JavaScript에서의 Objects 정의

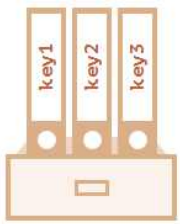
☑ Objects

JavaScript엔 8 가지 Data Types이 있다. 이 중 7개는 오직 하나의 데이터(string, number 등)만 담을 수 있어 'Primitive Type'이라 부른다.

그런데 object type은 primitive type과 달리 다양한 데이터를 담을 수 있다. key로 구분된 데이터 집합이나 복잡한 entity를 저장할 수 있다. object는 JavaScript 거의 모든 면에 녹아있는 개념이므로 JavaScript를 잘 다루려면 object를 잘 이해하고 있어야 한다.

object는 중괄호 {…}를 이용해 만들 수 있다. 중괄호 안에는 'key: value' 쌍으로 구성된 property를 여러 개 넣을 수 있는데, key엔 string, value엔 모든 자료형이 허용된다. property key는 'property name' 이라고도 부른다.

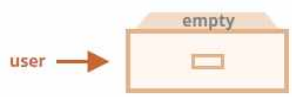
cabinet을 상상하면 object를 이해하기 쉽다. cabinet 안 파일은 property, 파일 각각에 붙어있는 이름표는 object의 key라고 생각하면 된다. 복잡한 cabinet 안에서 이름표를 보고 원하는 파일을 쉽게 찾을 수 있듯이, object에선 key를 이용해 property를 쉽게 찾을 수 있다. 추가나 삭제도 마찬가지이다.



빈 object(빈 cabinet)를 만드는 방법은 두 가지가 있다.

방법1)
```
let user = new Object();          // 'object constructor' 문법
```

방법2)
```
let user = {};                    // 'object literal' 문법
```



중괄호 {...}를 이용해 object를 선언하는 것을 object literal 이라고 부른다. object를 선언할 땐 주로 이 방법을 사용한다.

❖ literal과 property

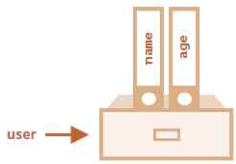
중괄호 {...} 안에는 'key: value' 쌍으로 구성된 property가 들어간다.

```
let user = {              // object
  name: "smu",            // key: "name",  value: "smu"
  age: 33                 // key: "age", value: 33
};
```

'콜론(:)'을 기준으로 왼쪽엔 key가, 오른쪽엔 value 가 위치한다. key는 property 'name' 혹은 'identifier' 라고도 부른다.

object user에는 property가 두 개 있다.
첫번째 property는 name으로 "name"과 value로 "smu" 이다.
두번째 property는 name으로 "age"와 value로 33 이다.

cabinet(object user) 안에 파일 property 2개가 담겨있는데, 각 파일에 "name", "age"라는 이름표가 붙어있다고 생각하면 쉽다.

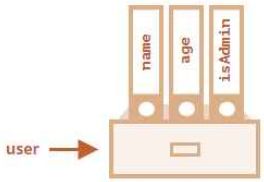


cabinet에 파일을 추가하고 뺄 수 있듯이 개발자는 property를 추가, 삭제할 수 있다.
dot notation(점 표기법)을 이용하면 property 값을 읽는 것도 가능하다.

```
// property 값 얻기
alert( user.name );       // smu
alert( user.age );        // 33
```

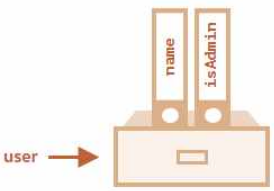
property value엔 모든 자료형이 올 수 있다. boolean property를 추가해보자.

```
user.isAdmin = true;
```



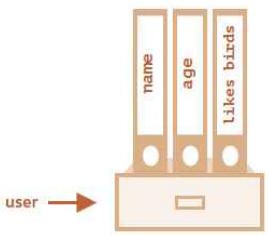
delete 연산자를 사용하면 property를 삭제할 수 있다.

```
delete user.age;
```



여러 단어를 조합해 property name을 만든 경우엔 property name을 따옴표(" ")로 묶어줘야 한다.

```
let user = {
  name: "smu",
  age: 33,
  "likes jobs": true      // 복수의 단어는 따옴표로 묶어야 한다.
};
```



