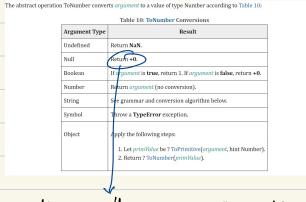


Li) Cocrcion → Type Conversion 2 + 3 = 5 2 + 3" = 23 "1" - 1 = 0 L2) Definition of Coercion: Whenever we do a operation, based on the ilp we can actually convert the ilp for operation we can convert type of input Two types of casting: i) Explicit: This conversion can be manually done by us 2) Implicit: Automatically converts the type. G cocrcion (3) Abstract Operation: These operations are NOT a part of the ECMA Script Conguage, they are defined here to Solely to aid the specifications of the semantics of the ECMA Script.

We as developer cannot use these operation directly.

(5) To Number Abstract Operation:

To Number converts argument to a value of type



(6) to Namber on String:

```
8  // toNumber on strings
9
10  console.log(1 - "453"); // -452
11  console.log(1 - "43rizon"); // 1 - NaN => NaN
```

Assignment

// Assignment why its -9?
console.log(1 - "0xa"); // -9

Ly hexa decimal => 10 decimal



12.8.4.1 Runtime Semantics: Evaluation

AdditiveExpression : AdditiveExpression - MultiplicativeExpression

- 1. Let *lref* be the result of evaluating *AdditiveExpression*.
- 2. Let *lval* be ? GetValue(*lref*).
- 3. Let rref be the result of evaluating MultiplicativeExpression.
- 4. Let rval be ? GetValue(rref).
- 5. Let *lnum* be ? ToNumber(*lval*).
- 6. Let rnum be ? ToNumber(rval).
- 7. Return the result of applying the subtraction operation to *lnum* and *rnum*. See the note below 12.8.5.



7.1.1 ToPrimitive (input[, PreferredType])

The abstract operation ToTrimitive takes an input argument and an optional argument PreferredType: The abstract operation ToTrimitive converts its 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 favour that type. Conversion occurs according to the following algorithm:

- Assert: input is an ECMAScript language value.
 If Type(input) is Object, then
- a. If PreferredType is not present, let hint be "default".
- a. If Preferred type is not present, let nint be "default".
 b. Else if Preferred Type is hint String, let hint be "string".
- c. Else PreferredType is hint Number, let hint be "number
- d. Let exoticToPrim be ? GetMethod(input, @@toPrimitive).
- d. Let exoticToPrim be ? GetMethod(input, 6
 e. If exoticToPrim is not undefined, then
 - i. Let result be ? Call(exoticToPrim, input, « hint »).
 - ii. If Type(result) is not Object, return result.
- iii. Throw a TypeError exception.
 f. If hint is "default", set hint to "number".
- g. Return ? OrdinaryToPrimitive(input, hint).

3. Return input.



| () optional |
|--|
| Convert the input to non-object type. |
| |
| preferred type: In case where i/p can be converted to maltiple type we take the decision using this. |
| To Prinitive Cora: |
| |
| C . D . Lavel T in MAT among |
| Casel: Preferred Type is NOT present |
| hint & default" |
| Case 2 · Preferred Type is a String default -> ran |
| hint = "String" |
| Case 3: Preformed Type is a number |
| l J' |

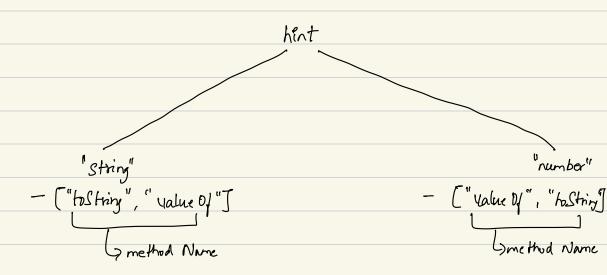
Ordinary To Primitive:

Ordinary To Primitive (input, hint)

7.1.1.1 OrdinaryToPrimitive (0, hint) When the abstract operation OrdinaryToPrimitive is called with arguments 0 and hint, the following steps are taken: 1. Assert: Type(in) is Object. 2. Assert: Type(hint) is String and its value is either "string" or "number". 3. If hint is "string", then a. Let methodNames be « "toString", "valueOf" ». 4. Else, a. Let methodNames be « "valueOf", "toString" ». 5. For each name in methodNames in List order, do a. Let method be ? Get(0, name). b. If [IsCallable(method) is true, then

i. Let result be ? Call(method, 0). ii. If Type(result) is not Object, return result.

