-scope/)

Dynamic Scoping




```
var x = 1;
function f() {
    console.log(x);
}
function g() {
    var x = 2;
    f();
}
g();
```

(Edward Yang)

Lua Examples

Dynamic Scoping in Emacs

“Consider the function Edit Picture, which is used to change certain editing commands slightly, temporarily, so that they are more convenient for editing text which is arranged into two-dimensional pictures. For example, printing characters are changed to replace existing text instead of shoving it over to the right. Edit Picture works by binding the values of parameter variables dynamically, and then calling the editor as a subroutine. The editor `exit' command causes a return to the Edit Picture subroutine, which returns immediately to the outer invocation of the editor. In the process, the dynamic variable bindings are unmade.” (Richard Stallman, EMACS: The Extensible, Customizable Display Editor)

“Some language designers believe that dynamic binding should be avoided, and explicit argument passing should be used instead… This cannot be done in an extensible system, however, because the author of the system cannot know what all the parameters will be. Imagine that the functions A and C are part of a user extension, while B is part of the standard system. The variable FOO does not exist in the standard system; it is part of the extension. To use explicit argument passing would require adding a new argument to B, which means rewriting B and everything that calls B. In the most common case, B is the editor command dispatcher loop, which is called from an awful number of places.” (Richard Stallman, EMACS: The Extensible, Customizable Display Editor)

More Applications

- Layout Engines
- Computer Graphics
- Environment Variables (sort of)

Exceptions are Dynamically Scoped Jumps

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
}
```

Dynamic Scoping Can Be Statically Typed

```
beside x y =
  let lhs = pretty d1 with ?width = ?xwidth
      rhs = pretty d2 with ?width = ?ywidth
  in
      zipConcat (fill ?xwidth (lines lhs))
                (lines rhs)
beside :: (?xwidth :: Int, ?ywidth :: Int) =>
          Doc -> Doc -> String
```

Lazy Evaluation

Problems with Laziness

- Can lead to unpredictable performance in real-time applications
  - Haskell lets you force thunks
- Thunks add overhead
  - Strictness analysis
- Thunks with side effects cause problems
- Thunk leaks

Extra

Implementation of Lexical Scoping