마지막 property 끝은 쉼표(,)로 끝날 수 있다.

```
let user = {
  name: "smu",
  age: 33,
}
```

이런 쉼표를 'trailing(길게 늘어지는)' 혹은 'hanging(매달리는)' 쉼표라고 부른다. 이렇게 끝에 쉼표를 붙이면 모든 property가 유사한 형태를 보이기 때문에 property를 추가, 삭제, 이동하는 게 쉬워진다.

const object는 수정될 수 있다. 주의한다. const로 선언된 object는 수정될 수 있다.

예시:

```
const user = {
  name: "SMU"
};

user.name = "SmartIT";            // (1)

alert(user.name);                 // SmartIT
```

(1)로 표시한 줄에서 오류를 일으키는 것처럼 보일 수 있지만 그렇지 않다. const는 user의 값을 고정하지만, 그 내용은 고정하지 않는다. const는 user=...를 전체적으로 설정하려고 할 때만 오류가 발생한다.

const object property를 만드는 또 다른 방법이 있다. 이후에 property flag와 descriptor(설명자)에서 다루도록 한다.

[참고]

object엔 property가 저장된다. 지금까진 property를 단순히 'key-value' 쌍의 관점에서만 다뤘다. 그러나 사실 property는 우리가 생각했던 것보다 더 유연하고 강력한 자료구조이다. object property 추가 구성 옵션 몇 가지를 다루고, 이 옵션들을 이용해 손쉽게 getter나 setter 함수를 만드는 법을 학습하도록 한다.

object property는 value 와 함께 flag라 불리는 특별한 속성 3가지를 갖는다.

- ☞ writable: true이면 값을 수정할 수 있다. 그렇지 않다면 읽기만 가능하다.
- ☞ enumerable: true이면 반복문을 사용해 나열할 수 있다. 그렇지 않다면 반복문을 사용해 나열할 수 없다.
- ☞ configurable: true이면 Property 삭제나 flag 수정이 가능하다. 그렇지 않다면 property 삭제와 flag 수정이 불가능하다.

property flag그는 특별한 경우가 아니고선 다룰 일이 없다. 지금까지 해왔던 '평범한 방식'으로 property를 만들면 해당 property의 flag는 모두 true가 된다. 이렇게 true로 설정된 flag는 언제든 수정할 수 있다.

❖ Square Bracket Notation(대괄호 표기법)

여러 단어를 조합해 property key를 만든 경우엔, 점 표기법을 사용해 property value를 읽을 수 없다.

```
// 문법 에러가 발생한다.
user.likes jobs = true
```

JavaScript는 위와 같은 코드를 이해하지 못한다. user.likes까지는 이해하다가 예상치 못한 jobs를 만나면 문법 에러를 뱉어낸다.

'dot(.)'은 key가 'valid variable identifier(유효한 변수 식별자)'인 경우에만 사용할 수 있다. 유효한 변수 식별자엔 공백이 없어야 한다. 또한 숫자로 시작하지 않아야 하며 $와 _를 제외한 특수 문자가 없어야 한다.

key가 유효한 변수 식별자가 아닌 경우엔 dot 표기법 대신에 'square bracket notation(대괄호 표기법)'이라 불리는 방법을 사용할 수 있다. 대괄호 표기법은 key에 어떤 string(문자열)이 있던지 상관없이 동작한다.

```
let user = {};

// set
user["likes jobs"] = true;

// get
alert(user["likes jobs"]); // true

// delete
delete user["likes jobs"];
```

이제 문법 에러가 발생하지 않는다. 대괄호 표기법 안에서 string(문자열)을 사용할 땐 string을 따옴표로 묶어줘야 한다는 점에 주의한다. 따옴표의 종류(" " 혹은 ' ')는 상관없다.

