Data Type

☑ Data Type

JavaScript에서 값은 항상 string이나 number 같은 특정한 data type에 속한다.
JavaScript에는 8가지 기본 data type이 있다. 여기에서는 이 data type 모두를 개괄적으로 다루도록 한다.

JavaScript의 변수는 data type에 관계없이 모든 데이터일 수 있다. 따라서 변수는 어떤 순간에 string일 수 있고 다른 순간엔 number가 될 수도 있다.

```
// no error
let message = "hello";
message = 123456;
```

이 처럼 data type은 있지만 변수에 저장되는 value(값0의 type(타입)은 언제든지 바꿀 수 있는 언어를 '동적 타입(dynamically typed)' 언어라고 부른다.

❖ Number

```
let n = 123;
n = 12.345;
```

number type은 integer(정수) 및 floating point(부동소수점) number(floating point number)를 나타낸다. number과 관련된 연산은 다양한데, 곱셈 *, 나눗셈 /, 덧셈 +, 뺄셈 - 등이 대표적이다.
number엔 일반적인 number 외에 Infinity, -Infinity, NaN같은 '특수 number 값(special numeric value)'이 포함된다.

☞ **Infinity는 어떤 number보다 더 큰 특수 값, 무한대(∞)를 나타낸다.**

어느 number든 0으로 나누면 무한대를 얻을 수 있다.

```
console.log( 1 / 0 );     // 무한대
```

Infinity를 직접 참조할 수도 있다.

```
console.log( Infinity );  // 무한대
```

☞ NaN은 계산 중에 error가 발생했다는 것을 나타내주는 값이다.

부정확하거나 정의되지 않은 수학 연산을 사용하면 계산 중에 에러가 발생하는데, 이때 NaN이 반환된다.

```
console.log( "number가 아님" / 2 ); // NaN, string을 number로 나누면 error가 발생한다.
```

NaN은 여간해선 바뀌지 않는다. NaN에 어떤 추가 연산을 해도 결국 NaN이 반환된다.

```
console.log( "number가 아님" / 2 + 5 ); // NaN
```

연산 과정 어디에선가 NaN이 반환되었다면, 이는 모든 결과에 영향을 미친다.

⊙ 수학 연산은 안전하다.

JavaScript에서 행해지는 수학 연산은 '안전'하다고 볼 수 있다. 0으로 나눈다거나 number가 아닌 string을 number로 취급하는 등의 이례적인 연산이 JavaScript에선 가능하다.
말이 안 되는 수학 연산을 하더라도 script는 fatal error(치명적인 에러)를 내뿜으며 죽지 않는다. NaN을 반환하며 연산이 종료될 뿐이다.

현실에선 special number 값을 number로 취급하진 않는다. 하지만 JavaScript에선 special number 값을 number으로 분류한다.

❖ BigInt

내부 표현 방식 때문에 JavaScript에선 (253-1)(9007199254740991) 보다 큰 값 혹은 -(253-1) 보다 작은 정수는 'number'을 사용해 나타낼 수 없다.
사실 대부분의 상황에서 이런 제약사항은 문제가 되지 않는다. 그렇지만 암호 관련 작업같이 아주 큰 number가 필요한 상황이거나 아주 높은 정밀도로 작업을 해야 할 때는 이런 큰 number가 필요하다.

BigInt type은 표준으로 채택된 지 얼마 안 된 data type으로, 길이에 상관없이 정수를 나타낼 수 있다. BigInt type 값은 정수 리터럴 끝에 n을 붙이면 만들 수 있다.

```
// 끝에 'n'이 붙으면 BigInt형 자료이다.
const bigInt = 1234567890123456789012345678901234567890n;
```

BigInt number는 자주 쓰이지 않기 때문에 여기서 자세히 다루지 않도록 한다.

⊙ 호환성 이슈

현재 시점엔 Firefox, Chrome, Edge, Safari에서만 BigInt를 지원한다. IE에선 지원하지 않는다.

❖ String

JavaScript에선 string을 따옴표로 묶는다.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

따옴표는 3 종류가 있다.

☞ Double quote(큰따옴표): "Hello"
☞ Single quote(작은따옴표): 'Hello'
☞ Backtick: `Hello`

큰따옴표와 작은따옴표는 '기본적인' 따옴표로, JavaScript에서는 이 둘에 차이를 두지 않는다.

