* The **console** is a tool that developers use to record the output of their JavaScript programs. The **console.log()**command is used to print, or log, text to the console. **console.log("Hello!");**
* Data type ***Null*** — Can only be null. It represents the absence of value.
* add .property to an instance of an object to return the value of that property e.g. **console.log(“Hello”.length);** //5. You can also add .method() to an instance to return values on the object. methods can be built in or custom made. e.g. **console.log(“Hey”.startsWith());** //H
* Libraries contain methods that you can call without creating an instance e.g. **console.log(Math.random()\*50);** //returns random number 0-50
* **const**, short for constant, is a JavaScript *keyword* that creates a new variable with a value that cannot change.
* You can create and reassign variables that you create with the **let** keyword.
* ***Undefined*** data type is assigned to variables that are not yet assigned a value.
* String interpolation in ES6 version of javascript: **let myPet = 'armadillo' console.log(`I own a pet ${myPet}.`)** this uses backticks instead of quotes and plus symbol
* If it exists but contains one of the following values then it is **falsy** meaning it will evaluate to false when used as conditional logic: ***false, 0 and -0, "" and '' (empty strings), null, undefined, NaN (Not a Number), document.all***
* To check if two things equal each other, we write **===**. To check if two things *do not* equal each other, we write !==. Use a single = to assign a value to a variable. Use ===to compare the values of two different variables.
* Logical operators: To say "both must be true," we use && and To say "either can be true," we use ||.
* A *function declaration* is a function that is bound to an identifier or name. **function square (number) { return number \* number; }** Function declarations require the keyword function, a name, and a function body.
* *arrow function* syntax: **variableName = () => {...}** this is a variable that stores a function
* With a *function expression* the identifier can be omitted, creating an anonymous function. Function expressions are often stored in a variable. **const square = function (number) { return number \* number; };** Also note function expressions end with a semi-colon since they are stored in a variable.
* .**push()**allows us to add items to the end of an array. **arrayName.push('item 3', 'item 4');** Similarly **.pop(),** removes the last item of an array. **arrayName.pop();**
* Variables that contain arrays can be declared with **let** or **const**. Even when declared with const, arrays are still mutable; they can be changed. However, a variable declared with const cannot be reassigned.
* methods called *iterators*, are called on arrays and complete such tasks as altering each element and selecting elements that fit certain criteria.
* **.forEach()**will execute the same code on each element of an array. **arrayName.forEach(function(item) { console.log(' - ' + item); });** alternatively you can use: **arrayname.forEach(item => console.log(' - ' + item));** .forEach() method: is an array method. It must be called upon an array. Any changes to the iterated array value won't be updated in the original array.The return value is undefined.
* **.map()**works just like .forEach but it returns a new array with elements that have been modified by the code in its block. **let bigNumbers = numbers.map(function(number) { return number \* 10; });** or alternatively: **let bigNumbers = numbers.map(numbers => numbers \* 10);**
* **.filter()**returns a new array. However, .filter() returns certain elements from the original array that evaluate to truthy based on conditions written in the block of the method. **let shortWords = words.filter(function(word) { return word.length < 6; });** or **let shortWords = words.filter(word => word.length < 6);**
* JavaScript ***objects***are containers that can store data and functions. The data we store in an object is not ordered — we can only access it by calling its associated *key*. **let restaurant = { name: 'Italian Bistro', seatingCapacity: 120, hasDineInSpecial: true, entrees: ['Penne alla Bolognese', 'Chicken Cacciatore', 'Linguine Pesto'] };**
* Properties in objects are separated by commas. Key-value pairs are always separated by a colon.
* To access the properties within an object, we connect the value's name to the key name with a period: **restaurant.entrees.** Another way to access a key's value is with *bracket notation*. **restaurant['entrees'].** One advantage that bracket notation has over dot notation is that you can use variables inside the brackets to select the keys of anobject.
* You can add more properties to an object by using **restaurant['appetizers'] = ['Fried Calamari', 'Bruschetta'];** or **restaurant.desserts = ['Homemade Tiramisu', 'Cannoli'];** Use the same to edit an existing property
* When objects have key-function pairs, we call the function a ***method*.** **const restaurant = { openRestaurant: () => { return 'Unlock the door, flip the open sign. We are open for business!'; }};**
* In the new version of javascript functions are written without arrow syntax or colon like this: **openRestaurant() { return 'Unlock the door, flip the open sign. We are open for business!'; }**
* use the **this** keyword to access properties inside of the same object. It is a dynamic variable that can change depending on the object that is calling the method.
* Getter and setter methods get and set the properties inside of an object.
* Developers use an underscore before a property name to indicate a property or value should not be modified directly by other code.
* Before changing the value of an object key it might be safe to first check if the value is valid, then after confirming all ow the value to be set. This should be done in a setter method: **set seatingCapacity(newCapacity) { if (typeof newCapacity === 'number') { this.\_seatingCapacity = newCapacity; console.log(`${newCapacity} is valid input.`); } else { console.log(`Change ${newCapacity} to a number.`)**