Objects & Arrays

Objects: {}

- property access
- bracket notation
- dot notation
- dot vs. bracket
- nested objects
- object literals
- iteration

property access

```
var container = {};

container.material = 'plastic';

// can also be represented as
container = {
    material: 'plastic'
};
```

Anything to the left of the dot is an object. Dot notation is used to assign a property, and dot notation is used to access a property.

```
var container = {};

container.material = 'plastic';
container.material; // ?
```

access with dots:

```
var container = {};
container.material = 'plastic';
var box = container.material;
box; // ?
container.material = 'wood';
box; // ?
```

access and assignment:

```
var container = {};
container.material = 'plastic';
container.material; // 'plastic'
container.size; // ?
```

bracket notation:

```
var container = {};
container['material'] = 'plastic';
container.material; // ?
```

bracket notation:

```
var container = {};
container['material'] = 'plastic';
container['material']; // ?
```

bracket notation:

```
var container = {};
container['material'] = 'plastic';
var box = container['material'];
box; // ?
```

bracket notation: variables

```
var container = {};
container['material'] = 'plastic';
var key = 'material';
container[key]; // ?
```

bracket notation is required when using variables to access the property on an object - key is no quotes

```
container['key']; // ?
```

bracket notation: allows the evaluation of expressions which aren't possible with dot notation

```
var container = {};
container['material'] = 'plastic';
var fn = function () {
    return 'material';
};
container[fn()]; // ?
```

object best practices (do's and don't's)

```
var container = {};
container['material'] = 'plastic';
var key = 'material';
container['key']; // ?
container.key; // ?
container[key]; // ?
```

don't use dot notation with a variable

don't use quotations around your variable

```
var container = {};
container['material'] = 'plastic';
var key = 'material';
container['key']; // undefined
container.key; // undefined
container[key]; // 'plastic'
```

non-valid characters: dot notation requires valid variable names - to use characters that are 'invalid' for variable names as property names, wrap in them in quotes and use bracket notation

```
var container = {};
container['material'] = 'plastic';
container[0] = 'spider';
container['%^&'] = 'cuckoo stuff';
var test = container['%^&'];
test; // ?
```

when using bracket notation, the property name is stringified under the hood (even if it's a number)

```
container = {
    'material': 'plastic',
    '0': 'spider',
    '%^&': 'cuckoo stuff'
};
```

storing data: any kind of data can be stored on an object, including other objects or functions (methods)

```
var container = {};
container['material'] = 'plastic';
container['size'] = {
    length: 120,
    width: 80,
    height: 4
};
container.getVolume = function () {
    return container.size.length * container.size.width * container.size.height
};
```

bracket notation vs. dot notation

When would you use one vs. the other? Generally speaking use dot notation unless you need to use a variable for a property of an object.

object literals: in object literal notation, property names can be strings, but not expressions

```
// given the following
var container = {};
container['size'] = 12;
container['~/["7'] = 'chair';
container['size']; // 12
container['~/["7']; // 'chair'

// object literal notation
var container = {
    'size': 12,
    '~/["7': 'chair'
};
```

iteration: use the for in loop - no guaranteed order

```
var container = {};
container['material'] = 'plastic';
container[0] = 'spider';
container['%^&'] = 'cuckoo stuff';

// to see the keys (var for "key" can be anything)
for (var key in container) {
    console.log(key);
}

// to see the values
for (var key in container) {
    console.log(container[key]);
}
```

iteration

```
var container = {};
container['material'] = 'plastic';
container[0] = 'spider';
container['%^&'] = 'cuckoo stuff';

for (var key in container) {
    console.log(container.key); // undefined
}
```

Arrays: []

- arrays vs. objects
- access and assignment
- native methods and properties
- iteration

Use arrays when order matters and named properties don't. Additionally, when the properties and methods available on an array are desired.

access and assignment

```
var container = [];
container[0] = true;
container[1] = 'spider';
container.push({ status: 'closed' });
var i = 0;
box[i]; // ?
box[1]; // ?
box.pop(); // ?

// can also be written as
container = [true, 'spider', { status: 'closed' }];

// after the last pop()
console.log(container); // [true, 'spider']
```

access and assignment: arrays are objects

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
container['size']; // ?
```

access and assignment: properties can be assigned and accessed in the same fashion (bracket notation or dot notation) as any other object

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
container[0]; // 'spider'
container['size']; // 12
container.0; // ?
```

iteration

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';

for (var key in container) {
    console.log(key); // ?
}
```

iteration: to belabor the point, there is no difference between the two following

```
var container = [];
var container2 = {};
container.full = true;
container2.full = true;
```

iteration: to iterate over the properties assigned to an array, for in can be used

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
for (var key in container) {
    console.log(container[key]); // ?
}
```

iteration: to iterate over the properties assigned to an array, for in can be used

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
for (var key in container) {
    console.log(container.key); // ?
}
```

When using an array we're usually not interested in those named properties, therefore we don't want to use a for in loop to iterate over the array.

iteration: there is no relationship between a for loop and an array, it is simply better suited for array iteration

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
container.push('Yowza!');

for (var i = 0, l = container.length; i < l; i += 1) {
    console.log(i); // ?
}
```

iteration: there is no relationship between a for loop and an array, it is simply better suited for array iteration

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
container.push('Yowza!');

for (var i = 0, l = container.length; i < l; i += 1) {
    console.log(container[i]); // ?
}
```

iteration

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
container.push('Yowza!');
container; // ['spider', 'Yowza!'];
console.log(container); // ['spider', 'Yowza!', size: 12]
```

native properties: what makes arrays special are the properties and methods that come with it, not the numerical indices themselves

```
var container = [];
container['size'] = 12;
container['0'] = 'spider';
container.length; // ?
```

native properties: what makes arrays special are the properties and methods that come with it, not the numerical indices themselves

```
var container = [];
container['0'] = 'spider';
container[3] = { largeContainer: false };
container['length']; // ?
```

native properties: what makes arrays special are the properties and methods that come with it, not the numerical indices themselves

```
var container = [];
container['0'] = 'spider';
container[5] = true;
container[length]; // ?
```

arrays demonstrate one of the advantages of using bracket notation for properties

```
// to view the last item in the array
container[container.length - 1];
```