Prototypes - Summary

Prototypes can be a confusing and tricky topic - that's why it's important to really understand them.

A prototype is an object (let's call it `"P"`) that is linked to another object (let's call it `"O"`) - it (the prototype object) kind of acts as a "**fallback object**" to which the other object (`"O"`) can reach out if you try to work with a property or method that's not defined on the object (`"O"`) itself.

EVERY object in JavaScript by default has such a fallback object (i.e. a prototype object) - more on that in the next lectures.

It can be especially confusing when we look at how you configure the prototype objects for "to be created" objects based on constructor functions (that is done via the `.prototype` property of the constructor function object).

Consider this example:

```
function User() {
    ... // some logic, doesn't matter => configures which properties
etc. user objects will have
}
User.prototype = { age: 30 }; // sets prototype object for "to be
created" user objects, NOT for User function object
```

The `User` function here also has a prototype object of course (i.e. a connected fallback object) - **but that is NOT the object the `prototype` property points at**. Instead, you access the connected fallback/ prototype object via the **special `__proto__` property** which EVERY object (remember, functions are objects) has.

The `prototype` property does something different: **It sets the prototype object, which new objects created with this User constructor function will have.**

That means:

```
const userA = new User();
userA.__proto__ === User.prototype; // true
userA.__proto__ === User.__proto__ // false
```