**ArrayBuffer**

 The basic binary object is ArrayBuffer – a reference to a fixed-length contiguous memory area.  
  
-It has a fixed length, we can’t increase  
or decrease it.  
-It takes exactly that much space in the  
memory.  
-To access individual bytes, another  
“view” object is needed, not  
buffer[index].  
  
To manipulate an ArrayBuffer, we need to use a “view” object.  
A view object does not store anything on its own. It’s the “eyeglasses” that give an

**TypedArray**

 TypedArray is a group of array-like objects in JavaScript that provide a way to read and write binary data in a structured manner.

The common term for all these views (Uint8Array, Uint32Array etc) is TypedArray. They share the same set of methods and properties.  
  
They are much more like regular arrays: have indexes and iterable.  
  
To access the ArrayBuffer, there are properties:  
  
arr.buffer – references the ArrayBuffer.  
arr.byteLength – the length of the ArrayBuffer.

**Out-of-bounds behaviour**

 Out-of-bounds behavior refers to what happens when you access or manipulate elements of an array or memory outside the valid or allocated range.  
  
What if we attempt to write an out-of-bounds value into a typed array?  
  
Uint8ClampedArray is special in this aspect, its behavior is different. It saves 255 for any number that is greater than 255, and 0 for any negative number. That behavior is useful for image processing.

**TypedArray methods**

 TypedArray is a group of array-like objects in JavaScript that provide a way to read and write binary data in a structured manner.  
  
There are different types of TypedArray objects, including Uint8Array, Int16Array, Float32Array, etc., which correspond to different data types.  
  
You can create a TypedArray that references an ArrayBuffer to access and manipulate its data.  
TypedArray has regular Array methods, with notable exceptions.  
  
We can iterate, map, slice, find, reduce,  
  
Things we can’t do though:  
1. No splice – we can’t “delete” a value, because typed arrays are views on a  
buffer, and these are fixed,contiguous areas of memory. All we  
can do is to assign a zero.  
2. No concat method.  
  
There are two additional methods that allows us to copy typed arrays, mix them, create new arrays from existing ones, and so on  
  
arr.set(fromArr, [offset]) copies all elements from fromArr to the arr, starting at position offset (0 by default).  
  
arr.subarray([begin, end]) creates a new view of the same type from begin to end (exclusive). That’s similar to slice method (that’s also supported), but doesn’t copy anything – just creates a new view, to operate on the given piece of data.

**DataView**

 DataView is another way to read and write binary data in an ArrayBuffer, allowing more fine-grained control over byte order and data types.  
  
It provides methods to read and write data of various types (e.g., getUint8, getInt32, setFloat64) at arbitrary byte offsets within the buffer.