11.9.3 The Abstract Equality Comparison Algorithm

The comparison x == y, where x and y are values, produces **true** or **false**. Such a comparison is performed as follows:

1. If Type(*x*) is the same as Type(*y*), then
   - If Type(*x*) is Undefined, return **true**.
   - If Type(*x*) is Null, return **true**.
   - If Type(*x*) is Number, then
     1. If *x* is **NaN**, return **false**.
     2. If *y* is **NaN**, return **false**.
     3. If *x* is the same Number value as *y*, return **true**.
     4. If *x* is **+0** and *y* is **−0**, return **true**.
     5. If *x* is **−0** and *y* is **+0**, return **true**.
     6. Return **false**.
   - If Type(*x*) is String, then return **true** if *x* and *y* are exactly the same sequence of characters (same length and same characters in corresponding positions). Otherwise, return **false**.
   - If Type(*x*) is Boolean, return **true** if *x* and *y* are both **true** or both **false**. Otherwise, return **false**.
   - Return **true** if *x* and *y* refer to the same object. Otherwise, return **false**.
2. If *x* is **null** and *y* is **undefined**, return **true**.
3. If *x* is **undefined** and *y* is **null**, return **true**.
4. If Type(*x*) is Number and Type(*y*) is String,
   return the result of the comparison *x* == ToNumber(*y*).
5. If Type(*x*) is String and Type(*y*) is Number,
   return the result of the comparison ToNumber(*x*) == *y*.
6. If Type(*x*) is Boolean, return the result of the comparison ToNumber(*x*) == *y*.
7. If Type(*y*) is Boolean, return the result of the comparison *x* == ToNumber(*y*).
8. If Type(*x*) is either String or Number and Type(*y*) is Object,
   return the result of the comparison *x* == ToPrimitive(*y*).
9. If Type(*x*) is Object and Type(*y*) is either String or Number,
   return the result of the comparison ToPrimitive(*x*) == *y*.
10. Return **false**.

NOTE 1Given the above definition of equality:

- String comparison can be forced by: `"" + a == "" + b`.
- Numeric comparison can be forced by: `+a == +b`.
- Boolean comparison can be forced by: `!a == !b`.

NOTE 2The equality operators maintain the following invariants:

- `A != B` is equivalent to `!(A == B)`.
- `A == B` is equivalent to `B == A`, except in the order of evaluation of `A` and `B`.

NOTE 3The equality operator is not always transitive. For example, there might be two distinct String objects, each representing the same String value; each String object would be considered equal to the String value by the `==` operator, but the two String objects would not be equal to each other. For Example:

- `new String("a") == "a"` and `"a" == new String("a")` are both **true**.
- `new String("a") == new String("a")` is **false**.

NOTE 4Comparison of Strings uses a simple equality test on sequences of code unit values. There is no attempt to use the more complex, semantically oriented definitions of character or string equality and collating order defined in the Unicode specification. Therefore Strings values that are canonically equal according to the Unicode standard could test as unequal. In effect this algorithm assumes that both Strings are already in normalised form.