What the Heck is Coercion ?

- Coercion is the implicit type conversion that happens when Javascript tries to perform an operation with values of different types.

- In order to understand, let's understand what Abstract Operations Are ?, since the Coercion algorithm internally uses many abstract operations to do Type Conversion.

- Abstract Operations are not a part of the ECMAScript language, they are defined solely to aid the specification of the semantics of the ECMAScript language.

- The ECMAScript language implicitly performs automatic type conversion as needed !
- These abstract operations are not available to end users, meaning we can not call it or see its code on using console.log. The Js internally uses it and calls it whenever it's necessary.

- The most important Abstract Operation when dealing with Coercion is ToPrimitive(input, [, preferredType])

ToPrimitive(input, [, PreferredType])

- It's the most important Abstract operation of Js, as other abstract operations heavily depend on it.

- It takes an input argument and an optional argument i.e PreferredType.

- The ToPrimitive converts the input argument to a non-Object type.

- If an object is capable of converting to more than one primitive type, it may use the optional hint PreferredType to favor that type.

Algorithm Details :

1. Assert: let input is an ECMPAScript language value.

   First we consider or assume that the given argument passed to ToPrimitive(input) be a valid Js type
   If input passed is already an primitive type, we will exit the function

   Else if input passed is an Object then -

2. If PreferredType is not present, let hint be “default”

   Consider an example of Abstract equality (==), There we don’t pass any hint !
   Another example is when we perform addition of two operands
   It’s Js who will pass the hint automatically, not us !

3. Else if PreferredType is hint String, let hint be “string”

   Consider an example of ToString() abstract operation, which will pass hints as “string” in its algorithm and ToString( ) is called in addition operation.

4. Else if PreferredType is hint Number, let hint be “number”

   Consider an example of ToNumber() abstract operation, which will pass hints as “number ” in its algorithm and ToNumber() is called in subtraction operation.

5. If hint is “default”, set hint to “number”

6. Let exoticToPrim be ? GetMethod(input, @@toPrimitve)

   The above algorithm tries to get a method called “@@toPrimitive” from an input object. This method if it exists, is a way for objects to define their custom behavior when being converted to a primitive value

7. If exoticToPrim is not undefined then, we call again another method called Call(exoticToPrim, input, <hint>).

   The above algorithm calls this “@@toPrimitve” method passing the input object as the value and hint (either “string”, “number” or “default”) as an argument.
   The result of this method call is stored in “result” !

8. If Type(result) is not an object, return result i.e we get the primitive value.

9. Else throw a TypeError Exception !

10. Else if hint is “default”, set hint to “number”

11. Then call the method OrdinaryToPrimitive(input, hint)

12. Return the input i.e final result

NOTE: When ToPrimitive is called with no hint, then it generally behaves as hint where Number. However, objects may override this behaviour by defining a @@toPrimitive method.

NOTE: In order to add “@@toPrimitive“ method to an object is Js, you can use the ‘Symbol.toPrimitive’ symbol.
This method allows you to define how an object should be converted to a primitive value (e.g string or number) when Js Engines tries to do so !

OrdinaryToPrimitive(input, hint)

1. Assert: Type(O) is object, here O corresponds to input passed

2. Assert: Type(hint) is **String**, and it’s value is either “string” or “number”

3. If hint is a “string”, then
   Create a methodNames as array with [toString, valueOf]
   These toString and valueOf methods are not abstract operations !
   Meaning we can call them, use them and see in the console.log !!

4. Else, we will store it as [valueOf , toString]

5. Then Js simply calls these methods based on hints, if it gets primitive value then returns result, else calls second method in an array to get primitive value, If that also returns non primitive value, Then throws TypeError Exception !!

NOTE: By default valueOf( ) returns the object itself when called on an object
And toString( ) returns [object Object]