On scoping

**🔍 Definitions**

**Scope** in computer programming refers to the region of a program where a particular name binding is valid. A name binding is the association between a name and an entity, such as a variable or function. The scope determines where in the code a name can be used to refer to the associated entity. Scopes help prevent name collisions by allowing the same name to refer to different entities in different parts of a program.

**Namespace** is a container that holds a set of identifiers (names) and ensures that all of them are unique within that context. Namespaces are used to organize code elements and to create globally unique types. They allow the same name to be used for different entities as long as they reside in different namespaces.

**🔄 Differences Between Scope and Namespace**

While both scope and namespace deal with the visibility and organization of identifiers in a program, they serve different purposes:

* **Scope** is concerned with the *visibility* of a name binding within a certain part of the program. It defines where in the code a particular name can be used to refer to its associated entity. For example, a variable declared inside a function has a scope limited to that function.
* **Namespace**, on the other hand, is about the *organization* of identifiers. It provides a context in which names are unique, allowing the same name to be used for different entities in different namespaces. For instance, two different classes in separate namespaces can have methods with the same name without conflict.

In essence, scope is about *where* a name is accessible, while namespace is about *how* names are grouped and distinguished from each other.

**🧠 Eelco Visser's Work on Scoping**

In Visser's framework, scopes and namespaces are distinct but related concepts. Scopes represent the regions of code where name bindings are valid, while namespaces are used to group related identifiers and manage name uniqueness. By modeling both scopes and namespaces explicitly, scope graphs enable precise and modular definitions of language semantics.

**🔍 Specificity in Scope Graphs**

In the scope graph framework, name resolution involves finding a path from a reference to a declaration. When multiple such paths exist, specificity rules determine which declaration a reference should resolve to. These rules ensure that the most appropriate declaration is chosen, maintaining the correctness of name binding.([Tufts Computer Science](https://www.cs.tufts.edu/comp/150FP/archive/eelco-visser/name-res-in-grace.pdf?utm_source=chatgpt.com))

The specificity rules can be summarized as follows:

1. **Lexical Proximity**: Declarations in the innermost (nearest) scope are preferred over those in outer scopes. This rule ensures that local declarations shadow outer ones.
2. **Explicit Imports Over Wildcard Imports**: If a name is imported both explicitly and via a wildcard, the explicit import takes precedence. This rule helps avoid ambiguity when multiple imports provide the same name.
3. **Direct Declarations Over Inherited Ones**: In object-oriented languages, a declaration in a class takes precedence over inherited declarations from superclasses. This rule aligns with the principle of method overriding.
4. **Avoiding Ambiguity**: If multiple declarations are equally specific and none of the above rules can disambiguate them, the reference is considered ambiguous, and resolution fails. This rule prevents unpredictable behavior due to ambiguous bindings.

These specificity rules are crucial for accurately modeling the name binding behavior of programming languages using scope graphs.