Backtick 으로 변수나 표현식을 감싼 후 ${…}안에 넣어주면, 아래와 같이 원하는 변수나 표현식을 string 중간에 손쉽게 넣을 수 있다.

```
let name = "Smart";

// 변수를 string 중간에 삽입
console.log( `Hello, ${name}!` );          // Hello, Smart!

// expression(표현식)을 string 중간에 삽입
console.log( `the result is ${1 + 2}` );   // the result is 3
```

${…} 안에는 name 같은 변수나 1 + 2 같은 수학 관련 표현식을 넣을 수 있다. 물론 더 복잡한 표현식도 넣을 수 있다. 무엇이든 들어갈 수 있다. 이렇게 string 중간에 들어간 변수나 표현식은 평가가 끝난 후 string의 일부가 된다.

큰따옴표나 작은따옴표를 사용하면 중간에 표현식을 넣을 수 없다는 점에 주의한다. 이 방법은 역 따옴표를 써야만 가능하다.

```
// the result is ${1 + 2} (큰따옴표는 확장 기능을 지원하지 않는다.)
console.log( "the result is ${1 + 2}" );
```

⊙ character type은 없다.

일부 언어는 character 하나를 저장할 때 쓰이는 data type, '글자(character)'형을 따로 지원한다. C 언어와 Java의 char가 대표적인 예이다.

JavaScript는 character type을 지원하지 않는다. string type만 있을 뿐이다. 여기엔 글자가 하나 혹은 여러 개 들어갈 수 있다.

❖ Boolean(logical type)

boolean type(논리 타입)은 true와 false 두 가지 값밖에 없는 data type 이다.
boolean type은 긍정(yes)이나 부정(no)을 나타내는 값을 저장할 때 사용한다. true는 긍정, false는 부정을 의미한다.

[예시]

```
let nameFieldChecked = true;       // 네, name field가 확인되었습니다(checked).
let ageFieldChecked = false;       // 아니요, age field를 확인하지 않았습니다(not checked)
```

boolean value(불린값)은 비교 결과를 저장할 때도 사용된다.

```
let isGreater = 4 > 1;

console.log( isGreater ); // true (비교 결과: "yes")
```

❖ 'null' value

null value는 지금까지 소개한 data type 중 어느 data type에도 속하지 않는 값 이다.
null 값은 오로지 null 값만 포함하는 별도의 data type을 만든다.

```
let age = null;
```

JavaScript의 null은 JavaScript 이외 언어의 null과 성격이 다르다. 다른 언어에선 null을 '존재하지 않는 object에 대한 'reference(참조)'나 'null pointer'를 나타낼 때 사용한다.

그러나 JavaScript에선 null을 '존재하지 않는(nothing)' 값, '비어 있는(empty)' 값, '알 수 없는(unknown)' 값을 나타내는 데 사용한다.

let age = null;은 나이(age)를 알 수 없거나 그 값이 비어있음을 보여준다.

❖ 'undefined' value

undefined value도 null 값처럼 자신만의 data type을 형성한다. undefined는 'value가 할당되지 않은 상태'를 나타낼 때 사용한다.
변수는 선언했지만, 값을 할당하지 않았다면 해당 변수에 undefined가 자동으로 할당된다.

```
let age;

console.log(age);          // 'undefined'가 출력된다.
```

개발자가 변수에 undefined를 명시적으로 할당하는 것도 가능하긴 하다.

```
let age = 100;

// 값을 undefined로 바뀐다.
age = undefined;

console.log(age);          // "undefined"
```

하지만 이렇게 undefined를 직접 할당하는 걸 권장하진 않는다. 변수가 '비어있거나' '알 수 없는' 상태라는 걸 나타내려면 null을 사용한다. undefined는 값이 할당되지 않은 변수의 초기값을 위해 예약어로 남겨둔다.

❖ Object와 Symbol

object type은 특수한 data type 이다.
object type을 제외한 다른 data type은 string이든 number든 한 가지만 표현할 수 있기 때문에 primitive data type이라 부른다. 반면 object는 data collection이나 complex entity를 표현할 수 있다.
이런 특징 때문에 JavaScript에서 object는 좀 더 특별한 취급을 받는다. 자세한 내용은 추후 Obejct에서 다루도록 한다.

symbol type은 object의 고유한 식별자(unique identifier)를 만들 때 사용된다. symbol type에 대해도 object 이후 학습하도록 한다.

