HamdyTechHub

Contents

Logic

Unary Plus + and Negation Operators -

بنستخدم + أو – قبل ال String أو Boolean عشان نحوله لـ Number بدل ما نستخدم ()Number

- Unary Plus: Returns number if it's not a number

```
console.log(+"-100")
```

المفروض ده string فهتحوله ل number والناتج هيكون -100 (سالب فـ موجب بـ سالب)

```
console.log(+"Osama")
```

هيرجع NaN عشان ده text ومش هينفع نحوله لرقم

```
console.log(+0xff)
```

ده Hexadecimal Numerical System وهيرجع في الاخر 255

```
console.log(+"")              // 0
console.log(+"null")          // 0
console.log(+"false")         // 0
console.log(+"true")          // 1
```

- Negation Operator: Returns number if it's not a number and **negates it**

```
console.log(-"-100")
```

المفروض ده string فهتحوله ل number والناتج هيكون 100 (سالب فـ سالب بـ موجب)

```
console.log(-"")              // -0
console.log(-"null")          // -0
console.log(-"false")         // -0
console.log(-"true")          // -1
```

استخدام ال unary مميز عن ال ()Number في انك تقدر تحول لسالب

طب لو هيكون فيه عمليات (يعني هنستخدم - , + بس في النص ما بين operands)؟ Type Coercion

نبذة سريعة عن type coercion قبل ما نخش علي اللي جاي: (ده لتحويل البيانات من نوع لنوع اثناء تشغيل الكود)

```
let a = "10";
let b = 20;
let c = true;
console.log(a + b)            // + triggers string: 1020
console.log(a - b)            // - triggers number: -10
console.log("" - 2)           // - triggers number: 0-2 = -2
console.log(false + true)     // false → 0 , true → 1: 1
console.log(a + c)            // 21
console.log(a + b + c)        // + triggers string: 1020true
```

Type Conversion and Type Coercion

- **Type conversion (Explicitly)**: is to **manually** convert from one type to another.

هنا انا بستخدم method عشان احول زي مثلا Number()

```
const inputYear = "1991";
console.log(inputYear+18);          // 199118 (unary plus triggers string Type coercion)
```

- نستخدم Number() عشان نعمل covert لـ String

```
const inputYear = "1991";
Number(inputYear);
console.log(inputYear+18);   // still 199118
```

لسه الناتج زى ما هو طب ليه ؟ لأنك مغيرتش في القيمة الاصلية فيا اما تعملها من تعريف "variable" او من جملة الطباقة

```
const inputYear = "1991";
console.log(Number(inputYear) +18); // 2009
```

طب ايه اللى يحصل لو حاولت احول "string" لا يحتوى على ارقام ؟

```
console.log(Number("jonas")); //NaN
```

هيطلع "NaN" واللى اختصارها "Not a number" وعلى فكرة النوع بتاع "NaN" هو "number"

```
console.log(typeof NaN);      // number
```

- So, JavaScript gives us "NaN" whenever an operation that involves numbers fails to produce a new number.

ولو عاوز احول من رقم ل "String" استخدم "String() method"

- **Number() Method**

```
Number("  10");        // returns 10
Number("10  ");        // returns 10
Number(" 10  ");       // returns 10
Number("10.33");       // returns 10.33
Number("10,33");       // returns NaN
Number("10 33");       // returns NaN
```

- **Type Coercion (Implicitly)**: is the **automatic** or implicit conversion of values from one data type to another.

```
console.log('I am ' + 23 + ' years old'); // I am 23 years old (String)
```

- Plus operator triggers a coercion to **String**, but not all operations do type coercion to String. For example:

- Minus operator

```
console.log('23' - '10' - 3); // 23-10-3 = 10 (number)
```

- Multiplication / division operator

```
console.log('23' * '10');     // 230 (number)
```

- Logical operator

```
console.log('23' > ‘10’); // true
```

```
console.log(2+3+4+'5') // ‘95’
```

```
console.log(2+3+'5'+4) // ‘554’
```

- **String conversion**

```
String(123) // explicit
123 + ''    // implicit
```

- **Boolean conversion**

```
Boolean(2)          // explicit
if (2) { ... }      // implicit due to logical context
!!2                 // implicit due to logical operator
2 || 'hello'        // implicit due to logical operator
```

- **Numeric conversion**

```
Number('123')   // explicit
+'123'          // implicit
123 != '456'    // implicit
4 > '5'         // implicit
5/null          // implicit
true | 0        // implicit
```

Here is how primitive values are converted to numbers:

```
Number(null)                // 0
Number(undefined)           // NaN
Number(true)                // 1
Number(false)               // 0
Number(" 12 ")              // 12
Number("-12.34")            // -12.34
Number("\n")                // 0
Number(" 12s ")             // NaN
Number(123)                 // 123
```

