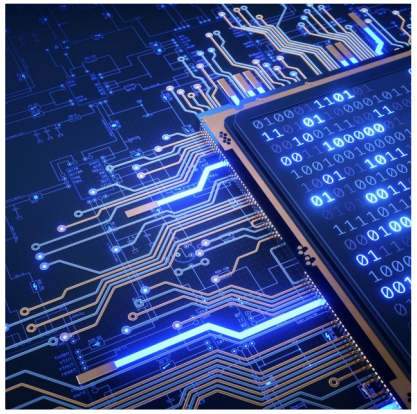
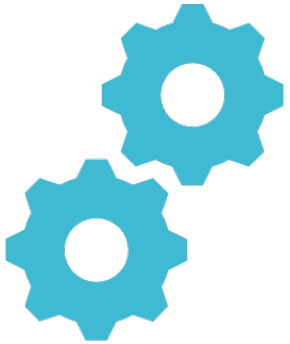
Basic Data Types and Operators



Data

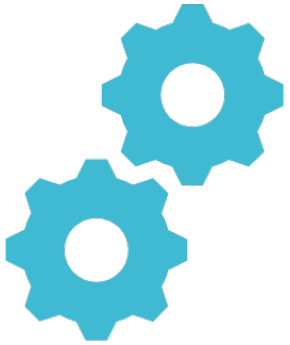


Processing over these data

Any computer program consists of

The first thing we need to learn about a programing language is how are the data represented and organized and what kind of processing we can apply on it

Data



Processing over these data

Any computer program consists of

Basic Data Types in JS

number

string

boolean

The data type determines two things:

- The range & format of the values that can be represented by this type
- The operations that can apply on these values

Basic **Data Types** in JS

number

string

boolean

Basic Data Types in JS

number

- All the following examples are considered of the same type
  - 13 (discrete)
  - 45.7 (fractional)
  - 2.998e8 (scientific = 2.998 * 10^8)

string

boolean

Basic Data Types in JS

number

Arithmetic Operators apply on values of type "number"

1. Minus/Negation (-)
   [unary operator, i.e. applies on one value]
2. Multiplication(*) – Division (/) – Remainder (%)
3. Subtraction (-) – Addition (+)

- The list above ordered based on the precedence
- We can use parentheses to avoid operator precedence confusion

(100 + 4) * 11 is not the same as 100 + 4 * 11

Data Types

number

Arithmetic Operators apply on values of type

%)

ce

*

Special Values

- Infinity
- -Infinity
- NaN (Not a Number)

5/0 = Infinity
Infinity – 1 = Infinity
Infinity – Infinity = NaN
0 – Infinity = -Infinity

Basic Data Types in JS

number

Arithmetic Operators

Special Values

- Infinity
- -Infinity
- NaN (Not a Numbe

5/0 = Infinity

Infinity – 1 = Infinity

Infinity – Infinity = N

0 – Infinity = -Infinity

The result of any operation that has NaN as an operand is always NaN, so we have to be careful because a mistake that results in NaN can silently propagate through the entire program