대괄호 표기법을 사용하면 아래 예시에서 변수를 key로 사용한 것과 같이 문자열뿐만 아니라 모든 표현식의 평가 결과를 property key로 사용할 수 있다.

```
let key = "likes jobs";
```

```
// user["likes jobs"] = true; 와 같다.
user[key] = true;
```

변수 key는 런타임에 평가되기 때문에 사용자 입력값 변경 등에 따라 값이 변경될 수 있다. 어떤 경우든, 평가가 끝난 이후의 결과가 property key로 사용된다. 이를 응용하면 코드를 유연하게 작성할 수 있다.

예시:

```
let user = {
  name: "smu",
  age: 33
};

let key = prompt("사용자의 어떤 정보를 얻고 싶으신가요?", "name");

// 변수로 접근
alert( user[key] );       // smu (prompt 창에 "name"을 입력한 경우)
```

그런데 dot 표기법은 이런 방식이 불가능하다.

```
let user = {
  name: "smu",
  age: 33
};

let key = "name";
alert( user.key );        // undefined
```

⊙ Computed property

object를 만들 때 object literal 안의 property key가 대괄호로 둘러싸여 있는 경우, 이를 computed property 라고 부른다.

예시:

```
let fruit = prompt("어떤 과일을 구매하시겠습니까?", "apple");

let bag = {
  [fruit]: 5,             // 변수 fruit에서 property name을 동적으로 받아온다.
};

alert( bag.apple );       // fruit에 "apple"이 할당되었다면, 5가 출력된다.
```

위 예시에서 [fruit]는 property name을 변수 fruit에서 가져오겠다는 것을 의미한다.

사용자가 prompt 대화상자에 apple을 입력했다면 bag엔 {apple: 5}가 할당되었을 것이다.

아래 예시는 위 예시와 동일하게 동작한다.

```
let fruit = prompt("어떤 과일을 구매하시겠습니까?", "apple");
let bag = {};

// variable fruit을 사용해 property name을 만들었다.
bag[fruit] = 5;
```

두 방식 중 Computed property를 사용한 예시가 더 깔끔해 보인다.

한편, 다음 예시처럼 대괄호 안에는 복잡한 표현식이 올 수도 있다.

```
let fruit = 'apple';
let bag = {
  [fruit + 'Computers']: 5        // bag.appleComputers = 5
};
```

대괄호 표기법은 property name과 value의 제약을 없애주기 때문에 dot 표기법보다 훨씬 강력하다. 그런데 작성하기 번거롭다는 단점이 있다.

이런 이유로 property name이 확정된 상황이고, 단순한 이름이라면 처음엔 dot 표기법을 사용하다가 뭔가 복잡한 상황이 발생했을 때 대괄호 표기법으로 바꾸는 경우가 많다.

❖ Property value shotthand

실무에선 property value를 기존 변수에서 받아와 사용하는 경우가 종종 있다.

예시:

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...등등
  };
}

let user = makeUser("smu", 33);
```

```
alert(user.name);        // smu
```

위 예시의 property들은 name과 value가 변수의 이름과 동일하다. 이렇게 변수를 사용해 property를 만드는 경우는 아주 흔한데, property value shorthand(프로퍼티 값 단축 구문) 을 사용하면 코드를 짧게 줄일 수 있다.

name:name 대신 name만 적어주어도 property를 설정할 수 있다.

```
function makeUser(name, age) {
  return {
    name,        // name: name 과 같음
    age,         // age: age 와 같음
    // ...
  };
}
```

한 object에서 일반 property와 shorthand property를 함께 사용하는 것도 가능하다.

```
let user = {
  name,          // name: name 과 같음
  age: 33
};
```

❖ Property names limitation(프로퍼티 이름의 제약사항)

변수 이름(key)엔 'for', 'let', 'return' 같은 예약어를 사용하면 안된다. 그러나 object property엔 이런 제약이 없다.

```
// 예약어를 key로 사용해도 괜찮다.
let obj = {
  for: 1,
  let: 2,
  return: 3
};

alert( obj.for + obj.let + obj.return );  // 6
```

이와 같이 property name엔 특별한 제약이 없다. 어떤 문자형, 심볼형 값도 property key가 될 수 있다(identifier로 쓰이는 Symbol에 대해선 뒤에서 다룰 예정이다).

string나 symbol에 속하지 않은 값은 string(문자열)로 automatically converted(자동형 변환)된다.

예시를 보자. key에 number(숫자) 0을 넣으면 string(문자열) "0"으로 자동변환 된다.