تحدي 1 من الزيرو: خمن ال output من كله

```
let a = 10;
let b = "20";
let c = 80;
console.log(++a + +b++ + +c++ - +a++);
console.log(++a + -b + +c++ - -a++ + +a);
console.log(--c + +b + --a* +b++ - +b * a + --a - +true);
```

تحدي 2 من الزيرو: من اللي عندك طلع ال output المطلوب

```
let d = "-100";
let e = "20";
let f = 30;
let g = true;
console.log(); //2000
console.log(); //173
```

ملاحظة مهمة: أي حاجة نكتبها في مكان الشرط (ما بين قوسين ()if أو : ? ()) بتتحول علطول لـ Boolean وده type coercion

Truthy and Falsy Values

- **Falsy values**: are values that are not exactly false, but will become false when we try to convert them into a Boolean.

In JavaScript we have only 5 falsy values:

1) 0 2) "" 3) undefined 4) null 5) NaN

- فلما تعمل "explicit conversion" ب "Boolean() method" هيحولولك "false"

فيه فرق بين

```
console.log(Number(undefined));      // NaN
console.log(Boolean(undefined));     // false
```

- خد بالك من الاكواد ده عشان هتلخبطك

```
console.log(Boolean("false"));       // true
console.log(Boolean({}));            // true
```

- لأن اول واحدة هتكون "String" بها حروف وليست "Empty String"

- وتانى واحدة تعبر عن "Empty object" وهى ليست من "Falsy values"

طب بالنسبة ل "Implicit conversion" او "Coercion" في ال "JavaScript" امتى بيحصل ؟

Type coercion to Booleans (Implicit):

1) Logical Operators 2) Logical context

Example in logical context:

```
const money = 0;
if (money){
  console.log("Don't spend it all!");
}else{
  console.log("You should get a job!");
}      // output You should get a job!
```

- في الكود ده ال "JavaScript" هيعمل "coercion" اتوماتيك وهيحول ال "0" إلى "false" عشان كده طبع جملة "else" ، وعلى فكرة ده تعتبر "Bug" ، ولو خليت "money" بأى رقم تانى غير الصفر هيطبع جملة "if"

```
const money ;
if (money){
  console.log("Don't spend it all!");
}else{
  console.log("You should get a job!");
}      // output You should get a job!
```

- برضه نفس الكلام هيطبع تانى جملة لأن كده "money" هتكون "undefined" عشان معملتهاش "set" لقيمة

Equality and inequality Operators

- We have two types:

خد بالك ان العلامات لازقين في بعض ومينفعش تعمل مسافات بينهم

1) Strict equality (= = =) or Strict inequality (! = =)

```
const age = '18';

if(age === 18) console.log("Equals");
```

- مش هيطبع حاجة لأن في "Strict" مينفعش تقارن اتنين مختلفين فى "type" فكده لازم تحول ال "String" إلى "number" عن طريق "manually conversion with Number() method"

2) Loose equality (= =) or loose inequality (! =)

```
const age = '18';

if(age == 18) console.log("Equals");
```

- هنا ال "JavaScript" بيعمل "coercion" عشان كده طبع الجملة لأنه حول ال "String" إلى "number"

- لكن بالرغم من كده "Strict" افضل بكثير من ال "Loose" حتى لو هتعمل "conversion" ، فأنك متعرفش ال "Loose" واتعامل علطول مع "Strict"

Short Circuiting (&& and | |)

The result of the Boolean operator doesn't always have to be Boolean.

```
console.log(3 || 'Jonas'); // output: 3
```

ال Boolean operators ليها 3 خواص:

- They use any datatype
- They return any datatype
- **Short Circuiting** evaluation

يعني ايه short circuiting ؟ بالنسبة لـ OR (مهتم بالـ Truthy)

يعني لو اول قيمة هي truthy value هنرجع القيمة ده علطول، وال JS مش هيتعب نفسه ويشوف ال operand التاني، بالتالي بقي خمن القيمة الاتية:

OR | | Short Circuiting:

```
console.log('' || 'Jonas'); // Jonas
console.log(true || 0); // true
console.log(undefined || null); // null
```

اشمعنا خد null وليس undefined مع الاتنين falsy؟ لأنه بيكمل ويشوف حاجة truthy فملقاش ووقف علي اخر حاجة وهي null فعشان كده رجعها

AND && Short Circuiting:

بالنسبة لـ AND (مهتم بالـ Falsy)

لو اول قيمة هي falsy value هيرجع القيمة ده علطول، وال JS مش هيتعب نفسه ويشوف ال operand التالي

```
console.log(0 && 'Jonas');  // 0
console.log(7 && 'Jonas');  // Jonas
console.log("Hello" && 23 && null && "Jonas");  // null
```

