자료구조와 자료형

Date 객체와 날짜

- Date => 날짜를 저장할 수 있고, 날짜와 관련된 메서드도 제공해주는 내장 객체
- 인수 없이 new 연산자와 함께 호출하면 현재 날짜와 시간이 저장된 Date 객체가 반환

```
let now = new Date();
// 현재 날짜 및 시간이 출력됨
alert( now ); // Sat Apr 17 2021 17:07:27 GMT+0900 (대한민국 표준시)
```

```
new Date(milliseconds)
```

- **milliseconds 단위의 유닉스 시간**을 나타내는 정수를 반환
  - 타임스탬프(timestamp) => 1970년의 첫날(1970년 1월 1일 0시 0분 0초)을 기준으로 흘러간 밀리초를 나타내는 정수
    - 타임스탬프 => 유닉스 타임스탬프, 유닉스 시간, POSIX 시간, epoch 등등 다양한 이름으로 불리움

```
// 1970년 1월 1일 0시 0분 0초(UTC+0)를 나타내는 객체
let Jan01_1970 = new Date(0);
alert( Jan01_1970 );

// 1970년 1월 1일의 24시간 후는 1970년 1월 2일(UTC+0)임
let Jan02_1970 = new Date(24 * 3600 * 1000);
alert( Jan02_1970 );
```

- 1970년 1월 1일 이전 날짜에 해당하는 타임스탬프 값은 음수

```
// 1970년 1월 1일의 24시간 전 (음수 전달)
let Dec31_1969 = new Date(-24 * 3600 * 1000);
alert( Dec31_1969 );
```

- 인수가 하나이며 날짜 정보가 담긴 문자열을 전달했다면, 해당 **문자열은 자동으로 구문 분석(parsed)** 되어 날짜 객체로 변환됨

```
new Date(datestring)
```

- 전달한 문자열은 GMT(UTC) 시간이라고 가정되며, 출력을 할 때에는 **현재 OS의 시간대의 offset이 더해진 시간이 출력**됨
  - 대한민국 표준시(KST)는 **UTC 시간에 9시간 더한 시간**
  - UTC와 Timezone 개념

```
// 날짜와 관련된 문자열을 생성자 함수로 전달
let date = new Date("2017-01-26");
alert(date);

// 인수로 시간은 지정하지 않았기 때문에 "GMT" 자정이라고 가정하고 코드가 실행되는 시간대(timezone)에 따라 출력 문자열이 바뀜
// 따라서 얼럿 창엔 한국이라면,
// "Thu Jan 26 2017 09:00:00 GMT+0900 (대한민국 표준시)"가 출력
// 어느 지역의 OS에서 출력하느냐에 따라서,
// "Thu Jan 26 2017 11:00:00 GMT+1100 (Australian Eastern Daylight Time)"
// 혹은
// "Wed Jan 25 2017 16:00:00 GMT-0800 (Pacific Standard Time)"등이 출력됩니다.
```

- 날짜와 관련된 숫자 정보를 직접 전달하여 날짜 객체 생성 가능 (단, UTC가 아닌, 지역 시간대의 객체가 생성됨을 유의)

```
new Date(year, month, date, hours, minutes, seconds, ms)
```

- year는 반드시 네 자리 숫자 => 2013 (O), 98 (X)
- month는 0(1월)부터 11(12월) 사이의 숫자
- date는 일을 나타내는데, 값이 없는 경우엔 1일로 처리됨
- hours/minutes/seconds/ms에는 각각 시, 분, 초, 1/1000초를 나타내며, 값이 없는 경우엔 0으로 처리됨

```
// 2011년 1월 1일, 0시 0분 0초
let d1 = new Date(2011, 0, 1, 0, 0, 0, 0);
// Sat Jan 01 2011 00:00:00 GMT+0900 (대한민국 표준시)
alert( d1 );

// hours를 비롯한 인수는 기본값이 0이므로 위의 날짜 객체와 동일한 내용의 날짜 객체 생성
let d2 = new Date(2011, 0, 1);
```

- 만약 지역 시간대가 아닌, **UTC 기준 시간으로 날짜 객체를 생성**해야 할 경우 Date.UTC를 사용
    - Date.UTC는 UTC를 기준으로한 타임스탬프 값을 반환

```
let arg = Date.UTC(2011, 0, 1, 0, 0, 0, 0);
alert( arg ); // 1293840000000

let d = new Date(arg);
// Sat Jan 01 2011 09:00:00 GMT+0900 (대한민국 표준시)
alert( d );
```

날짜 구성요소 얻기

```
getFullYear() : 연도(네 자릿수)를 반환합니다.
getMonth() : 월을 반환합니다(0 이상 11 이하).
getDate() : 일을 반환합니다(1 이상 31 이하). 어! 그런데 메서드 이름이 뭔가 이상하네요.
getHours(), getMinutes(), getSeconds(), getMilliseconds() : 시, 분, 초, 밀리초를 반환합니다.
getDay() : 일요일을 나타내는 0부터 토요일을 나타내는 6까지의 숫자 중 하나를 반환합니다.
```

