Javascript

Autres objets intégrés

Date

Syntaxe

```
new Date();
new Date(valeur);
new Date(chaîneDate);
new Date(année, mois[, jour[, heures[, minutes[, secondes[,
millisecondes]]]]]);
```

`valeur` : nombre de millisecondes depuis le 1 Janvier 1970
`chaineDate` : chaîne conforme à l'ISO8601

Exemples

```
new Date(); //date courante
new Date(1427241600000);
new Date('2015-03-25');
new Date('2015-03-25T15:36');
new Date(2015, 02, 25); //0 => janvier, ..., 11 => décembre
new Date(2015, 02, 25, 15, 36);
```

Pas de notation littérale

Date

L'API n'est ni très fournie, ni très pratique.
⇨ Référence complète

Bibliothèque conseillées :

- Day.js
- Luxon (gestion des fuseaux horaires)

Bientôt : Temporal

Math

L'objet Math est un objet natif dont les méthodes et propriétés permettent l'utilisation de constantes et fonctions mathématiques.

L'objet Math n'est pas un constructeur. Toutes les propriétés et les méthodes sont statiques.

Exemple

```
Math.round(10.2546); //10
```

Math

Propriétés

- `E` : nombre d'Euler
- `LN2` : logarithme naturel de 2
- `LN10` : logarithme naturel de 10
- `LOG2E` : logarithme de base 2 de E
- `LOG10E` : logarithme de base 2 de E
- `PI` : circonférence / diamètre d'un cercle
- `SQRT1_2` : racine carrée de 1/2
- `SQRT2` : racine carrée de 2

Math
Méthodes 1/3

- `abs` : valeur absolue d'un nombre
- `acos` : arc cosinus d'un nombre
- `asin` : arc sinus d'un nombre
- `atan` : arc tangente d'un nombre
- `atan2` : arc tangente du quotient de ses arguments
- `ceil` : plus petit entier supérieur ou égal à la valeur
- `cos` : cosinus d'un nombre
- `exp` : exponentielle d'un nombre
- `floor` : plus grand entier inférieur ou égal à la valeur

Math
Méthodes 2/3

- `log` : logarithme naturel d'un nombre
- `max` : plus grande valeur d'une liste de nombres
- `min` : plus petite valeur d'une liste de nombres
- `pow` : calcul de x à la puissance y
- `random` : nombre aléatoire entre 0 inclus et 1 exclu
- `round` : arrondi d'un nombre
- `sin` : sinus d'un nombre
- `sqrt` : racine carrée d'un nombre
- `tan` : tangente d'un nombre

Math
Méthodes 3/3 (ES6)

- `sign` : signe d'un nombre
- `trunc` : troncature
- `cbrt` : racine cubique
- `expm1` : renvoie Math.exp(x)-1
- `log1p` : renvoie Math.log(1 + x)
- `log2` : logarithme en base 2
- `log10` : logarithme en base 10
- `sinh` : sinus hyperbolique
- `cosh` : cosinus hyperbolique
- `tanh` : tangente hyperbolique
- `asinh` : arc sinus hyperbolique
- `acosh` : arc cosinus hyperbolique
- `atanh` : arc tangente hyperbolique

Expressions rationnelles

Motifs utilisés pour correspondre à certaines combinaisons de caractères au sein de chaînes de caractères.
En javascript, les expressions rationnelles sont également des objets.

Expressions rationnelles

Création

Constructeur

```
let regToto = new RegExp('toto','i');
```

Notation littérale

```
let regToto = /toto/i;
```

Le deuxième argument est optionnel, il précise un mode de recherche (dans notre exemple, `i` permet d'ignorer la casse).

Expressions rationnelles

Quelques exemples

```
//teste si la chaîne se termine par "toto" en ignorant la casse
/toto$/i.test("MonNomEstToto"); //true

//remplace toutes les occurences de "to" par "ta" en ignorant la casse
"MonNomEstToto".replace(/to/ig,'ta'); //"MonNomEsttata"

//renvoie les chaînes et sous-chaînes
let matches = /^MonNomEst(\w+)$/i.exec("MonNomEstToto");
matches[0]; //"MonNomEstToto"
matches[1]; //"Toto" (entre parenthèses capturantes)
```

Liens :

⇨ Introduction aux expressions rationnelles

⇨ Référence complète de l'objet RegExp

JSON

L'objet JSON n'est pas un constructeur. Il ne possède que deux méthodes statiques :

- `parse` : convertit une chaîne de caractères JSON en objet.

```
let toto = JSON.parse('{"nom":"Toto","age":25}');
toto.age; //25
```

- `stringify` : convertit un objet en chaîne JSON.

```
let str = JSON.stringify({"nom":"Toto","age":25});
//'{"nom":"Toto","age":25}'
```

Error

Constructeur d'objets représentant une erreur.

Des instances d'objets Error sont levées lorsqu'une erreur d'exécution survient.

```
try { JSON.parse("(toto)"); }
catch(e) { e instanceof Error; /* true */ }
```

L'objet Error peut aussi être utilisé pour représenter des erreurs définies par l'utilisateur.

```
function maFct(nb) {
  if (typeof nb != "number") {
    throw new Error("L'argument doit être un nombre");
  }
}
```

Types d'erreur

Il existe six constructeurs d'erreurs dérivés de Error :

- `EvalError`
- `RangeError`
- `ReferenceError`
- `SyntaxError`
- `TypeError`
- `URIError`

```
try { JSON.parse("(toto)"); }
catch(e) {
  e instanceof Error; //true
  e instanceof SyntaxError; //true
  e.constructor; //function SyntaxError
  e.message; //"JSON.parse: unexpected character
  //at line 1 column 1 of the JSON data"
}
```