B.3.2 Web Legacy Compatibility for Block-Level Function Declarations

Prior to the Sixth Edition, the ECMAScript specification did not define the occurrence of a *FunctionDeclaration* as an element of a *Block* statement's *StatementList*. However, support for that form of *FunctionDeclaration* was an allowable extension and most browser-hosted ECMAScript implementations permitted them. Unfortunately, the semantics of such declarations differ among those implementations. Because of these semantic differences, existing web ECMAScript code that uses *Block* level function declarations is only portable among browser implementation if the usage only depends upon the semantic intersection of all of the browser implementations for such declarations. The following are the use cases that fall within that intersection semantics:

1. A function is declared and only reference within a single block
    - A function declaration with the name *f* is declared exactly once within the function code of an enclosing function *g* and that declaration is nested within a *Block*.
    - No other declaration of *f* that is not a **`var`** declaration occurs within the function code of *g*
    - All references to *f* occur within the *StatementList* of the *Block* containing the declaration of *f*.

2. A function is declared and possibly used within a single block but also referenced within subsequent blocks.
    - A function declaration with the name *f* is declared exactly once within the function code of an enclosing function *g* and that declaration is nested within a *Block*.
    - No other declaration of *f* that is not a **`var`** declaration occurs within the function code of *g*
    - References to *f* may occur within the *StatementList* of the *Block* containing the declaration of *f*.
    - References to *f* occur within the function code of *g* that lexically follows the *Block* containing the declaration of *f*.

3. A function is declared and possibly used within a single *Block* but also referenced by an inner function definition that is not contained within that same *Block*.
    - A function declaration with the name *f* is declared exactly once within the function code of an enclosing function *g* and that declaration is nested within a *Block*.
    - No other declaration of *f* that is not a **`var`** declaration occurs within the function code of *g*
    - References to *f* occur within another function *h* that is nested within *g* and no other declaration of *f* shadows the references to *f* from within *h*.
    - All invocations of *h* occur after the declaration of *f* has been evaluated.

Use cases 2 and 3 for a given function declaration f might occur within the same function.

The first use case is interoperable with the inclusion of *Block* level function declarations in the sixth edition. Any pre-existing ECMAScript code that employees that use case will operate using the Block level function declarations semantics defined by clauses 10 and 13 of this specification.

Sixth edition interoperability for the second and third use cases requires the following extensions to the clauses 10 and 13 semantics. These extensions are applied to a non-strict mode functions *g* if the above pre-conditions of use cases 2 and/or 3 exist at the time of static semantic analysis of *g*. However, the last pre-condition of use case 3 is not included in this determination and the determination is only applied to function declarations that are nested within syntactic constructs that are specified in the Fifth edition of this specification.

1. Let *B* be environment record for the construct within *g* that introduces a new environment contour and which most closely encloses the declaration of *f*, all function code references to *f*, and the definitions of all nested functions that contain syntactically unshadowed references to *f*. This syntactic construct may be the definition of *g* itself, in which case *B* is the function environment record for *g*.

2. As part of the instantiation of *B*, its CreateMutableBinding concrete method is called with arguments "f" (the string name of the function) and **false**. This creates an unitialised binding for the name *f*. Any reference that resolves to that binding prior to step 3 below will throw a **ReferenceError** exception.
3. When the InitializeBinding concrete method is used to initialise the binding for the function declaration *f* also invoke InitializeBind on *B* using the same arguments.

If an ECMAScript implication has a mechanism that produces diagnostic warning messages, a warning should be produced for each function *g* for which the above steps are performed.