```
let obj = {
  0: "test"      // "0": "test"와 동일하다.
};

// 숫자 0은 문자열 "0"으로 변환되기 때문에 두 alert 창은 같은 property에 접근한다.
alert( obj["0"] );       // test
alert( obj[0] );         // test (동일한 property)
```

이와같이 object property key에 쓸 수 있는 문자열엔 제약이 없지만, 역사적인 이유 때문에 특별 대우를 받는 이름이 하나 있다. 바로, __proto__ 이다.

```
let obj = {};
obj.__proto__ = 5;       // 숫자를 할당한다.
// [object Object] => 숫자를 할당했지만 값은 object가 되었다. 의도한대로 동작하지 않는다.
alert(obj.__proto__);
```

primitive 5를 할당했는데 무시된 것을 확인할 수 있다.

__proto__의 본질은 prototype inheritance에서 이 문제를 어떻게 해결할 수 있을지에 대해선 prototype method와 __proto__가 없는 object에서 자세히 다루도록 한다.

⊙ 'in' operator로 property 존재 여부 확인하기

JavaScript object의 중요한 특징 중 하나는 다른 언어와는 달리, 존재하지 않는 property에 접근하려 해도 에러가 발생하지 않고 undefined를 반환한다는 것이다.

이런 특징을 응용하면 property 존재 여부를 쉽게 확인할 수 있다.

```
let user = {};

alert( user.noSuchProperty === undefined ); // true는 'property가 존재하지 않음'을 의미한다.
```

이렇게 undefined와 비교하는 것 이외에도 연산자 in을 사용하면 property 존재 여부를 확인할 수 있다.

문법은 다음과 같다.

```
"key" in object
```

예시:

```
let user = { name: "smu", age: 33 };

alert( "age" in user );    // user.age가 존재하므로 true가 출력된다.
alert( "gender" in user ); // user.gender는 존재하지 않기 때문에 false가 출력된.
```

in 왼쪽엔 반드시 property name이 와야 한다. property name은 보통 따옴표로 감싼 string(문자열)이다.

따옴표를 생략하면 아래 예시와 같이 엉뚱한 변수가 조사 대상이 된다.

```
let user = { age: 33 };

let key = "age";
// true, 변수 key에 저장된 값("age")을 사용해 property 존재 여부를 확인한다.
alert( key in user );
```

그런데 이쯤 되면 "undefined 랑 비교해도 충분한데 왜 in oprator가 있는 거지?"라는 의문이 들 수 있다.
대부분의 경우, comparison operator(일치/비교 연산자)를 사용해서 property 존재 여부를 알아내는 방법("=== undefined")은 꽤 잘 동작한다. 그런데 가끔은 이 방법이 실패할 때도 있다. 이럴 때 in을 사용하면 property 존재 여부를 제대로 판별할 수 있다.

property는 존재하는데, 값에 undefined를 할당한 예시를 살펴보자.

```
let obj = {
  test: undefined
};

// 값이 `undefined`이므로, alert 창엔 undefined가 출력된다. 그런데 property test는 존재한다.
alert( obj.test );

// `in`을 사용하면 property 유무를 제대로 확인할 수 있다(true가 출력됨).
alert( "test" in obj );
```

obj.test는 실제 존재하는 property 이다. 따라서 in operator는 정상적으로 true를 반환한다.

undefined는 변수는 정의되어 있으나 value가 할당되지 않은 경우에 쓰기 때문에 property value가 undefined인 경우는 흔치 않다. value를 '알 수 없거나(unknown)' 값이 '비어 있다는(empty)' 것을 나

타낼 때는 주로 null을 사용한다. 위 예시에서 in operator는 자리에 어울리지 않는 초대손님처럼 보인다.

⊙ 'for…in' loop

for..in loop(반복문)을 사용하면 object의 모든 key를 순회할 수 있다. for..in은 학습했던 for(;;) 반복문과는 완전히 다르다.

문법:

```
for (key in object) {
  // 각 property key(key)를 이용하여 본문(body)을 실행한다.
}
```

아래 예시를 실행하면 object user의 모든 property가 출력된다.

