Working With the Date Type

In this lesson, we'll cover the date type in JavaScript.
Let's begin!

WE'LL COVER THE FOLLOWING

- Creating a date variable
  - Examples

JavaScript specifies `Boolean`, `Number`, and `String` as primitive value types. Interestingly, it does not use a primitive value type for handling date and time values.

Instead,

---

*the standard defines the **date** type (reference type) that stores dates as the number of milliseconds that have passed since midnight on January 1, 1970 UTC (Universal Time Code).*

Leveraging this data storage format, the `Date` type can accurately represent dates almost *300,000* years before or after January 1, 1970.

Creating a date variable

To create a date, use the `Date()` constructor function, and pass the number representation as an argument. When the argument is omitted, the new object is assigned the current date and time:

```
var now = new Date();
var y2001 = new Date(978303600000);
console.log(now.toLocaleString());
console.log(y2001.toLocaleString());
```

The console output shows something like this:

console

```
//today's time and date,
//yours will be different according to current time and date of your locale
11/29/2019, 6:28:34 AM
1/1/2001 00:00:00
```

The first line tells you when this script was run before inserting the output into the manuscript of the course. The second line shows that the strange `978303600000` number means the first moment of January 1, 2001.

Using magic numbers to create a Date instance is not the easiest. The `Date.parse()` and `Date.UTC()` methods provide a better way; both return a number that represents the date value.

`Date.parse()` accepts a string and supports these formats:

```
ISO 8601 extended format: YYYY-MM-DDTHH:mm:ss.sssZ (e.g. 2008-06-12T23:34:02).
month/date/year (e.g. 12/24/2008)
month-name date, year (e.g. October 22, 1996)

day-of-week month-name date year hours:minutes:seconds time-zone
```

```
(e.g. Sun November 24 2013 12:38:17 GMT+0100)
```

With `Date.parse()`, you can forget about the magic number `978303600000`, and write this more naturally:

```
var y2001 = new Date(
    Date.parse("01/01/2001"));
```

The `Date.UTC()` function accepts many number parameters: `year`, `zero-based month` (January is 0, February is 1, etc.), day of the month (1-31), `hour` (0-23), `minutes`, `seconds`, and `milliseconds`.

The year and month parameters are required, all others are optional. The default day of the month value is 1, all other parameters default to 0.

Examples

Let's see a few examples:

```
var num = Date.UTC(2013, 2, 18, 6, 44);
console.log(new Date(num));
num = Date.UTC(2001, 1, 31);
console.log(new Date(num));
```

When you examine the console output, you can observe a few surprising things (lines are broken intentionally):

console

```
Mon Mar 18 2013 07:44:00 GMT+0100
  (Central Europe Standard Time)
Sat Mar 03 2001 01:00:00 GMT+0100
  (Central Europe Standard Time)
```

The first `Date` construction specified `6:44` as the time. Because `Date.UTC()` constructs number value for UTC time, it has been modified to `07:44:00 GMT+0001`.

This code was run in Hungary, which is located in the GMT+1 time zone.

The sample tried to use an invalid date in the second `Date` value: `“February 31, 2001”`. Instead of declaring this date invalid, the `Date.UTC()` method calculated March 3 by correcting February 31 as the third day after February 28.

To make using the `Date` construction easier, you can invoke the date constructor to mimic the `Date.parse()` and `Date.UTC()` functions:

```
var y2001 = new Date("01/01/2001");
var date = new Date(2013, 2, 18, 6, 44);
console.log(y2001);
console.log(date);
```

However, the console output shows that the second way of `Date` construction uses locale time in contrast to `Date.UTC()`, which calculates UTC time, because 6:44 is calculated as 6:44 GMT+1, instead of 7:44 GMT+1, as with `Date.UTC()`:

console

```
Mon Jan 01 2001 00:00:00 GMT+0100
  (Central Europe Standard Time)
Mon Mar 18 2013 06:44:00 GMT+0100
  (Central Europe Standard Time)
```

You can use dozens of methods with the date type. You can access the date and time parts with methods such as `getDate()`, `getTime()`, `getYear()`, `getMonth()`, `getDay()`, `getFullYear()`, `getUTCMonth()`, and many more.

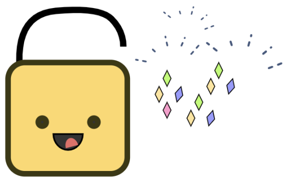
Similarly, you can use a number of methods to set different parts of the Date instance, such as `setDate()`, `setTime()`, `setFullyear()`, and `setMonth()` among the others.

You can use date formatting methods, such as `toDateString()`, `toTimeString()`, `toLocalDateString()`, `toLocalTimeString()`, `toUTCString()`.

NOTE: For a complete list of available date methods, see the date type

> reference on MDN. When you navigate to the reference page, you'll see the methods' reference links as `Date.prototype.methodname()` , for example, `Date.prototype.getDate()` . In the next section you will get a short overview about object prototypes, and you'll understand this notation.

Achievement unlocked! 🎉



Congratulations! You've learned how to deal with the date types in JavaScript.

Great work! Give yourself a round of applause! :)

In the *next lesson*, we'll learn about the object type.