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> NaN + 5
NaN
> NaN - 5
NaN
> NaN / 5
NaN
> NaN * 5
NaN
>
```

Basic Data Types in JS

number

string

- Any literals surrounded by matching
  - Double quotes "Lie on the ocean"
  - Single quotes 'Float on the

boolean

Backticks `Down on the sea

- Similar to the string literals in Java or C++
- Having two types of quotes allows us to use one of the quotes inside the string without breaking it

number

string

- Any literals surrounded by matching
  - Double quotes "Lie on the ocean"
  - Single quotes 'Float on the
  - Backticks `Down on the sea`

boolean

- Similar to the string literals in Java or C++
- Having two types of quotes allows us to use one of the quotes inside the string without breaking it

numbe

string

- Any literals surrounded by matching
  - Double quotes "Lie on the ocean"
  - Single quotes 'Float on the

boolean

Backticks `Down on the sea

Basic Data Types in JS

number

string

- Any literals surrounded by matching
  - Double quotes "Lie on the ocean"
  - Single quotes 'Float on the

boolean

Backticks `Down on the sea

Basic Data Types in JS

number

string

There’s only one operator that applies on string values, which is , the concatenation operator(+)

"con" + 'cate' + `nate` = "concatenate"

- Similar to the string literals in Java or C++
- Having two types of quotes allows us to use one of the quotes inside the string without breaking it

number

string

- Any literals surrounded by matching
- Double quotes "Lie on the

“outer string ‘inner string’ ”
‘outer string “inner string” ’
Or we can use the escape character (\) just like in Java
“outer string \”outer string\” “

- \t is translated into a tab and
- \n is translated into new line and
- \\ will treat the backslash like a normal character not as an escape character

Ba
Ty

Strings surrounded by backticks are a bit different

- They can contain line breaks without breaking the string
- They can also contain expressions enclosed in ${...}

number

string

- Any literals surrounded by matching
  - Double quotes "Lie on the ocean"
  - Single quotes 'Float on the
  - Backticks `Down on the sea`

boolean

Ba
Ty

Strings surrounded by backticks are a bit different

- They can contain line breaks without breaking the string
- They can also contain expressions enclosed in ${...}

number

st

The following string is valid:

`first line
second line`

- unded by
- Double quotes "Lie on the ocean"
- Single quotes 'Float on the

boolean

Backticks `Down on the sea

Ba
Ty

Strings surrounded by backticks are a bit different

- They can contain a line breaks without breaking the string
- They can also contain expressions enclosed in ${...}

number

string

- A
n

`half of 100 is ${100 / 2}`
The expression will be evaluated, converted to a string then concatenated to produce `half of 100 is 50`

ocean

- Single quotes 'Float on the

Backticks `Down on the sea

boolean

Basic Data Types in JS

number

string

boolean

- Only two values are allowed
  - true or false

Basic Dat
Types in J

Logical Operators apply on values of type "boolean"

- Negation /Not (!) - Unary
- Logical AND (&&) , Logical OR (||) – Binary
- Conditional operator - Ternary (condition ? <value when true> : <value when false>)

boolean

- Only two values are allowed
  - true or false

Basic Dat
Types in J

Logical Operators apply on values of type "boolean"

- Negation /Not (!) - Unary
- Logical And (&&
- Conditional ope
  (condition ? val
  false)

Comparison operators produce boolean values

- >,<,<=,<=,==,!=
- === , !==

bool

- Only two values are allowed
  - true or false

Short-circuiting of logical ope

A	B	!A	A && B	A || B
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

- **<first expression> || <second expression >**
  - If the first expression is true the second expression will never be evaluated
- **<first expression> && <second expression>**
  - If the first expression is false the second expression will never be evaluated

Basic Data Types in JS

numbe

string

boolean

“typeof” operator allows us to know the type of a given value.

typeof 4.5
number

typeof “abc"
string

null
type: object

undefined
type:
undefined

Empty Values

Both of these values indicate non-existent or meaningless values

Type Conversion

Explicit Type Conversion

- You can explicitly convert a data value from one type to another using on of the following functions
  - Number()
  - String()
  - Boolean()

String to Number

```
> Number("5")
5
> Number("five")
NaN
> Number("")
0
```

Boolean to Number

```
> Number(true)
1
> Number(false)
0
>
```

Special values to Number

```
> Number(null)
0
> Number(undefined)
NaN
> Number(NaN)
NaN
```

Explicit Type Conve

Converting values to Number

You can explicitly convert a data value from one type to another

Number to String

```
> String(5)
'5'
> String(5) + String(6)
'56'
> Number("5") + Number(6)
11
>
```

Boolean to String

```
> String(true)
'true'
> String(false)
'false'
```

Special Values to String

```
> String(undefined)
'undefined'
> String(NaN)
'NaN'
> String(null)
'null'
```

Explicit Type Conve

Converting values to String

You can explicitly convert a data value from one type to another

Explicit Type Conversion

You can explicitly convert a data value from one type to another

Converting values to Boolean

undefined, NaN
- 0 (zero)
- “” (empty string)

Converted to

Anything else is converted to true

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> Boolean(null)
false
> Boolean(NaN)
false
> Boolean(undefined)
false
> Boolean(0)
false
> Boolean("")
false
> Boolean("abc")
true
> Boolean(5)
true
>
```

