**Classes & Objects:**

* Open Demos>JavaScript>classes>scripts.js in your preferred text editor
* Run index.html in a live server and open the console.
* In scripts.js, create a basic object called alien. We’ll use this to remind students that objects can be created in JS without classes, but they aren’t ideal for OOP.
  + Give the alien name, health, and position keys with basic filler values
  + Give the alien functions for fly and takeDamage
* Call the fly and takeDamage methods on the alien object, then log the health and position values
  + This is a valid and simple way of creating an object, but can get tedious when you need many similar objects.
* Run the code
* Delete the current code and create a class named Alien
  + Create a constructor that takes name and health arguments and sets the name and health attributes to those arguments’ values and the position value to [0, 0]
  + Create functions for fly and take damage
* Create an instance of the Alien class
  + Call the fly and takeDamage functions
  + Log the health and position
* Everything should look the same when the code is run, but now we can more easily create more aliens
  + Demonstrate this by creating 2 more aliens

**Inheritance:**

* Open Demos>JavaScript>inheritance>scripts.js in your preferred text editor
* Run index.html in a live server and open the console.
* Show students that there are already 2 classes in the file, Alien and Player.
  + These classes represent and enemy and player character in a video game. Each has a name, health, position, and list of enemies.
  + Both classes are very similar and even have some repeated code, so they should share a parent class.
* At the top of the file, create a new Character class.
  + The constructor will take name, health, and position arguments while setting the enemies equal to an empty array.
  + Create a move function that mimics the alien’s fly function.
  + Copy the alien’s takeDamage function.
* Make the Alien class extend the Character class.
  + Remove the fly and takeDamage functions
  + Empty the constructor and add a call to the super constructor passing in the name and health arguments, and set the position to [0, 0].
* Make the Player class extend the Character class.
  + Remove the walk and takeDamage functions
  + Empty the constructor and add a call to the super constructor passing in the name argument and set the health to 5 and position to [5, 5].
* Correct all calls to the walk and fly functions to the move function and save the file.
* Everything should work the same, but now we have a lot less repeated code, and it will be easier to add more kinds of character classes later on.

**Encapsulation:**

* Open Demos>JavaScript>encapsulation>scripts.js in your preferred text editor
* Run index.html in a live server and open the console.
* Create a Customer class whose constructor takes and sets a name, age, ssn, and email.
* Create an instance of customer and pass in dummy data for all of the properties.
* Log the customer object’s name, and ssn.
* Explain to students that an ssn is a very private piece of information and our system needs to be able to keep it safe from accidentally being revealed.
* Make ssn and age private properties.
* Run the code and show that ssn is no longer printed to console. We get a warning that the field is private and cannot be accessed outside of Customer.
* We don’t want anyone to be able to access or change the ssn, but we’re fine with accessing age, so let’s give it a getter and setter method.
* In the setter method, use an if statement to throw an error message if the age is below 0, as this is not possible.
* Call the setter method and pass in a negative number to show students that we get the error.
* Call the getter method and show students that the age is still set to what we originally passed in.

**Polymorphism:**

* Open Demos>JavaScript>polymorphism>scripts.js in your preferred text editor.
* Run index.html in a live server and open the console.
* Create a Student class with private name and classesAndGrades variables.
* Set the classesAndGrades variable to an empty array so that you can use array method and properties later.
* Create a constructor that only takes and sets the name.
* Create getters for each variable.
* Create a getGPA method
  + Create a block scope variable called gpa
  + Loop through the objects in the classesAndGrades array and create a switch case that adds a number to the gpa variable based off a letter grade
    - Ex. case “A”: gpa+= 4;
  + After the loop, set the gpa equal to itself divided by the number of classesAndGrades
* Create a method to add a classAndGrade object to the classesAndGrades array
  + Ex. this.#classesAndGrades.push(classAndGrade)
* Initialize a student object and add a few classes with grades.
  + Ex. student.addClassAndGrade({class: “Math”, grade: “A”})
* Console log the students gpa
* GPAs are calculated higher for students in Advanced Placement (college level) classes, but everything else is the same.
* Create an APStudent class that extends the Student class.
  + Copy the Student getGPA method and be sure you’re calling this.getClassesAndGrades to access the parent’s private array
  + Update the switch cases to return 1 extra point per grade letter (“A” = 5, “B” = 4, etc.)
* Update your Student object to be an APStudent object and reload the code. The GPA should be 1 point higher now.

**TypeScript:**

* Open Demos>JavaScript>typescript in VS Code.
* Open the terminal and run **npm i --save-dev typescript** and wait for everything to install.
* Create a new file called **.gitignore** and add **node\_modules** to the file so that the node\_modules folder is not tracked in git.
* Create a new file called **tsconfig.json** and add the following to the file:
  + {  
     “compilerOptions”: {  
     “target”: “ES6”,  
     “module”: “CommonJS”,  
     “sourceMap”: false  
     }  
    }
* Click on the **Terminal** dropdown menu and select **Configure Default Build Task**
* Select **tsc: watch**
* A new file called **tasks.json** will be created inside of a new folder called .vscode
* (If you are running git bash as your default CLI in VS Code) Add the following lines to the tasks.json file after option and before problemMatcher:
  + "options": { "shell": { "executable": "powershell.exe" } }
* Create a **scripts.ts** file.
* Create a Character class with private fields name (type string), healthPoints, and attackPower (both type number).
  + Create a constructor that accepts and sets all fields.
  + Create getters for each field and a setter for healthPoints.
  + Create a function called attack that takes a target (type Character) and a weapon (type Weapon)
  + Set the target’s HP to their current HP minus the attacker’s attackPower
  + Log `${this.getName()} attacked ${target.getName()} with a ${weapon}`
* Above the Character class, create a type Weapon that accepts certain strings that contain weapons.
  + Ex. "Rusty Sword" | "Sword" | "Bow and Arrow" | "Mace" | "Wand"
* Create a class called Player that extends Character and has a private weaponsList field that accepts an array of weapons and is set to an empty array.
  + Create a constructor that only takes the name and calls super() passing in the name, 100 HP, and 20 attackPower.
  + In the constructor, push a starter Weapon into the weaponsList.
  + Create a getter for the weaponsList and a function to pickup a weapon to add to the list.
* Create a class called Enemy that extends Character and has a private weapon field of type Weapon.
  + Create a constructor that only takes the name and weapon and calls super() passing in the name, 100 HP, and 10 attackPower.
  + In the constructor, also set the weapon to the passed in weapon.
  + Create a getter for the weapon.
  + Override the attack method to take only the target Character and call the super.attack() method passing in the target and this.#weapon.
* Instantiate a Player and an Enemy.
* Have the Player pickup a new weapon and then attack the enemy with it.
* Log the enemy’s new HP.
* Press **ctrl + shift + B** to ensure that VS Code is watching compiling the code.
* Run **node scripts.js** in the CLI.