Enumerability and ownership of properties

Every property in JavaScript objects can be classified by three factors:

- Enumerable or non-enumerable;
- String or symbol;
- Own property or inherited property from the prototype chain.

Enumerable properties are those properties whose internal enumerable flag is set to true, which is the default for properties created via simple assignment or via a property initializer. Properties defined via `Object.defineProperty` and such are not enumerable by default. Most iteration means (such as `for...in` loops and `Object.keys`) only visit enumerable keys.

Ownership of properties is determined by whether the property belongs to the object directly and not to its prototype chain.

All properties, enumerable or not, string or symbol, own or inherited, can be accessed with dot notation or bracket notation. In this section, we will focus on JavaScript means that visit a group of object properties one-by-one.

Querying object properties

There are four built-in ways to query a property of an object. They all support both string and symbol keys. The following table summarizes when each method returns `true`.

	Enumerable, own	Enumerable, inherited	Non-enumerable, own	Non-enumerable, inherited
`propertyIsEnumerable()`	`true` ✓	`false` ✗	`false` ✗	`false` ✗
`hasOwnProperty()`	`true` ✓	`false` ✗	`true` ✓	`false` ✗
`Object.hasOwn()`	`true` ✓	`false` ✗	`true` ✓	`false` ✗
`in`	`true` ✓	`true` ✓	`true` ✓	`true` ✓

Traversing object properties

There are many methods in JavaScript that traverse a group of properties of an object. Sometimes, these properties are returned as an array; sometimes, they are iterated one-by-one in a loop; sometimes, they are used for constructing or mutating another object. The following table summarizes when a property may be visited.

Methods that only visit string properties or only symbol properties will have an extra note. ☑ means a property of this type will be visited; ✗ means it will not.

	Enumerable, own	Enumerable, inherited	Non-enumerable, own	Non-enumerable, inherited
`Object.keys` `Object.values` `Object.entries`	☑ (strings)	✗	✗	✗
`Object.getOwnPropertyNames`	☑ (strings)	✗	☑ (strings)	✗
`Object.getOwnPropertySymbols`	☑ (symbols)	✗	☑ (symbols)	✗
`Object.getOwnPropertyDescriptors`	☑	✗	☑	✗
`Reflect.ownKeys`	☑	✗	☑	✗
`for...in`	☑ (strings)	☑ (strings)	✗	✗
`Object.assign` `(After the first parameter)`	☑	✗	✗	✗
`Object spread`	☑	✗	✗	✗

Obtaining properties by enumerability/ownership

Note that this is not the most efficient algorithm for all cases, but useful for a quick demonstration.

- Detection can occur by `SimplePropertyRetriever.theGetMethodYouWant(obj).includes(prop)`
- Iteration can occur by `SimplePropertyRetriever.theGetMethodYouWant(obj).forEach((value, prop) => {}); (or use filter(), map()`, etc.)

```
const SimplePropertyRetriever = {
  getOwnEnumerables(obj) {
    return this._getPropertyNames(obj, true, false, this._enumerable);
    // Or could use for...in filtered with Object.hasOwn or just this: return
    Object.keys(obj);
  },
  getOwnNonenumerables(obj) {
    return this._getPropertyNames(obj, true, false, this._notEnumerable);
  },
  getOwnEnumerablesAndNonenumerables(obj) {
    return this._getPropertyNames(
      obj,
      true,
      false,
      this._enumerableAndNotEnumerable,
    );
    // Or just use: return Object.getOwnPropertyNames(obj);
  },
  getPrototypeEnumerables(obj) {
    return this._getPropertyNames(obj, false, true, this._enumerable);
  },
  getPrototypeNonenumerables(obj) {
    return this._getPropertyNames(obj, false, true, this._notEnumerable);
  },
  getPrototypeEnumerablesAndNonenumerables(obj) {
    return this._getPropertyNames(
      obj,
      false,
      true,
      this._enumerableAndNotEnumerable,
    );
  },
  getOwnAndPrototypeEnumerables(obj) {
    return this._getPropertyNames(obj, true, true, this._enumerable);
    // Or could use unfiltered for...in
  },
  getOwnAndPrototypeNonenumerables(obj) {
    return this._getPropertyNames(obj, true, true, this._notEnumerable);
  },
  getOwnAndPrototypeEnumerablesAndNonenumerables(obj) {
    return this._getPropertyNames(
      obj,
      true,
      true,
      this._enumerableAndNotEnumerable,
    );
  },
  // Private static property checker callbacks
  _enumerable(obj, prop) {
    return Object.prototype.propertyIsEnumerable.call(obj, prop);
    },
    _notEnumerable(obj, prop) {
      return !Object.prototype.propertyIsEnumerable.call(obj, prop);
    },
    _enumerableAndNotEnumerable(obj, prop) {
      return true;
    },
    // Inspired by http://stackoverflow.com/a/8024294/271577
    _getPropertyNames(obj, iterateSelf, iteratePrototype, shouldInclude) {
      const props = [];
      do {
        if (iterateSelf) {
          Object.getOwnPropertyNames(obj).forEach((prop) => {
            if (props.indexOf(prop) === -1 && shouldInclude(obj, prop)) {
              props.push(prop);
            }
          });
        }
        if (!iteratePrototype) {
          break;
        }
        iterateSelf = true;
        obj = Object.getPrototypeOf(obj);
      } while (obj);
      return props;
    },
  };
```