Automatic type conversion (*type coercion*)

When an operation applies on values of different types, the JS engine will try to convert one of them in order to produce a value

Arithmetic Operations

- Since arithmetic operations are supposed to be applied on numbers, JS will call Number() function on non-numeric values

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> 8 * null
0
> 5 + null
5
> null - 5
-5
> null / 5
0
>
```

Reminder:
- Number(null) returns 0
- Number(undefined) returns NaN

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> undefined + 5
NaN
> undefined / 5
NaN
> undefined - 5
NaN
> undefined * 5
NaN
>
```

Applied on special values

Arithmetic Operations

Since arithmetic operations are supposed to be applied on numbers, JS will call Number() function on non-numeric values

Arithmetic Operations

Since arithmetic operations are supposed to be applied on numbers, JS will call Number() function on non-numeric values

Applied on Strings

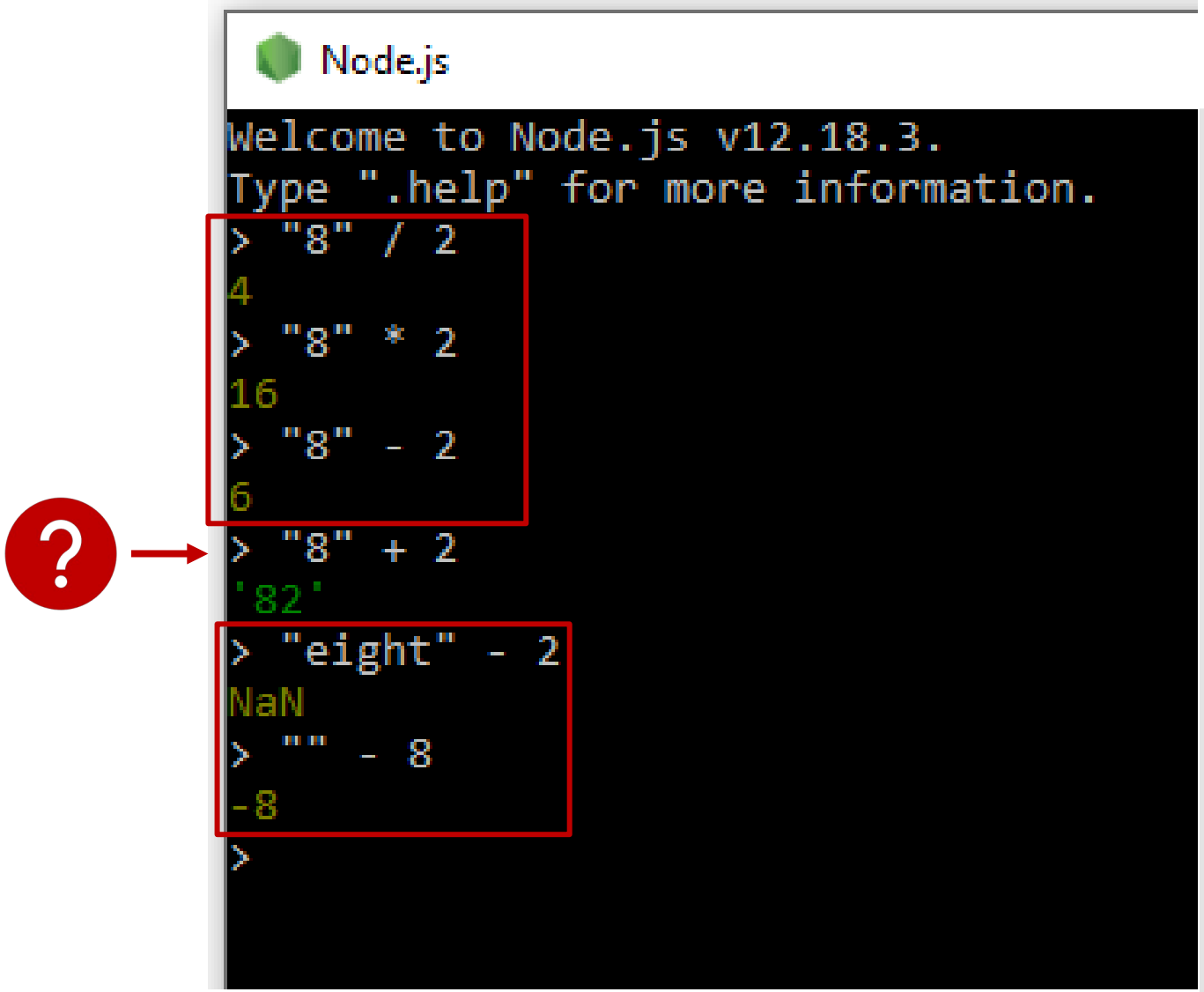
Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> "8" / 2
4
> "8" * 2
16
> "8" - 2
6
> "8" + 2
'82'
> "eight" - 2
NaN
> "" - 8
-8
>
```

Arithmetic Operations

Since arithmetic operations are supposed to be applied on numbers, JS will call Number() function on non-numeric values

Applied on Strings



Arithmetic Operations

Since arithmetic operations are supposed to be applied on numbers, JS will call Number() function on non-numeric values

Applied on Strings

operator is applied on a string it's always interpreted as **concatenati on** not ac

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> "8" / 2
4
> "8" * 2
16
> "8" - 2
6
> "8" + 2
'82'
> "eight" - 2
NaN
> "" - 8
-8
>
```

