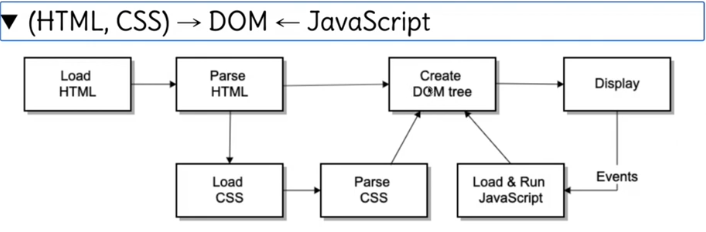
JavaScript



JS wont work until you have a DOM model.



Objects do things, defined as methods or functions and functions receive parameters to do what needs to be done.



Const give an area of memory a label, that can be referred to. It is a keyword in JS that tells code to reserve and area of memory and label that area of memory ‘my paragraph’.



The functions are within the constant. The ‘.’

Operator

# Object

Label

Keyword

Valid java code:

content



; end of statement.

Key word const. refeference to an area of memory

You are manipulating the address space, which is memory with code/data that is going to be executed.

{} = content of an object, whatever is the result of this, put it inside the ‘label’.

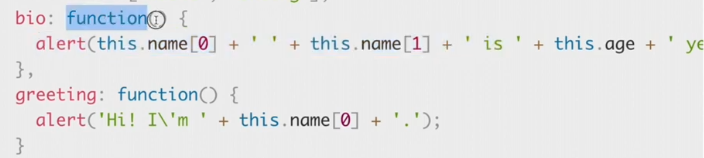


Array

Literal string

integer

* Const person all within {};
* Line 1,2,3,4 similar, all start with a label, colon which is the delimiter, and some data.
* Line 2 and 5. Literal strings separated by commas inside a square bracket. This name is an ‘Array’. Arrays have values inside them separated by commas.
* The data inside the label are basic data types.
* Line 2 to 5 are all attributes of the object ‘person’.



* Functions always has parenthesis (), which implies you cannot pass and receive parameters to do what they need to do.
*  - (“p”) is a parameter that has been passed into the function queryselector.
*  This is a function with no parameters.
* This is a function with parameters passed inside of the ().
* Functions can call each other; they can be nested. Functions can have other functions within them.
* This greeting function has an alert function with parameters passed inside of it.
* person.bio will result in this:



* person.greeting will result in this:



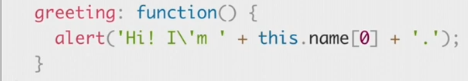
You can say .queryselector (“p”) and it will have a query selector defined and it will receive ‘p’ as the parameter and go and pick out the particular branch within the data structure that’s a paragraph.

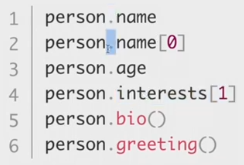


Index location

String

Operator

* this. refers to the attributes contained within the object. A scope in terms of memory, this area within memory cannot be changed by code outside of the block of code in const person.
* Refers to the name defined within this particular object. If there was a global name defined outside of this scope, that name is in a different area of memory than the name in this object. So, this. Allows you to refer to the name defined in this scope.
* People say global variables bad to use because if you had something defined outside of a particular object that is available to all objects, this value can change in memory by any line of code.
* 
* This will be called by person.greeting and will concatenate this sentence in a display:
* Hi! I’m Bob.
* \ allows the literal punctuation ‘ to be used in a string.
* ‘ this is a special character in JS.



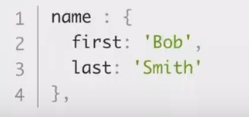
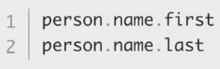
* All of these are now valid and can be called.
*  we know these are functions because of the () and the can be called.
*  you know that’s an index for an array.
*  you can call an attribute of the object.

