Consider the following JavaScript function, big, in which the two functions sub1 and sub2 are nested:

```
function big() {
  function sub1() {
    var x = 7;
    sub2();
  }
  function sub2() {
    var y = x;
  }
  var x = 3;
  sub1();
}
```

Under static scoping, the reference to the variable x in sub2 is to the x declared in the procedure big. This is true because the search for x begins in the procedure in which the reference occurs, sub2, but no declaration for x is found there. The search continues in the static parent of sub2, big, where the declaration of x is found. The x declared in sub1 is ignored, because it is not in the static ancestry of sub2.

In some languages that use static scoping, regardless of whether nested subprograms are allowed, some variable declarations can be hidden from some other code segments. For example, consider again the JavaScript function big. The variable x is declared in both big and in sub1, which is nested inside big. Within sub1, every simple reference to x is to the local x. Therefore, the ou x is hidden from sub1.

In Ada, hidden variables from ancestor scopes can be accessed with sel tive references, which include the ancestor scope's name. For example, if previous example function big were written in Ada, the x declared in could be accessed in sub1 by the reference big.x.

5.5.2 Blocks

Many languages allow new static scopes to be defined in the midst of ex able code. This powerful concept, introduced in ALGOL 60, allows a sec of code to have its own local variables whose scope is minimized. Such ables are typically stack dynamic, so their storage is allocated when the se is entered and deallocated when the section is exited. Such a section of is called a **block**. Blocks provide the origin of the phrase **block-stru language**.

The C-based languages allow any compound statement (a st sequence surrounded by matched braces) to have declarations and define a new scope. Such compound statements are called blocks. For if list were an integer array, one could write