JAVASCRIPT PT. 2

ANDREAS DRANIDIS

OBJECTS

An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where key is a string (also called a “property name”), and value can be anything.

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax

let user = {     // an object
  name: "John",  // by key "name" store value "John"
  age: 30,       // by key "age" store value 30
};
```

In the user object, there are two properties: The first property has the name "name" and the value "John". The second one has the name "age" and the value 30.

```
// get property values of the object:
alert( user.name ); // John
alert( user.age ); // 30

user.isAdmin = true;

delete user.age;
```

SQUARE BRACKETS

```
// this would give a syntax error
user.likes birds = true

// set
user["likes birds"] = true;

// get
alert(user["likes birds"]); // true

// delete
delete user["likes birds"];

let key = "likes birds";

// same as user["likes birds"] = true;
user[key] = true;
```

PROPERTY VALUE SHORTHAND

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...other properties
  };
}

let user = makeUser("John", 30);
alert(user.name); // John

function makeUser(name, age) {
  return {
    name, // same as name: name
    age,  // same as age: age
    // ...
  };
}
```

PROPERTY EXISTENCE TEST, “IN” OPERATOR

```
let user = {};

alert( user.noSuchProperty === undefined ); // true means "no such property"

let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist

let obj = {
  test: undefined
};

alert( obj.test ); // it's undefined, so - no such property?

alert( "test" in obj ); // true, the property does exist!
```

THE “FOR…IN” LOOP

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key );  // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
}
```

```
let codes = {
  "49": "Germany",
  "41": "Switzerland",
  "44": "Great Britain",
  // ..,
  "1": "USA"
};

for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}
```

OBJECT REFERENCES

A variable assigned to an object stores not the object itself, but its “address in memory” – in other words “a reference” to it.

```
let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // changed by the "admin" reference

alert(user.name); // 'Pete', changes are seen from the "user" reference
```

Comparison by reference

```
let a = {};
let b = a; // copy the reference

alert( a == b ); // true, both variables reference the same object
alert( a === b ); // true
```

```
let a = {};
let b = {}; // two independent objects

alert( a == b ); // false
```

OBJECT METHODS, "THIS"

```
user = {
  sayHi: function() {
    alert("Hello");
  }
};

// method shorthand looks better, right?
user = {
  sayHi() { // same as "sayHi: function(){...}"
    alert("Hello");
  }
};
```

It’s common that an object method needs to access the information stored in the object to do its job.
For instance, the code inside user.sayHi() may need the name of the user.
To access the object, a method can use the this keyword.
The value of this is the object “before dot”, the one used to call the method.

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }

};

user.sayHi(); // John
```

“THIS” IS NOT BOUND

```
function sayHi() {
  alert( this.name );
}
```

The value of this is evaluated during the run-time, depending on the context.

```
let user = { name: "John" };
let admin = { name: "Admin" };

function sayHi() {
  alert( this.name );
}

// use the same function in two objects
user.f = sayHi;
admin.f = sayHi;

// these calls have different this
// "this" inside the function is the object "before the dot"
user.f(); // John  (this == user)
admin.f(); // Admin  (this == admin)

admin['f'](); // Admin (dot or square brackets access the method – doesn't matter)
```

The concept of run-time evaluated this has both pluses and minuses. On the one hand, a function can be reused for different objects. On the other hand, the greater flexibility creates more possibilities for mistakes.

ARROW FUNCTIONS HAVE NO “THIS”

```
let user = {
  firstName: "Ilya",
  sayHi() {
    let arrow = () => alert(this.firstName);
    arrow();
  }
};

user.sayHi(); // Ilya
```

CONSTRUCTOR, OPERATOR "NEW"

Constructor functions technically are regular functions. There are two conventions though:

- They are named with capital letter first.
- They should be executed only with "new" operator.

```
function User(name) {
  this.name = name;
  this.isAdmin = false;
}

let user = new User("Jack");

alert(user.name); // Jack
alert(user.isAdmin); // false
```

```
function User(name) {
  // this = {};  (implicitly)

  // add properties to this
  this.name = name;
  this.isAdmin = false;

  // return this;  (implicitly)
}
```

The main purpose of constructors – to implement reusable object creation code.

METHODS OF PRIMITIVES

One of the best things about objects is that we can store a function as one of its properties.

```
let john = {
  name: "John",
  sayHi: function() {
    alert("Hi buddy!");
  }
};

john.sayHi(); // Hi buddy!
```

Objects are “heavier” than primitives. They require additional resources to support the internal machinery.

- Primitives are still primitive. A single value, as desired.
- The language allows access to methods and properties of strings, numbers, booleans and symbols.
- In order for that to work, a special “object wrapper” that provides the extra functionality is created, and then is destroyed.

```
let str = "Hello";

alert( str.toUpperCase() ); // HELLO
```

NUMBERS

toString(base)

```
let num = 255;

alert( num.toString(16) );  // ff
alert( num.toString(2) );   // 11111111
```

Math.floor

Rounds down: 3.1 becomes 3, and -1.1 becomes -2.

Math.ceil

Rounds up: 3.1 becomes 4, and -1.1 becomes -1.

Math.round

Rounds to the nearest integer: 3.1 becomes 3, 3.6 becomes 4, the middle case: 3.5 rounds up to 4 too.

```
alert( parseInt('100px') ); // 100
alert( parseFloat('12.5em') ); // 12.5

alert( parseInt('12.3') ); // 12, only the integer part is returned
alert( parseFloat('12.3.4') ); // 12.3, the second point stops the reading
```

STRINGS

Quotes

```
let single = 'single-quoted';
let double = "double-quoted";

let backticks = `backticks`;
```

```
function sum(a, b) {
  return a + b;
}

alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
```

```
let str1 = "Hello\nWorld"; // two lines using a "newline symbol"

// two lines using a normal newline and backticks
let str2 = `Hello
World`;

alert(str1 == str2); // true
```

STRINGS

```
alert( `My\n`.length ); // 3
```

```
let str = `Hello`;

// the first character
alert( str[0] ); // H
alert( str.charAt(0) ); // H

// the last character
alert( str[str.length - 1] ); // o
```

```
alert( 'Interface'.toUpperCase() ); // INTERFACE
alert( 'Interface'.toLowerCase() ); // interface
alert( 'Interface'[0].toLowerCase() ); // 'i'
```

```
alert( "Widget with id".includes("Widget") ); // true
alert( "Hello".includes("Bye") ); // false
let str = "stringify";

// start at the 4th position from the right, end at the 1st from the right
alert( str.slice(-4, -1) ); // 'gif'
```

ARRAYS

```
let arr = new Array();
let arr = [];
```

```
let fruits = ["Apple", "Orange", "Plum"];
fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]
fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]
alert( fruits.length ); // 4
alert( fruits ); // Apple,Orange,Plum, Lemon
alert( fruits.pop() ); // remove "Lemon" and alert it
fruits.push("Pear"); // Apple,Orange,Plum, Lemon, Pear
alert( fruits.shift() ); // remove Apple and alert it
fruits.unshift('Apple'); // Apple, Orange, Pear
```

```
for (let fruit of fruits) {
  alert( fruit );
}
```

ARRAYS

```
arr.forEach(function(item, index, array) {
  // ... do something with item
});
```

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`);
});
```