**Objects**

Objects are containters storing related data and functionality. They are like dictionaries in Python.

Remember commas between properties and methods!

1. **Creating Object Literals (Object classes)**

Objects can be assigned to variables just like any JS type. We use curly braces, {}, to designate an *object literal*:

let spaceship = {key:value}; // spaceship is an object

- A key’s value can be of any data type, including functions or other objects.

- When we have a key that does not have any special characters in it (including spaces), JavaScript allows us to omit the quotation marks

- **The keys are going to be both property and method names.**

1. **Accessing Properties**

We use the dot notation to access properties whose names are strings without special characters.

Diagram

Description automatically generated

We MUST use bracket notation (like indexing) when accessing keys that have numbers, spaces, or special characters in them. We must also use brackets, for variables inside it to help us choose which keys we want to access.

Diagram

Description automatically generated

let returnAnyProp = (objectName, propName) => objectName[propName];  
returnAnyProp(spaceship, 'homePlanet'); // Returns spaceship[‘homePlanet’]

Accessing unexisting properties will return undefined.

1. **Changing/Adding/Deleting Properties**

We can add/change properties by calling keys directly and assign values. If no such key exist, it is added.

We can delete a property from an object with the delete operator: delete <object>[<keyName>];

1. **Methods**

We can include methods in our object literals by creating key-value pairs.

**- Keys are method’s name** ( <methodName> : function () {…} )

**- Values are anonymous function expression**

With the new syntax, we can omit the ‘:’ and the ‘function’ keyword

const alienShip = {  
  invade () {   
    console.log('Hello! We have come to dominate your planet. Instead of Earth, it shall be called New Xaculon.')  
  }  
};

1. **Nested Objects**

We can include other objects in an object, with keys being the ‘nested object’s identifier’, and values being ‘{}’, which contains that object’s properties and methods.

Text

Description automatically generated

1. **Pass By Reference**

Objects are passed by reference. This means that when we pass the variable assigned to an object into a function as an argument, **we can change the values of the properties** of that object globally, through function.

However, we cannot reassign the object through functions. The reason is that we passed variable assigned to the object, which contains the memory location of the object properties, not the memory location of the variable.

🡪 We pass by reference the object’s properties, but passed by value the object itself.

Text

Description automatically generated

1. **Looping Through Objects**

Since the key-value pairs in objects are unordered, we **loop through keys** in JavaScript **using for…in**

**for (<initialize a variable> in <object literal/nested object>){**

**//statement;**

**}**

**Text

Description automatically generated**

|  |
| --- |
| **NOTE:** be careful when accessing objects, or values, in loops. In this example, each crewMember (looping variables) are the **keys** representing nested objects (captain, medic, translator). But we CANNOT access the values associated with these keys from key names. We need to reference these keys from the object literals: object\_literal[object].key 🡪 value  NOT obejct.key --X> value |