

---

## Instadate build unknown

A minimal high performance date library for Node.js and Browser. Use it to compare and manipulate dates.

### Installation

```
1 npm install instadate
```

### Usage

#### ES6

```
1 import instadate from "instadate";
2
3 const date1 = new Date();
4 const date2 = instadate.addDays(date1, 1);
5
6 instadate.differenceInDates(date1, date2); // => 1
```

#### ES5

```
1 var instadate = require("instadate");
2
3 var date1 = new Date();
4 var date2 = instadate.addDays(date1, 1);
5
6 instadate.differenceInDates(date1, date2); // => 1
```

### Motivation behind Instadate

Current popular date libraries put a lot of effort into doing a lot causing them to lose a lot of performance because of heavy abstractions. Instadate on the other hand only has a handful of features that are all geared towards performance. Instadate is more of a wrapper around the native JavaScript Date than a full on date library.

Use Instadate when you need to run thousands of date manipulations or comparisons per second. Instadate is managed and used by Teamweek team calendar.

**Instadate..**

- 
- is **fast** (10 - 1000 times faster than moment)
  - is **small** (just look at the source)
  - is **immutable** (Instadate will always return new date objects and never modify the given ones)

## API Reference

### **utc(date)**

Returns the UTC time in milliseconds.

### **noon(date)**

Returns a copy of the date with hours set to 12 and minutes, seconds and milliseconds set to 0.

### **differenceInDays(from, to)**

Returns the difference in days (24 hour periods) between two dates. Returned result can be negative.

### **differenceInHours(from, to)**

Returns the difference in hours (60 minute periods) between two dates. Returned result can be negative.

### **differenceInMinutes(from, to)**

Returns the difference in minutes (60 second periods) between two dates. Returned result can be negative.

### **differenceInSeconds(from, to)**

Returns the difference in seconds (1000 millisecond periods) between two dates. Returned result can be negative.

### **differenceInWeekendDays(from, to)**

Returns the difference in days (24 hour periods) between two dates. Excludes work days. Returned result can be negative.

### **differenceInWorkDays(from, to)**

Returns the difference in days (24 hour periods) between two dates. Excludes weekend days. Returned result can be negative.

### **differenceInDates(from, to)**

Similar to `differenceInDays` however counts all dates that fit into the period, not 24 hour periods.

---

**addYears(date, years)**

Returns a cloned date with years added to it, use negative input for subtraction.

**addMonths(date, months)**

Returns a cloned date with months added to it, use negative input for subtraction.

**addDays(date, days)**

Returns a cloned date with days added to it, use negative input for subtraction.

**addHours(date, hours)**

Returns a cloned date with hours added to it, use negative input for subtraction.

**addMinutes(date, minutes)**

Returns a cloned date with minutes added to it, use negative input for subtraction.

**addSeconds(date, seconds)**

Returns a cloned date with seconds added to it, use negative input for subtraction.

**addMilliseconds(date, milliseconds)**

Returns a cloned date with milliseconds added to it, use negative input for subtraction.

**isSameYear(date1, date2)**

Returns if the years are equal

**isSameMonth(date1, date2)**

Returns if the years and months are equal

**isSameDay(date1, date2)**

Returns if the years, months and dates are equal

**equal(date1, date2)**

Returns if the dates are equal to millisecond precision

**min(date1, date2)**

Returns the earliest date

**max(date1, date2)**

Returns the latest date

**dates(start, end)**

Returns all dates within start and end period. Uses `differenceInDates` internally.

---

### **dateString(date)**

Returns the date part of a Date as a string. Example: Mon Jan 25 2016.

### **isoDateString(date)**

Returns the date part of an iso Date as a string. Example: 2016-01-25.

### **parseISOString(isoString)**

Parses an ISO string to a date. If used in a browser environment also resets the timezone offset using `resetTimezoneOffset`.

```
1 const date = instadate.parseISOString("2017-04-24");
2 // Sun Apr 24 2017 00:00:00 GMT-0700 (PDT)
```

### **resetTimezoneOffset(date)**

Resets the timezone offset caused by browsers when parsing a date.

```
1 let date = new Date("2017-04-24");
2 // Sun Apr 23 2017 17:00:00 GMT-0700 (PDT)
3 // Note how the parsed date is 23, not 24
4 // This is caused by browsers and does not happen in node
5
6 date = instadate.resetTimezoneOffset(date);
7 // Sun Apr 24 2017 00:00:00 GMT-0700 (PDT)
```

### **firstDateInMonth(date)**

Returns the first date of the month.

### **lastDateInMonth(date)**

Returns the last date of the month.

### **isWeekendDay(day)**

Checks if the given day (0-6) is a weekend day.

### **isWorkDay(day)**

Checks if the given day (0-6) is a work day.

### **isWeekendDate(date)**

Checks if the given date is a weekend day.

### **isWorkDate(date)**

Checks if the given date is a work day.

### **setWeekendDays(daysArray)**

---

Sets the given days (array containing integers from 0 to 6) as weekend days. Any day not in the array will be set as a work day. After the weekend days are set all Instadate functions will use the new work and weekend days.

**daysInPeriod(firstDay, length, daysArray)**

Counts the days defined in daysArray within the given period.

- firstDay - day to start counting from (0-6)
- length - the total of days to count (can be negative)
- daysArray - array of days to count (if you want to count Mondays and Tuesdays use [1, 2])

**weekendDaysInPeriod(firstDay, length)**

Counts weekend days within period.

**workDaysInPeriod(firstDay, length)**

Counts work days within period.

**isAfter(date1, date2)** Checks if date1 is after date2.

**isYearAfter(date1, date2)**

Checks if date1 year is after date2 year.

**isMonthAfter(date1, date2)**

Checks if date1 month is after date2 month.

**isDayAfter(date1, date2)**

Checks if date1 date is after date2 date.

**isBefore(date1, date2)**

Checks if date1 is before date2.

**isYearBefore(date1, date2)**

Checks if date1 year is before date2 year.

**isMonthBefore(date1, date2)**

Checks if date1 month is before date2 month.

**isDayBefore(date1, date2)**

Checks if date1 date is before date2 date.

**isYearBetween(date1, date2, date3)**

Checks if date1 year is between date2 and date3 years.

---

### **isMonthBetween(date1, date2, date3)**

Checks if date1 month is between date2 and date3 months.

### **isDayBetween(date1, date2, date3)**

Checks if date1 is between date2 and date3.

### **isoWeekDay(day)**

Converts day (0-6) to iso day (1-7).

### **daysInMonth(month, year)**

Returns how many days are in a month.

```
1 let date = new Date();
2 instadate.daysInMonth(date.getMonth(), date.getFullYear()); //28, 29,
   30 or 31
```

## **Testing**

```
1 npm test
```

## **License**

MIT

## **Changelog**

- 0.6.0 - Added `daysInMonth` function.
- 0.5.0 - Changed `isoDateString` to work correctly with UTC+13, UTC+14 & UTC-12. Removed `dateString` function.
- 0.4.0 - Added `parseISOString` and `resetTimezoneOffset` functions
- 0.3.2 - Performance optimisations. Subtract dates instead of comparing them.
- 0.3.1 - Modified `addDays`, `addHours`, `addMinutes`, `addSeconds` and `addMilliseconds` functions to work with dayling saving changes.