Arithmetic Operations

Since arithmetic operations are supposed to be applied on numbers, JS will call Number() function on non-numeric values

Applied on Boolean

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> false + 5
5
> false - 5
-5
> false / 5
0
> false * 5
0
> true + 5
6
> true - 5
-4
> true / 5
0.2
> true * 5
5
>
```

Logical Operators

Since logical operators are supposed to apply on Boolean values, JS will call Boolean() function on non-Boolean values when it applies a logical operator

Logical Operators

Since logical operators are supposed to apply on Boolean values, JS will call Boolean() function on non-Boolean values when it applies a logical operator

Conversion rules

- null , zero , NaN, empty string, and undefined are considered **false**
- Anything else is considered **true**

The OR (||) operator

(can be used to create fallback mechanism for null values)

The first operand is evaluated first:

- If the value of the first operand can be converted to true,
  - it will return as the value of the entire logical expression
  - The s

```
> (6 + 5) || (5 + 9)
11
```

uated

- If the value of the first operand can be converted to false
  - The sec

```
> !(6 + 5) || (5 + 9)
14
```

ated, and its value will retu re logical expression

Logical Operators

Since logical operators are supposed to apply on Boolean values, JS will call Boolean() function on non-Boolean values when it applies a logical operator

Conversion rules

- null , zero , NaN, empty string, and undefined are considered **false**
- Anything else is considered **true**

The AND (&&) operator

The first operand is evaluated

- If the value of the 1st operand can be converted to false,
  - It will return as the value of the logical expression
  - The second operand will never be evaluated

```
> !(6 + 5) && (5 + 9)
false
```

- If the value of the 1st operand can be converted to true
  - The second operand will be evaluated, and its value will return as the value of the logical expression

```
> (6 + 5) && (5 + 9)
14
```

Comparison operators

null value with comparison operators

When used with comparison operators the null value is generally considered as zero except for the equality and inequality operators

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> null > -1
true
> null < 1
true
> null == 0
false
> null != 0
true
> null >= 0
true
> null <= 0
true
>
```

null value with comparison operators

When used with comparison operators the null value is generally considered as zero except for the equality and inequality operators

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> null > -1
true
> null < 1
true
> null == 0
false
> null != 0
true
> null >= 0
true
> null <= 0
true
>
```

null is only equal to another null or **undefined**

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more informatic
> null == 0
false
> null == null
true
> null == undefined
true
>
```

null value with comparison operators

When used with comparison operators the null value is generally considered as zero except for the equality and inequality operators