```
let user = {
  name: "smu",
  age: 33,
  isAdmin: true
};

for (let key in user) {
  // key
  alert( key );          // name, age, isAdmin
  // key에 해당하는 값
  alert( user[key] );    // smu, 33, true
}
```

for..in 반복문에서도 for(;;)문처럼 반복 변수(looping variable)를 선언(let key)했다는 점을 주목한다. 반복 변수명은 자유롭게 정할 수 있다. 'for (let prop in obj)'같이 key 말고 다른 변수명을 사용해도 괜찮다.

⊙ object 정렬 방식

object와 object property를 다루다 보면 "property엔 순서가 있을까?"라는 의문이 생기기 마련이다. 반복문은 property를 추가한 순서대로 실행될 지, 그리고 이 순서는 항상 동일할지 궁금해진다.

object는 '특별한 방식으로 정렬' 된다. integer property(정수 프로퍼티)는 자동으로 정렬되고, 그 외의 property는 object에 추가한 순서 그대로 정렬된다. 예제를 통해 살펴보자.

아래 object 엔 국제전화 국가 번호가 담겨있다.

```
let codes = {
  "82": "대한민국",
  // ..,
  "49": "독일",
  "41": "스위스",
  "44": "영국",
  // ..,
  "1": "미국"
};

for (let code in codes) {
  alert(code); // 1, 41, 44, 49, 82
}
```

현재 개발 중인 애플리케이션의 주 사용자가 한국인이라고 가정해 보자. 나라 번호를 선택하는 화면에서 82가 맨 앞에 오도록 하는 게 좋을 것이다. 그런데 코드를 실행해 보면 예상과는 전혀 다른 결과가 출력된다. 미국(1)이 첫 번째로 출력된다. 그 뒤로 스위스(41), 영국(44), 독일(49), 대한민국(82) 이 차례대로 출력된다. 이유는 국가 번호(key)가 integer(정수)이어서 1, 41, 44, 49, 82 순으로 property가 자동 정렬되었기 때문이다.

☞ Integer property

'integer property'라는 용어는 변형없이 integer에서 왔다 갔다 할 수 있는 string(문자열)을 의미한다.
string "82"는 integer로 변환하거나 변환한 integer를 다시 string로 바꿔도 변형이 없기 때문에 정 integer property 이다. 그러나 '+82'와 '1.2'는 integer property가 아니다.

```
// 함수 Math.trunc는 소수점 아래를 버리고 숫자의 정수부만 반환한다.
// '82'가 출력된다. 기존에 입력한 값과 같으므로 integer property 이다.
alert( String(Math.trunc(Number("82"))) );

// '82'가 출력된다. 기존에 입력한 값(+82)과 다르므로 integer property가 아니다.
alert( String(Math.trunc(Number("+82"))) );

// '1'이 출력된다. 기존에 입력한 값(1.2)과 다르므로 integer property가 아니다.
alert( String(Math.trunc(Number("1.2"))) );
```

한편, key가 integer가 아닌 경우엔 작성된 순서대로 property가 나열된다. 예시를 살펴보자.

```
let user = {
  name: "smu",
  surname: " semyung"
```

```
};
user.age = 33;    // property를 하나 추가한다.

// integer property가 아닌 property는 추가된 순서대로 나열된다.
for (let prop in user) {
  alert( prop ); // name, surname, age
}
```

위 예시에서 82(대한민국 국가 번호)를 가장 위에 출력되도록 하려면 국가 번호가 integer로 취급되지 않도록 속임수를 쓰면 된다. 각 나라 번호 앞에 "+"를 붙여본다.

아래 같이 한다.

```
let codes = {
  "+82": "대한민국",
  // ..,
  "+49": "독일",
  "+41": "스위스",
  "+44": "영국",
  // ..,
  "+1": "미국"
};

for (let code in codes) {
  alert( +code ); // 82, 49, 41, 44, 1
}
```

이제 원하는 대로 대한민국 국가 번호가 가장 먼저 출력되는 것을 확인할 수 있다.