- 위의 메서드들은 모두는 지역 시간 기준 날짜 구성요소를 반환
- 위 메서드 이름에 있는 **get 다음에 "UTC"를 붙여주면 표준시(UTC+0) 기준의 날짜 구성 요소를 반환**해주는 메서드인 getUTCFullYear(), getUTCMonth(), getUTCDay()등에 접근 가능

```
// 현재 일시
let date = new Date();

// 현지 시간 기준 시
alert( date.getHours() );
// 표준시간대(UTC+0, 일광 절약 시간제를 적용하지 않은 런던 시간) 기준 시
alert( date.getUTCHours() );
```

```
getTime() : 주어진 일시와 1970년 1월 1일 00시 00분 00초 사이의 간격(밀리초 단위)인 타임스탬프를 반환합니다.
getTimezoneOffset() : 현지 시간과 표준 시간의 차이(분)를 반환합니다.
```

- 위의 메서드는 시간대 정보와 무관한 시간 정보를 반환함

```
// 타임스탬프값 반환
alert( new Date().getTime() );

// UTC-1 시간대에서 이 예시를 실행하면 60이 출력
// UTC+3 시간대에서 이 예시를 실행하면 -180이 출력
// 한국(UTC+9) 시간대에서는 -540이 출력 (UTC시간 +9시간이므로, -60*9로 계산)
alert( new Date().getTimezoneOffset() );
```

날짜 구성요소 설정하기

```
setFullYear(year, [month], [date])
setMonth(month, [date])
setDate(date)
setHours(hour, [min], [sec], [ms])
setMinutes(min, [sec], [ms])
setSeconds(sec, [ms])
setMilliseconds(ms)
setTime(milliseconds) (1970년 1월 1일 00:00:00 UTC부터 밀리초 이후를 나타내는 날짜를 설정)
```

```
let today = new Date();

today.setHours(0);
// 날짜는 변경되지 않고 시만 0으로 변경
alert(today);

today.setHours(0, 0, 0, 0);
// 날짜는 변경되지 않고 시, 분, 초가 모두 변경 (00시 00분 00초)
alert(today);
```

Date 객체를 숫자로 변경해 시간차 측정하기

- Date 객체를 숫자형으로 변경하면 타임스탬프 반환

```
let date = new Date();
// date.getTime()를 호출한 것과 동일한 값 반환
alert(+date);
```

- 벤치마크 작업 시 사용 가능

```
// 측정 시작
let start = new Date();

// 시간이 걸리는, 원하는 작업을 수행
for(let i = 0; i < 100000; i++) {
    let doSomething = i * i * i;
}

// 측정 종료
let end = new Date();

// 빼기 이항 연산자(-)를 사용하므로 숫자형으로 형 변환이 일어나고 결과적으로 타임스탬프를 뺀 값이 됨
alert( `작업 완료까지 ${end - start} 밀리초가 걸렸습니다.` );
```

Date.now()

- Date 객체를 만들지 않고도 시차를 측정하기 위해서 현재 타임스탬프를 반환하는 메서드 Date.now 사용 가능
    - Date.now는 `new Date().getTime()`과 동일한 결과를 반환하지만, **중간에 Date 객체를 만들지 않는다는 장점이 있음**

```
// 측정 시작
let start = Date.now();

// 시간이 걸리는, 원하는 작업을 수행
for(let i = 0; i < 100000; i++) {
    let doSomething = i * i * i;
}

// 측정 종료
let end = Date.now();

alert( `작업 완료까지 ${end - start} 밀리초가 걸렸습니다.` );
```

Date.parse와 문자열

- Date.parse(str) 메서드를 사용하면 **문자열에서 타임스탬프 값을 반환**받을 수 있음 (날짜 객체를 반환받는게 아님을 유의)
    - 단, 문자열의 형식은 YYYY-MM-DDTHH:mm:ss.sssZ와 같은 형식(ISO8601)으로 전달
        - yyyy-MM-dd'T'hh:mm:ss.sss'Z' (UTC)

        - yyyy-MM-dd'T'hh:mm:ss.sss**[+|-]hh:mm** (다른 시간대면 +, - 기호와 시간 전달)
    - YYYY-MM-DD, YYYY-MM, YYYY같이 더 짧은 문자열 형식도 가능
    - 문자열의 형식이 조건에 맞지 않은 경우엔 **NaN이 반환**됨

```
// UTC 시간에서 -7시간인 Timezone 시간
let ms1 = Date.parse('2012-01-26T13:51:50.417-07:00');
alert(ms1); // 1327611110417

let ms2 = Date.parse('2012-01-26');
alert(ms2); // 1327536000000
```

- parse 메소드 호출로 반환받은 타임스탬프를 이용해서 Date 객체를 만들 수 있음

```
let date = new Date( Date.parse('2012-01-26T13:51:50.417-07:00') );
alert(date);
```

- 보통 Date 객체를 그대로 쓰기보다는, 날짜 관련 라이브러리 중 가장 많이 사용되는 momentjs 라이브러리를 사용하여 날짜 관련 작업 처리를 진행