It's good practice to apply fallback mechanism to replace null values to avoid these problems

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> null > -1
true
> null < 1
true
> null == 0
false
> null != 0
true
> null >= 0
true
> null <= 0
true
>
```

null is only equal to another null or **undefined**

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more informatic
> null == 0
false
> null == null
true
> null == undefined
true
>
```

null value with comparison operators

When used with comparison operators the null value is generally considered as zero except for the equality and inequality operators

```
> null || 5
5
> null || 'value not found'
'value not found'
>
```

It's good practice to apply fallback mechanism to replace null values to avoid these problems

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> null > -1
true
> null < 1
true
> null == 0
false
> null != 0
true
> null >= 0
true
> null <= 0
true
>
```

null is only equal to another null or **undefined**

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more informatic
> null == 0
false
> null == null
true
> null == undefined
true
>
```

Strings with comparison operators

Generally when you compare a string to another string, the comparison happens between the corresponding unicode values of the characters

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> "abc" < "ABC"
false
> "abc" > "ABC"
true
```

```
> console.log('H'.charCodeAt(0) > 3);
true
```

Strings with comparison operators

But when compared to numeric values JS will attempt to convert it to a number if possible

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> "8" > 7
true
> "8" < 9
true
> "8" == 8
true
> "8" != 8
false
> "8" === 8
false
> "8" !== 8
true
> "8" >= 8
true
> "8" > 8 || "8" === 8
false
> "8" <= 8
true
> "8" < 8 || "8" === 8
false
>
```

Strings with comparison operators

But when compared to numeric values JS will attempt to convert it to a number if possible

The conversion can be avoided by using the three-character comparison operators (sometimes called the exact comparison operators)

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> "8" > 7
true
> "8" < 9
true
> "8" == 8
true
> "8" != 8
false
> "8" === 8
false
> "8" !== 8
true
> "8" >= 8
true
> "8" > 8 || "8" === 8
false
> "8" <= 8
true
> "8" < 8 || "8" === 8
false
>
```

Booleans with comparison operators

- **false** is converted to number zero
- **true** is converted to the number 1
- The conversion can be avoided using the three-character (exact) comparison operators

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> false > -1
true
> false < 1
true
> false == 0
true
> false != 0
false
> true > 1
false
> true < 2
true
> true == 1
true
> true != 1
false
> false === 0
false
> false !== 0
true
> true === 1
false
> true !== 1
true
>
```

Bindings

Binding

```
let caught = 5 * 5;
```

- This statement creates a binding called “caught” and uses it to grab hold of the value produced by multiplying 5 by 5.
- After a binding has been defined, its name can be used as ***an expression***.
  - The value of such an expression is the value the binding currently holds.

```
console.log(caught);
```

- The assignment (=) operator can be used at any time on existing bindings to disconnect them from their current value and have them point to a new one.

  - caught = 10;

Binding

A single (let) statement may define multiple bindings. The definitions must be separated by commas.

```
let one = 1, two = 2;
console.log(one + two);
// → 3
```

Binding using var & const

```
var name = "Ayda";
const greeting = "Hello ";
console.log(greeting + name);
// → Hello Ayda
```

- **var** stands for variable, similar to **let** with some differences (we'll talk about them latter)
  - is the way bindings were declared in pre-2015 JavaScript
- **const** stands for constant, it defines bindings that cannot be changed

JavaScript is a weakly typed language

In a weakly typed language, variables are not given a specific type

Node.js

```
Welcome to Node.js v12.18.3.
Type ".help" for more information.
> let x = 5;
undefined
> typeof x;
'number'
> console.log(x);
5
undefined
> x = "abc";
'abc'
> typeof x;
'string'
> console.log(x);
abc
undefined
> x = false;
false
> typeof x;
'boolean'
> console.log(x);
false
undefined
>
```

Binding names (Identifiers)

- Binding names can be any word that is not a reserved keyword (e.g. **let**)
- Digits can be part of binding names, but the name must not start with a digit.
- A binding name may include dollar signs ($) or underscores (_) but no other special characters.