# Sub-namespaces

It is even possible to make the value of an object another object. For example, changing the name attribute from:

* 

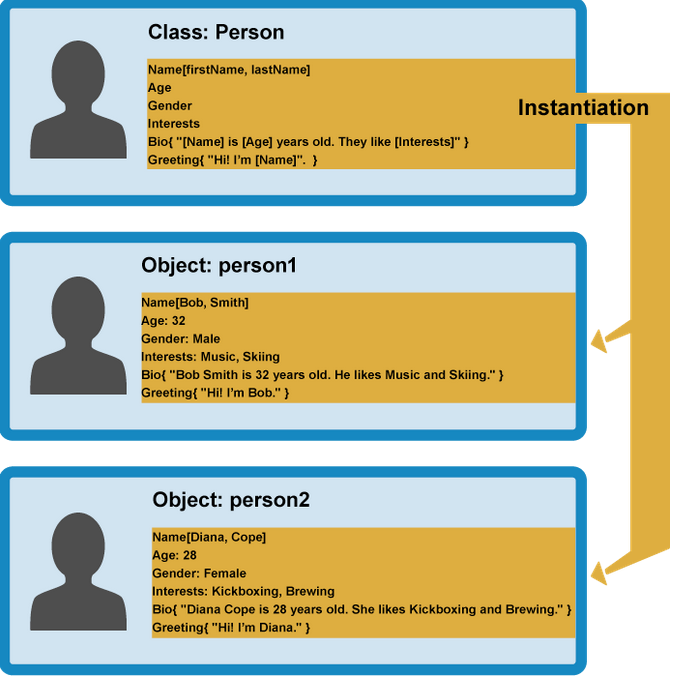
To

*  This.
* This way is more object orientated.
* This can then be accessed like this: 

# Object oriented programming

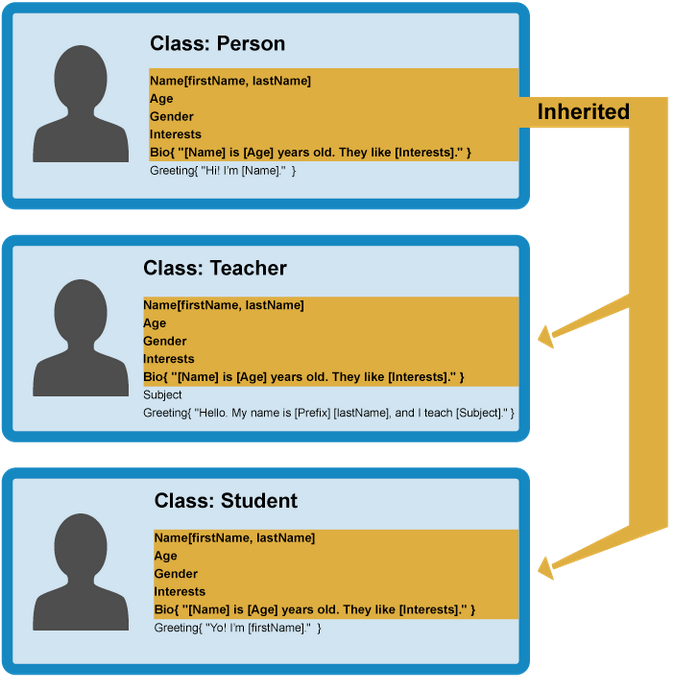
The basic idea of OOP is that we use objects to model real world things that we want to represent inside our programs, and/or provide a simple way to access functionality that would otherwise be hard or impossible to make use of.

Objects can contain related data and code, which represent information about the thing you are trying to model, and functionality or behavior that you want it to have. Object data (and often, functions too) can be stored neatly (the official word is **encapsulated**) inside an object package (which can be given a specific name to refer to, which is sometimes called a **namespace**), making it easy to structure and access; objects are also commonly used as data stores that can be easily sent across the network.

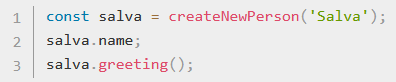


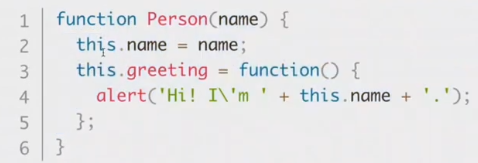
* We have a class person that has a strict set of attributes. When you create an object from this, you can instantiate person 1, person 2, person 3 based on the original prototype. This is what object instantiation is.
* Say you have 10 people in a musician group and two more join, you need two more objects. You do not go and rewrite the class. You have one definition a musician and create musician one, two, three. Create a new scope in memory.
* This can happen at run time. You can read in the value passed to a function. give 10 definitions of musicians.
* You have the basic definition and uses of It In memory.
* This implies there must be a sort of constructor.
* Class has a constructor.
* That instructors job is to instantiate new objects of that type.