ليه رجع Jonas في تاني واحدة؟ برضه نفس التفكير بتاع اللي معلم عليه فوق وهوا، انه بيدور علي falsy value فملقاش وواقف عند اخر حاجة وهوا Jonas فرجعها

مثال عملي: ناخده علي AND Operator، احنا ممكن نستبدله بـ if statement في حالة زي اللي جاية:

```
if(restaurant.orderPizza){
    restaurant.orderPizza('mushrooms', 'spinach');
}
```

كده لو في حالة ان restaurant.orderPizza موجودة هينفذ اللي جوا، طب لو هنعملها بـ &&

```
restaurant.orderPizza && restaurant.orderPizza('mushrooms',
'spinach');
```

كده لو هي مش موجودة يعني undefined يعني filthy value هترجع علطول، لكن لو لأ هيضطر يكمل للاخر ويرجعلك اخر حاجة وقف عندها

The Nullish Coalescing Operator (??) ES 2020

Nullish: returns true if the first operator is falsy but not nullish, true (**false**, **0**, **""** are truthy)

(null, undefined are falsy / nullish)

هوا OR بس الزيادة انه معتبر 0 و '' و false دول Truthy

```
restaurant.numGuests = 0;
const guests = restaurant.numGuests || 10;
console.log(guests)    // 10
```

ال Nullish بيكون بقي ?? فكرته بقي: انه بيعتبر ال "" , 0 الاتنين truthy فلو لقاهم يرجعهم عادي

```
restaurant.numGuests = 0;
const guests2 = restaurant.numGuests ?? 10;
console.log(guests2)  // 0
```

يعني من الاخر لو اول operand كان بـ null / undefined ساعتها بس هيرجع تاني operand

Logical Assignment Operators ES 2021

```
const rest1 = {
    name: 'Capri',
    numGuests: 20,
};
const rest2 = {
    name: 'La Piazza',
    owner: 'Giovanni Rossi',
};
rest1.numGuests = rest1.numGuests || 10;
rest2.numGuests = rest2.numGuests || 10;
```

كده rest1 هيكون زي ما هوا، و rest2 هيتضاف 10 لـ prop جديدة اللي هي numGuests جواه

ممكن نعمل نفس السطر ده بدل التكرار كالاتي، (عامل زي augmented assign operator والحجات ده)

```
rest1.numGuests ||= 10;
rest2.numGuests ||= 10;
```

نفس الكلام تقدر تعمله مع ??, &&

Optional Chaining (?.) ES 2020 Feature

تم شرحه في سكشن ال Object، اضغط هنا

Statements and Expressions

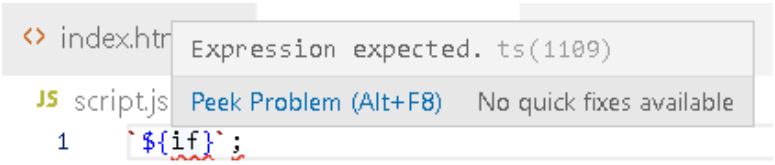
- **Expression**: is a piece of code that produces a value:

```
3+4 , 1991 , true && false && !false
```

- **Statement**: is a bigger piece of code that is executed to performs some action, but does not produce a value on itself.

```
var x = 3 ; // variable decleration  , if(23>10) const str = "23";
```

- طب ازاى افرق بينهم ؟ هنستخدم ال "${}" اللى موجود في "template literal" فهى بتاخد "expression" وليس "statement"



- An expression evaluates to a value, but a statement does something.

```
x + 2          # an expression
x = 1          # a statement
y = x + 1      # a statement
print y        # a statement (in 2.x)
```

Conditional Ternary Operator (To set a value to a variable, depending on condition)

```
condition ? exprIfTrue : exprIfFalse
```

- (مهم) طب هل "ternary operator" هتكون "statement" ولا "expression" ؟

أي "operator" هو "expression" عشان بيطلع قيمة والدليل على كده ان احنا ممكن نخزن الجملة اللى فوق في "variable"

```
const age = 23;
const drink = age >=18 ? console.log("wine") : console.log("milk");
```

هنا هنخزن ال output value اللي هيطلع من ternary operator في drink (عشان expression دايما produce a value)

وبرضه عشان نتأكد اكتر نعمل "test" بـ "template literal"

```
const age = 23;

console.log(`I like to drink ${age >= 18 ? 'wine' : 'milk'}`);
```

مينفعش اكتب return أو break او continue مع ال Ternary Operator لأن التلاتة دول statements واحنا زي ما قولنا ان ternary operator ده expression