Two members are considered identical when

- they are public properties with identical names, optionality, and types,
- they are private properties originating in the same declaration and having identical types,
- they are identical call signatures,
- they are identical construct signatures, or
- they are index signatures of identical kind with identical types.

Two call or construct signatures are considered identical when they have the same number of type parameters and, considering those type parameters pairwise identical, have identical type parameter constraints, identical number of parameters with identical kind (required, optional or rest) and types, and identical return types.

Note that, except for primitive types and classes with private members, it is structure, not naming, of types that determines identity. Also, note that parameter names are not significant when determining identity of signatures.

Private properties match only if they originate in the same declaration and have identical types. Two distinct types might contain properties that originate in the same declaration if the types are separate parameterized references to the same generic class. In the example

```
class C<T> { private x: T; }

interface X { f(): string; }

interface Y { f(): string; }

var a: C<X>;
var b: C<Y>;
```

the variables 'a' and 'b' are of identical types because the two type references to 'C' create types with a private member 'x' that originates in the same declaration, and because the two private 'x' members have types with identical sets of members once the type arguments 'X' and 'Y' are substituted.

3.8.3 Subtypes and Supertypes

S is a ***subtype*** of a type *T*, and *T* is a ***supertype*** of *S*, if one of the following is true, where *S'* denotes the apparent type (section 3.8.1) of *S*:

- *S* and *T* are identical types.
- *T* is the Any type.
- *S* is the Undefined type.
- *S* is the Null type and *T* is not the Undefined type.
- *S* is an enum type and *T* is the primitive type Number.
- *S* is a string literal type and *T* is the primitive type String.
- *S* and *T* are type parameters, and *S* is directly or indirectly constrained to *T*.
- *S'* and *T* are object types and, for each member *M* in *T*, one of the following is true: