11.1 Lesson Plan - Hard Hat Zone: Constructors at Work

Overview

In this class we will be introducing students to the usage of JavaScript constructors and how they can be used to dynamically create objects with similar schemas.

```
Summary: Complete activities 1–4 in Unit 11
```

Instructor Priorities

- Reintroduce students to the basics of JavaScript objects (properties and methods)
- Students should understand how to create a basic JavaScript constructor
- Students should be able to use constructors and user input to dynamically create objects

Instructor's Notes

- This class is a really fun and relaxed one! Constructors are extremely useful in creating objects of similar types and allow for the development of very interesting applications. So long as your students have a firm understanding of how objects function, they should find today's lesson and activities very enjoyable.
- At the beginning of this class, make sure to go over the basics of objects once again so as to help your students recall how they are created and used. This will ultimately help them in understanding constructors better.
- If you have time at the end of class, feel free to go over JavaScript prototypes as well. They are very similar to constructors in their usage, but operate a little differently.

Class Objectives

- To master the basics of JavaScript objects
- To create basic JavaScript constructors for usage in Node applications
- To create a simple Node application which uses methods contained within a constructed object

1. Instructor Do: Welcome Class (5 mins)

- Welcome your students to class and tell them that today we will be revisiting objects and building upon our past knowledge of them to create more dynamic object-oriented applications.

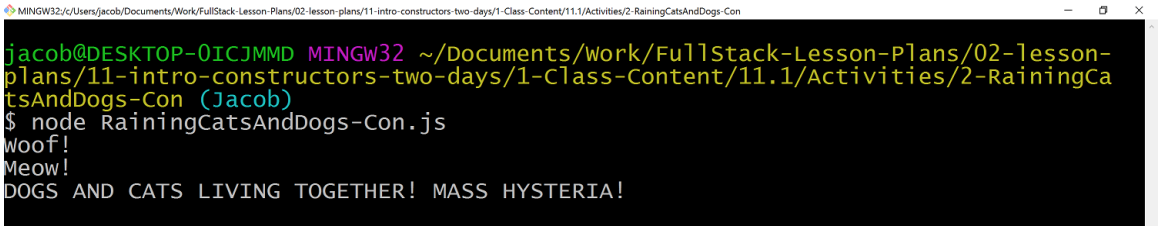
- Before diving into the new material, however, we have a small activity for them to work through beforehand to reintroduce them to objects.

2. Students Do: Raining Cats and Dogs (15 mins)

- Slack out the following instructions
- **Instructions**
  - Make a dogs object with three keys...
    - First key called "raining" with a value of true
    - Second key called "noise" with a value of "Woof!"
    - Third key called "makeNoise" which contains a function which console.logs the noise to the screen if it is raining dogs
  - Make a cats object with three keys...
    - First key called "raining" with a value of false
    - Second key called "noise" with a value of "Meow!"
    - Third key called "makeNoise" which contains a function which console.logs the noise to the screen if it is raining cats
  - Make the dog bark
  - Make the cat meow
  - BONUS: Create a function called "massHysteria" which takes in both the cats and the dogs object and prints "DOGS AND CATS LIVING TOGETHER! MASS HYSTERIA!" if both of the `raining` keys are equal to true.
  - BONUS: Look to see if you can find any means to simplify your code further and further

3. Everyone Do: Raining Cats and Dogs Demo (10 mins)

- Open up Sublime and call upon a couple students at random to come up and code out specific parts of the assignment whilst explaining their logic to the class.
- Run the completed code in Git Bash and then go over the code line-by-line with the class
  - The code should look similar to the code we have provided you with inside of the Activities folder: `rainingCatsAndDogs-NoCon.js` in `01-RainingCatsAndDogs-NoCon`
    - This is what will print out on screen...



  - Make certain to point out how we created keys and methods within the dogs and cats objects
  - Also make certain to point out how we were able to call the values/methods of our cats and dogs objects later on in the code
- Ask the class if they see anything in this code which seems redundant upon further examination

    - The cats and dogs objects have the same keys and the same overall layout. The only thing that differs between the two is the data contained within their keys.

- Ask your students if any of them came up with a foolproof way to get rid of this redundancy

    - Answer... JavaScript Constructors!

    - Another potential answer could be JavaScript prototypes, but if this solution comes up then tell the class that we may go over prototypes later on if we have time.

4. Instructor Do: Cats and Dogs, Constructed Together! (15 mins)

- Open `rainingCatsAndDogs-NoCon.js` in `02-RainingCatsAndDogs-NoCon` within Sublime and ask your students to point out the differences between this code and the one we created for the last activity. Go over the code as they discover differences.

- This code uses what's known as a constructor and it is essentially a function which can be called upon to create an object with a particular layout.

    - Be sure to point out how the first letter of the constructor is capitalized. This syntax is to make it easier for coders to distinguish constructors from their other functions.

- In order to create an object using a constructor, you would initialize a specified variable to hold the value `new Constructor()` where `Constructor()` is the function created earlier.

    - This tells the computer to create a new object using the predefined constructor schema

- Objects created using this method can then be called as per usual

    - Warn your students that they should never call a constructor function by itself. If this is done, it will create global variables for those keys within it, potentially overwriting any variables that share the same name.

5. Students Do: Character Creation (30 mins)

- See if there are any questions and then slack out the following activity

- **Instructions**

    - Over the course of this activity you are going to be using constructors to create simplistic characters for use within a very basic Roleplaying Game (RPG)

    - Each character created using your constructor should have the following properties...

        - Name: The character's name --> String

        - Profession: What the character does for a living --> String

        - Gender: The character's gender --> String

        - Age: The character's age --> Integer

        - Strength: Abstraction for how strong the character is --> Integer

        - HitPoints (HP): Abstraction for how much health the character has --> Integer

        - PrintStats: Function which prints all of a character's properties to the screen

    - Once you have created your constructor, create two new characters and print their properties to the screen

        - Fool around and get comfortable with your constructor before moving onto the next parts of the activity

    - Now that you feel comfortable with your constructor, it is time to start making this character creation system a little more reactive by adding in some more methods...

        - IsAlive: Function which prints whether or not this character is alive by looking into their hitpoints and determining whether they are above or below zero.
        - Attack: Function which takes in a second character's hitpoints and subtracts this character's strength from it.
        - LevelUp: Function which increases this character's Age by 1, their Strength by 5, and their HitPoints by 25.
    - BONUS: After completing the previous sections and making sure they work, you now have everything you need to create a very basic RPG where two characters fight one another. Don't worry if you cannot finish this part of the activity as, by completing the above sections you are well on your way to mastering constructors!

6. Everyone Do: Character Creation Summary (15 mins)

- Open up `characterCreate.js` in `03-CharacterCreate` in Sublime and start to go over the code line-by-line, making certain to point out how this constructor would allow for programmers to create any number of "Character" objects by changing up the values passed into the constructor.
- Also make certain to point out how the `Character.attack()` and `Character.levelUp()` methods can be called to alter the values contained within either object. This allows us to run these methods instead of having to create external functions to do the same thing.
- See if anyone in the class was able to tackle the bonus - no worries if no one did - and then start working alongside the class to construct a very basic RPG where two characters attack each other until one is defeated.
    - Solution to this can be found within `characterCreate-withRPG.js` in `03-CharacterCreate` in case you are low on time or no one in the class was able to come up with a solution to the problem.
    - Output of the code should look something like this...

```
Name: Crusher
Profession: Warrior
Gender: Male
Age: 25
Strength: 10
HitPoints: 5

-------------

Name: Dodger
Profession: Rogue
Gender: Female
Age: 24
Strength: 25
HitPoints: 55

-------------

Crusher is still alive!

-------------

Dodger is still alive!

-------------

Name: Crusher
Profession: Warrior
Gender: Male
Age: 25
Strength: 10
HitPoints: -20

-------------

Name: Dodger
Profession: Rogue
Gender: Female
Age: 24
Strength: 25
HitPoints: 45

-------------

Crusher has died!
```

BREAK TIME (15 mins)

7. Everyone Do: Building On Constructors (10 mins)

- Once your class has come back together after break, ask them how we might want to go about adding new and unique properties/methods to constructed objects.
    - They may not wholly understand why you would want to do this, so explain to your students how there may be some cases in which you may want constructed objects to differ in some way.
    - For example, a teacher and student object could both be constructed from a basic Programmer object, but we may want the teacher property to include a unique property called `programsToGrade` which would not be included in the student object.
- There is a very simple solution to this problem as it is EXACTLY like adding new properties and methods to regular JavaScript objects. All you have to do is assign a new property by calling the object (`student`), attach the new property to it after a period (`student.HW`), and then assign it a value (`student.HW="completed"`)

- The same would go for adding a new method except it would be declared to `function(){}`

8. Students Do: Tamagotchi Time (40 mins)

- See if there are any questions and then slack out the following activity
- **Instructions**
  - Remember Tamagotchis? They were those little toys that contained "digital pets" which you could feed, pet, play with, and care for. Kind of like Furbies but a whole lot less terrifying. Over the next thirty minutes or so, you are going to create your own basic Tamagotchi clone using constructors.
  - Create a constructor called "DigitalPal" which will create four properties and four methods...
    - The first property is "hungry" and it initially starts out as false
    - The second property is "sleepy" and it initially starts out as false
    - The third property is "bored" and it initially starts out as true
    - The fourth property is "age" and it initially starts out at 0
    - The first method is "feed()" - If hungry is true, print "That was yummy!", set hungry to false, and then set sleepy to true. If hungry is false, print "No thanks! I'm full."
    - The second method is "sleep()" - If sleepy is true, print "Zzzzzzzz", set sleepy to false, then set bored to true, and then run increaseAge(). If sleepy is false, print "No way! I'm not tired."
    - The third method is "play()" - If bored is true, print "Yay! Let's play!", set bored to false, and then set hungry to true. If bored is false, print "Not right now. Later?"
    - The fourth method is "increaseAge()" - This method runs within the sleepy() method when the DigitalPal goes to sleep and increases the DigitalPal's age by one while also printing "Happy Birthday to me! I am "+age+" old!"
  - Create a variable named "Dog" that is set to a new DigitalPal before adding the following unique properties/methods to it...
    - Outside - Initially set to false
    - Bark() - Prints out "Woof! Woof!" when run
    - goOutside() - If outside is false, prints "Yay! I love the outdoors!", sets outside to true, and runs Bark(). If outside is true, prints "We're already outside though..."
    - goInside() - If outside is true, prints "Do we have to? Fine..." and sets outside to false. If outside is false, prints "I'm already inside..."
  - Make a second variable named "Cat" that is set to a new DigitalPal and add the following methods to it:
    - HouseCondition - Initially set to 100... But not for long...
    - meow() - prints out "Meow! Meow!" when run
    - destroyFurniture() - Lowers HouseCondition by 10 and prints "MUAHAHAHAHA! TAKE THAT FURNITURE!" to the screen. Also sets bored to false and sleepy to true. If HouseCondition is equal to 0, then this should not run anymore.
    - buyNewFurniture() - Raises HouseCondition by 50 and prints "Are you sure about that?" to the screen.
  - Play around with your newly created digital pets for a bit and see what else you could add in order to make them even more exciting!

- BONUS: Make it so your Tamagotchis run off of user-input. It may sound easy at first, but this can actually be quite challenging. We will be going over this in more detail during the next class.

9. Everyone Do: Tamagotchi Summary (15 mins)

- Ask your class to raise their hands if they managed to get their "Digital Pets" to function properly and then call upon some of the more confident students to come up to the front of the class to code out the many different parts of the activity.
  - This activity has a lot of parts to it, so you should be able to get everyone to come up to code out sections of it. We would recommend having each student who comes up tackle at least one method.
- Once the code is complete, add in the code to make it so the program will run a few lines in the console and then go over it once more.

10. --EXTRA-- Instructor Do: Prototypes (15 mins)

Copyright