WORKING WITH OBJECTS

Ultimate JavaScript Objects

INTRODUCTION

- JavaScript objects are easy to learn, yet tough to master
- Numerous ways to create, edit and access – some with differing consequences
- New ES6 features add to required body of knowledge
- Some methods (creation, mapping) are used every day
- Some features (like Freezing and Copying) are more obscure but still useful

MODULE ROADMAP – WORKING WITH OBJECTS

- Create objects using object literal notation (and alternatives)
- Add and access object properties
- Iterate over properties
- Map arrays of objects
- Explore ES6 features – symbol, observe and freeze
- Learn about object.prototype

CREATING OBJECTS

- Objects can be created using object literal syntax { }
- Objects can also be created with the new keyword
- Can also be created with Object.create
- No perfect solution for every situation
- Using *new* allows for classlike behavior and prototypical inheritance

ADDING AND ACCESSING OBJECT PROPERTIES

- Objects can be initialized by putting key value pairs inside curly brackets
- Multiple key-value pairs separated by commas
- Properties can be added to existing objects with dot or bracket notation
- Access also occurs via dot or bracket notation
- Some properties only accessible via bracket notation

```
{
  “name”:“Jon Snow”,
  “greatWolf”:“Ghost”
}
```

ADDING METHODS TO OBJECTS

- Objects can have properties which are functions
- These are called methods
- No difference between method and function except *this* keyword
- No practical reason to implement non-static methods without *this*
- The *this* keyword will be fully explained in the next chapter on Object Scope

REMOVING OBJECT PROPERTIES

- Property values can be removed without removing the key by setting them to *undefined*
- Both key and value can be removed by using the delete keyword

ITERATING THROUGH OBJECT PROPERTIES

- Object properties can be iterated through via For In Loop
- Used to be very challenging to do this in ES5, now is easy
- Looping doesn't usually make sense since objects tend to have different types of values as variables
- Sometimes can be useful with array-like objects

MAPPING ARRAYS OF OBJECTS

- An extremely common problem is needing to transform (or map) an array of congruent objects into a different type of object
- E.g., database transfer script, React store
- Can easily be mapped with built-in array.map()
- Editing existing objects / copying both possible

SYMBOLLY AMAZING - ACCESSING OBJECT PROPERTIES WITH SYMBOLS

- Keeping references to object properties via strings is a time-honored and beloved hack
  - Using non-unique indexes can lead to catastrophically hard-to-debug errors
  - Two separate components of app may unwittingly use same property name, i.e., “name” or “health”
- Symbols are guaranteed to be unique
- Conceptually identical to a GUID (long, random string)

“A **symbol** is a unique and immutable data type and may be used as an identifier for object properties. ” - MDN

FREEZE & SEAL

- New to ES6
- Both restrict (control) ways an object can be modified
- Mostly used to prevent developer error
- Prevents new properties from being added (seal) or anything at all from changing (freeze)

OBSERVING CHANGES WITH PROXIES

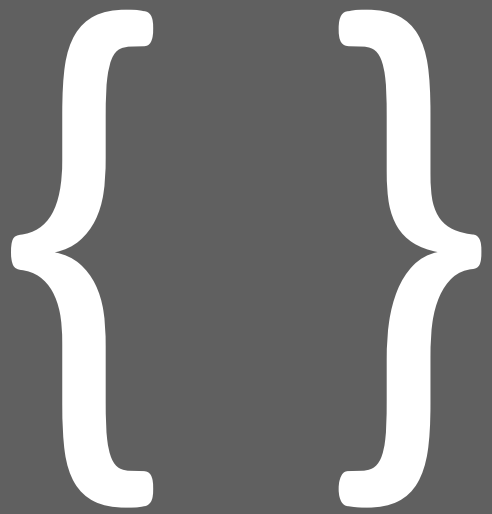
- Developers often require code to run whenever an object changes or is accessed
- When working with asynchronous code (i.e., other user collaborating remotely) this can happen often but unpredictably
- Object.observe() was originally created for this purpose but was deprecated
- Proxies allow greatly increased oversight of objects
- Control how values are changed

PROTOTYPE

- Prototype is a powerful but highly confusing feature of objects
- When a function is invoked to create an object, that object has a prototype property which is the function itself
- All instances share prototype property as reference
- Highly confusing, difficult to use and can lead to disastrous scope confusion
- To be implemented only with great caution (i.e., library implementation) and only when no better solution exists

CONCLUSION

- Objects can be created various ways – literal syntax is recommended
- Objects can have any number of properties which are easily accessed or changed
- Can be iterated with For In Loop
- Symbols are the preferred way of accessing properties internally
- Freeze and Seal both prevent object from changing
- Proxies can add further control to objects



OBJECT QUICK REFERENCE

Technique	Code
Create an object	let obj = {}
Set an object property	obj.name = “objecto”
Get an object property (bracket notation)	let name = obj.name;
Prevent new properties from being added	Object.seal(obj);
Prevent properties from changing	Object.freeze(obj)
Iterate through object properties	for (var prop in obj) {…}