The identity (`===`) operator behaves identically to the equality (`==`) operator except no type conversion is done, and the types must be the same to be considered equal.

The `==` operator will compare for equality *after doing any necessary type conversions*.
The `===` operator will **not** do the conversion, so if two values are not the same type `===` will simply return `false`.
Both are equally quick.

To quote Douglas Crockford's excellent JavaScript: The Good Parts:

> JavaScript has two sets of equality operators: `===` and `!==`, and their evil twins `==` and `!=`.
> The good ones work the way you would expect. If the two operands are of the same type and have the same value, then `===` produces `true` and `!==` produces `false`.
> The evil twins do the right thing when the operands are of the same type, but if they are of different types, they attempt to coerce the values. the rules by which they do that are complicated and unmemorable.
>
> These are some of the interesting cases:

```
'' == '0'          // false
0 == ''            // true
0 == '0'           // true

false == 'false'   // false
false == '0'       // true

false == undefined  // false
false == null       // false
null == undefined   // true

' \t\r\n ' == 0     // true
```

> The lack of transitivity is alarming.
>
> My advice is to never use the evil twins. Instead, always use `===` and `!==`.
> All of the comparisons just shown produce `false` with the `===` operator.