❖ typeof operator

typeof operator는 argument의 data type을 반환한다. data type에 따라 처리 방식을 다르게 하고 싶거나, 변수의 data type을 빠르게 알아내고자 할 때 유용하다.

typeof operator는 두 가지 형태의 문법을 지원한다.

① operator: typeof x
② 함수: typeof(x)

괄호가 있든 없든 결과가 동일하다.

☞ **typeof x를 호출하면 인수의 data type을 나타내는 string을 반환한다.**

```
typeof undefined          // "undefined"
typeof 0                  // "number"
typeof 10n                // "bigint"
typeof true               // "boolean"
typeof "foo"              // "string"
typeof Symbol("id")       // "symbol"
typeof Math               // "object"      (1)
typeof null               // "object"      (2)
typeof alert              // "function"    (3)
```

마지막 세 줄은 약간의 설명이 필요하다

☞ Math는 수학 연산을 제공하는 built-in object이므로 "object"가 출력된다. built-in object는 object type이라는 것을 알려주기 위해 이런 예시를 작성해 보았다.
☞ typeof null의 결과는 "object" 이다. null은 별도의 고유한 data type을 가지는 특수 값으로 object가 아니지만, 하위 호환성을 유지하기 위해 이런 오류를 수정하지 않고 남겨둔 상황이다. 언어 자체의 오류이므로 null이 object가 아님에 유의한다.
☞ alert은 function 이기 때문에 typeof alert의 결과는 "function" 이다. JavaScript에는 특별한 "function" type이 없다는 것도 알게 될 것이다. function은 object type에 속한다. 그러나 typeof는 "function"를 반환하여 다르게 취급한다. 그것도 JavaScript의 초창기에서 온 것이다. 기술적으로 이러한 동작은 올바르지 않지만 실제로는 편리할 수 있다.

⊙ typeof(x) syntax

또 다른 구문인 typeof(x)를 접할 수도 있다. typeof x와 동일하다.
명확히 하자면 typeof는 function이 아니라 operator 이다. 여기서 괄호는 typeof의 일부가 아니다. 수

학적 그룹화에 사용되는 일종의 괄호 이다. 일반적으로 이러한 괄호에는 (2 + 2)와 같은 수학적 표현이 포함되지만 여기서는 하나의 인수(x)만 포함한다. 구문적으로 typeof operator와 argument 사이에 공백을 피할 수 있으며 일부 사람들은 이를 좋아한다. 어떤 사람들은 typeof(x)를 선호하지만 typeof x 구문이 훨씬 더 일반적이다.

요약 하면 JavaScript에는 8가지 기본 data type이 있다.

- ☞ number: integer, floating-point 등의 number를 나타낼 때 사용한다. 정수의 한계는 ±(2**53-1) 이다.
- ☞ bigint: 길이 제약 없이 정수를 나타낼 수 있다.
- ☞ string: 빈 string이나 글자들로 이뤄진 string을 나타낼 때 사용한다. 단일 문자를 나타내는 별도의 data type은 없다.
- ☞ boolean: true, false를 나타낼 때 사용한다.
- ☞ null: null 값만을 위한 독립 data type 이다. null은 알 수 없는 값을 나타낸다.
- ☞ undefined: undefined 값만을 위한 독립 data type 이다. undefined는 할당되지 않은 값을 나타낸다.
- ☞ object: 복잡한 datastructure를 표현할 때 사용한다.
- ☞ symbol: object의 고유 식별자를 만들 때 사용한다.

typeof operator를 사용하면 변수에 어떤 type이 저장되어 있는지 확인할 수 있다.

- ☞ typeof x 또는 typeof(x) 형태로 사용한다.
- ☞ "string"과 같은 type의 이름을 가진 string을 반환한다.
- ☞ null의 typeof 연산은 "object"인데, 이는 언어상 오류이다. null은 object가 아니다.

[과제]

String quote

중요도: 5

아래 script의 결과를 예측해 보자.

```
let name = "Semyung";

console.log( `hello ${1}` );      // ?
console.log( `hello ${"name"}` ); // ?
console.log( `hello ${name}` );   // ?
```

Backtick은 ${...} 내부의 expression을 string에 포함한다.

```
let name = "Semyung";

// expression은 number 1 이다.
console.log( `hello ${1}` );       // hello 1

// expression은 string "name" 이다.
console.log( `hello ${"name"}` ); // hello name

// expression 안에 변수가 들어가 있기 때문에, 이 변수가 평가되어 전체 string이 반환된다.
console.log( `hello ${name}` );    // hello Semyung
```