6. Throw a TypeError exception.



```
let obj = {
    toString() {
        // by default gives [Object object]
        return "Let's learn JAVASCRIPT"; // still you can override
    },
    valueOf() {
        // by default it returns the same object
    }
}
```

Coerción in Addition Operator

12.8.3.1 Runtime Semantics: Evaluation AdditiveExpression: AdditiveExpression + MultiplicativeExpression 1. Let Iref be the result of evaluating AdditiveExpression. 2. Let Ival be? GetValue(Iref). 3. Let rref be the result of evaluating MultiplicativeExpression. 4. Let rval be? GetValue(rref). 5. Let Iprim be? ToPrimitive(Ival). 6. Let rprim be? ToPrimitive(rval) 7. If Type(Iprim) is String or Type(rprim) is String, then a. Let Istr be? ToString(Iprim). b. Let rstr be? ToString(rprim). c. Return the string-concatenation of Istr and rstr. 8. Let Inum be? ToNumber(Iprim). 9. Let rnum be? ToNumber(Iprim).

```
# + "10" => 10 | Onary Operator
```

10. Return the result of applying the addition operation to *lnum* and *rnum*.

```
1 let obj = {};
2
3 console.log(obj); // {|}
5 console.log("18" + obj); // 18[object Object]
6
7 console.log(18 + obj); // 18[object Object]
8
```

Intro to To Boolean

| Argument Type | Result | | |
|---------------|--|--|--|
| Undefined | Return false. | | |
| Null | Return false. | | |
| Boolean | Return argument. | | |
| Number | If argument is +0, -0, or NaN, return false; otherwise return true. | | |
| String | If argument is the empty String (its length is zero), return false; otherwise return true. | | |
| Symbol | Return true. | | |
| Object | Return true. | | |

```
[undefined, null, NaN, to, -0, false] > they are falsy value
```

```
1 let x = 10;
2 console.log(!x); // false
3
4 let y = undefined;
5 console.log(!y); // true
```

Equality Operator:

$$==$$
 \rightarrow obstract equality
 $===$ \rightarrow Strict equality

Both abstract & strict equality check for type. but what they do with it depends.

Abstract Equality & Strict Equality:

Abstract Equality i

If type of (2) is same as type of (y)

=) then works as strict equality

If type is NOT same

x = null y = indefined

the return true # x = indefined y = null

the return true

if type of x = Number type of y = String

then converts $y \rightarrow foNumber$ tha again does the compassion

if type of x = String type of y = number

than convert & -> ho Number

again does the compassion

if type of x = Boolean

then convert $x \rightarrow b N amber (0 \rightarrow 1 \rightarrow 0)$

then do compassion

the convert $y \rightarrow$ to Number

the convert $y \rightarrow$ to Number

the type of x is either String, Number, Symbol and type of y is Object

then convert $y \rightarrow$ to Primitive the compare

the if type of x is object and type of y is string, number, symbol

then convert $x \rightarrow$ to Primitive then compare

IT none of above rules follow then between false

Strict Equality:

If type of (x) is different from type of (y)
then return false

If type (a) is Number then # If x is NAN, return false (don't case value # If y is NAN, return false (don't care value Q x) Gr-Nan the return false. # If x is same Number value as y return true. (0=== 10 >> The

If
$$x$$
 is same Number value as y return true.

 $|0===10 \Rightarrow$ The

If $x=0$ and $y=-0$ return true

Return false

If Same Value Non Number (N, y)

```
7.2.12 SameValueNonNumber ( x, y )

The internal comparison abstract operation SameValueNonNumber(x, y), where neither x nor y are Number values, produces true or false. Such a comparison is performed as follows:

1. Assert: Type(x) is not Number.
2. Assert: Type(x) is the same as Type(y).
3. If Type(x) is Unlettine true.
4. If Type(x) is Null. return true.
5. If Type(x) is String, then
a. If x and y are exactly the same sequence of code units (same length and same code units at corresponding indices), return true; otherwise, return false.
6. If Type(x) is Boolean, then
a. If x and y are both true or both false, return true; otherwise, return false.
7. If Type(x) is Symbol, then
a. If x and y are both the same Symbol value, return true; otherwise, return false.
8. If x and y are the same Object value, return true; otherwise, return false.
8. If x and y are the same Object value, return true; otherwise, return false.
```

Coercion Special Casus

Naw! Not a Number:

H Negative Zero:

```
■ 12,negativeZerojs U X

05-First Pillar of IS > ■ 12,negativeZerojs > ② customSign

1 let x = -0;
2 console.log(x === 0); // true

3

4 console.log(Object.is(x, -0)); // T

5 console.log(Object.is(x, 0)); // F

6

7 console.log(Math.sign(-3)); // -1

8 console.log(Math.sign(2)); // 1

9 console.log(Math.sign(-0)); // -0

10 console.log(Math.sign(0)); // 0
```

Boxing

Uncaught SyntaxError: Invalid or unexpected token

(undefined).toString();
> Uncaught TypeError: Cannot read properties of undefined (reading 'toString')
at <anonymous>:1:13

(1) hosting >

Here comes the concept of boxing concept.

Boxing: Your J.S types get converted to non-primitive when you try to do boxing.