the shortcut

Fullstack Web Development Tutorial Lesson 6

Today's lesson will cover

- Object copying, references
- Object methods, "this"
- Constructor functions, operator "new"

JavaScript fundamentals

Objects copying

- One of the fundamental differences of objects vs primitives is that they are stored and copied “by reference”.
- Primitive values: strings, numbers, booleans – are assigned/copied “as a whole value”
  - For instance:
    - ```
      let message = "Hello!";

      let phrase = message;
      ```
  - As a result we have two independent variables, each one is storing the string `"Hello!"`
  - Objects are not like that.
  - **A variable stores not the object itself, but its “address in memory”, in other words “a reference” to it.**
- Object is stored somewhere in memory. And the variable `user` has a “reference” to it
  - **When an object variable is copied – the reference is copied, the object is not duplicated**
- Comparison by reference: The equality `==` and strict equality `===` operators for objects work exactly the same.
  - **Two objects are equal only if they are the same object**
      ```

Objects cloning, merging and nested cloning

- Copying an object variable creates one more reference to the same object. Copying by reference is good most of the time
- However, if we do want to create independent copy or clone, we need to create a new object and replicate the structure of the existing one by iterating over its properties and copying them on the primitive level
- Also, we can use Object.assign for copying object properties
  - Syntax: `Object.assign(dest, [src1, src2, src3...])`
  - Dest is target object
  - It copies the properties of all source objects `src1, ..., srcN` into the target `dest`. In other words, properties of all arguments starting from the second are copied into the first object.
  - The call returns `dest`.
  - We also can use `Object.assign` to replace `for..in` loop for simple cloning
- Nested cloning: Properties can be references to other objects as well, not just primitive
  - Good to know, but not must understand: We should use the cloning loop that examines each value of `object[key]` and, if it's an object, then replicate its structure as well. That is called a "deep cloning"

Exercise: Objects cloning and merging

- Context: You are working on a project where you would like to have common `user` object for all users, and a separate `superUser` object to assign admin access
- Create an object `user` with properties `name: "John"` and `age: 30`
- Create another object called `superUser` with property `isAdmin: true`
- Create a clone object `adminUser` merging properties of `user` and `superUser`
- Check if the clone object has been merged properly on console

Object methods, “this”

- Objects are usually created to represent entities of the real world, like users, orders and so on. In the real world, a user can *act*: select something from the shopping cart, login, logout etc. Properties within an object which contains a function is called method
- Method shorthand may apparently work similarly but there are object inheritance differences which we will cover later
- “this” in methods:
  - **To access the object, a method can use the `this` keyword**
- “This” is not bound:
  - Keyword `this` behaves unlike most other programming languages. It can be used in any function
  - That being said, arrow functions have no “this”

Exercise: Object methods, “this”

- Create an object `calculator` with three methods
  - `read()` prompts for two values and saves them as object properties.
  - `sum()` returns the sum of saved values.
  - `multiply()` multiplies saved values and returns the result.
- ```
  let calculator = {

    // ... your code ...

  };

  calculator.read();

  alert( ‘Sum of a and b is: ‘ + calculator.sum() );

  alert( ‘Product of a and b is: ‘ + calculator.multiply() );
  ```
  ```

Object Constructor, operator “new”

- The regular object literal - `{...}` syntax allows to create one object. But often we need to create many similar objects, like multiple users or menu items and so on.
- That can be done using constructor functions and the "new" operator
- Constructor functions technically are regular functions. There are two conventions though:
  - They are named with capital letter first.
  - They should be executed only with `"new"` operator.
- The main purpose of constructors – to implement reusable object creation code
- Technically, any function can be used as a constructor. That is: any function can be run with `new`, and it will execute the algorithm above. The “capital letter first” is a common agreement, to make it clear that a function is to be run with `new`
- Return from constructors: Usually, constructors do not have a return statement. But if there is a `return` statement, then the rule is simple:
  - `return` with an object returns that object, in all other cases `this` is returned
- Methods can be used in constructor but to create complex objects, we will use Classes in future
  - Every constructor has a .prototype operator with which you can modify properties, and even add methods

Exercise: Object Constructor, operator “new”

- Use a constructor function to create an object called GeoShape with properties name, sides and sideLength
- Add a new method to the `GeoShape` class's prototype called `findPerimeter()`, which calculates its perimeter (number of sides * length of the sides- the length of the shape's outer edge) and logs the result to the console.
- Create a new instance of the `GeoShape` class called `square`. Give it a `name` of `square` and a `sideLength` of `6`.
- Call your `findPerimeter()` method on the newly created instance, to see whether it logs the calculation result to the browser DevTools' console as expected.
- Create a new instance of `GeoShape` called `triangle`, with a `name` of `triangle` and a `sideLength` of `5`.
- Call `triangle.findPerimeter()` to check that it works OK

Summary

- Objects copying and cloning:
  - Objects are assigned and copied by reference. In other words, a variable stores not the “object value”, but a “reference” (address in memory) for the value. So copying such a variable or passing it as a function argument copies that reference, not the object.
  - All operations via copied references (like adding/removing properties) are performed on the same single object.
  - To make a “real copy” (a clone) we can use `Object.assign` for the so-called “shallow copy” (nested objects are copied by reference) or a “deep cloning” function

- Object methods:
  - Functions that are stored in object properties are called “methods”
  - The value of “this” is defined at run-time. “This” has no value until function is called
  - Function can be copied between objects
  - Arrow functions are special. They have no “this”. When “this” is accessed inside arrow function, it refers to value from an outside function

- Constructors:
  - Constructor functions or, briefly, constructors, are regular functions, but there’s a common agreement to name them with capital letter first.
  - Constructor functions should only be called using `new`. Such a call implies a creation of empty `this` at the start and returning the populated one at the end.

the shortcut

Self Study Assignments

To Dos

- Continue freecodecamp Javascript. Ideally finish before we resume after summer.
- Continue with FCC HTML, CSS lessons. Ideally finish all the lessons by end of this month.
- If you need help pushing your HTML CSS project on GIthub and using Github pages let me know right away.