# Object inheritance

* 
* There are situations where you say person but now, they are defined as either teacher or student. This is not instantiation it is inheritance.
* All persons have a name and age, they have common attributes, but each one has something different in them.
* Inheritance is part of OOP.
* Inherits from the base and adds in extra attributes.

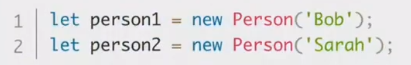


* Function with name passed as parameter. There is an area in memory that has a value for that name and that is passed to this function.
* We have a top level empty object. We define references to this and add attributes/functions to it.
* 
* You can now create a new person by calling this function
* 
* Creates a constant.
* Give it a name, salva.
* Execute the createnewperson function and pass salva to it.
* Returns a new object.

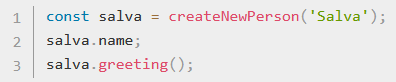


This code is much more clearer as is saves you having to define object name, rather you just use ‘this’ and it already knows the scope.

You can now use this:

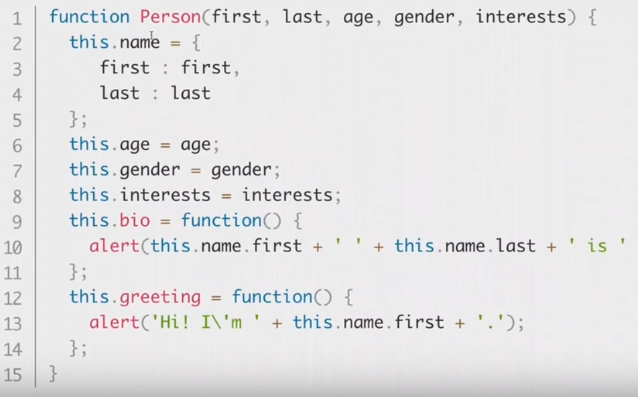


Instead of:

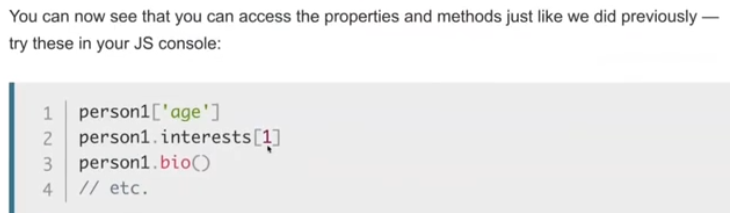


Notice that it is not cons, it is let.

General rule is define everything as a ‘cons’ unless you want to change it, then use ‘let’. Let replaces

A more realistic example. 

* All coded using the dot notation (more object oriented).
* Name is not an array. It is coded by making it an object. Embedding objects within a scope.
* 
* This will be called to create an object instance from the function person. Pass in the parameters.
* This can be put in a loop and can add loads of persons.



# Events

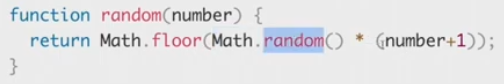
**events** are actions or occurrences that happen in the system you are programming

Each available event has an **event handler**, which is a block of code (usually a JavaScript function that you as a programmer create) that runs when the event fires. When such a block of code is defined to run in response to an event, we say we are **registering an event handler**.

# Event example



* Line 1 – defines a reference to this object, that has a method, which receives a function, that contains the parameter button.
* The querySelector() method returns the first element that matches a specified CSS selector(s) in the document. In this case, button.
*  btn refers to the <button> node in the DOM.



This function randoms, where you pass a number in the () and using library math, these methods are available so nothing needs to be defined.

* generating a random number and + 1 to it. This returs a random number.

# Event handling example



* Object btn has a whole set of methods and events associated with it.
* Inside the btn.onclick function, there will be objects like window that allows you to get data like x y coordinate of the cursor, and other code to check whether you are on the button or not.
* Once that event happens, then execute the function.
* Line 8 defines a random colour in a constant. Is a label that contains a string. This contains the valid CSS declaration.
* A more beneficial way to code this is:
* 
* This is better because if you wanted to call bgchange to other elements associated with changing background colour, you would use the bgchange function, rather than coding it all again.
* The result can be = to.
* Why define twice when you can define once and call whenever.
* In more complex code, where you want the same thing to happen more than once, then you should only define the function once, then call it whenever it is needed.