

# HMM...OUR TANK'S A LITTLE FULL

Let's build another method that removes any object from our aquarium



```
var aquarium = {  
  🐠 Nemo: { type: "fish", species: "clownfish", length: 3.7 },  
  🐡 Dory: { type: "fish", species: "blue tang", length: 6.2 },  
  🌟 Peach: { type: "echinoderm", species: "starfish", length: 5.3 },  
  🏰 "Coral Castle": { type: "environment", material: "coquina", moves: false },  
  🐉 "Dragon Statue": { type: "environment", material: "plastic", moves: false },  
  addCritter: function ( name, type, species, length ){  
    this[name] = {type: type, species: species, length: length};  
  },  
  🐟 Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }  
};
```

```
aquarium.takeOut = function ( name ) {  
  this[name].name = name;  
  var temp = this[name];  
  delete this[name];  
  return temp;  
};
```

```
var toy = aquarium.takeOut("Dragon Statue");
```

*Our removal method works for toy Objects too!*

# HMM...OUR TANK'S A LITTLE FULL

Let's build another method that removes any object from our aquarium



```
var aquarium = {  
  🐠 Nemo: { type: "fish", species: "clownfish", length: 3.7 },  
  🐠 Dory: { type: "fish", species: "blue tang", length: 6.2 },  
  🌟 Peach: { type: "echinoderm", species: "starfish", length: 5.3 },  
  🏰 "Coral Castle": { type: "environment", material: "coquina", moves: false },  
  🐉  
  addCritter: function ( name, type, species, length ){  
    this[name] = {type: type, species: species, length: length};  
  },  
  🐠 Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }  
};
```

```
aquarium.takeOut = function ( name ) {  
  this[name].name = name;  
  var temp = this[name];  
  delete this[name];  
  return temp;  
};
```

```
var toy = aquarium.takeOut("Dragon Statue");
```

Our removal method works for toy Objects too!

# HMM...OUR TANK'S A LITTLE FULL

Let's build another method that removes any object from our aquarium



```
var aquarium = {  
  🐠 Nemo: { type: "fish", species: "clownfish", length: 3.7 },  
  🐡 Dory: { type: "fish", species: "blue tang", length: 6.2 },  
  🌟 Peach: { type: "echinoderm", species: "starfish", length: 5.3 },  
  🏰 "Coral Castle": { type: "environment", material: "coquina", moves: false },  
  
  addCritter: function ( name, type, species, length ){  
    this[name] = {type: type, species: species, length: length};  
  },  
  🐟 Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }  
};
```

```
aquarium.takeOut = function ( name ) {  
  this[name].name = name;  
  var temp = this[name];  
  delete this[name];  
  return temp;  
};
```



```
var toy = aquarium.takeOut("Dragon Statue");
```

Our removal method works for toy Objects too!

# HMM...OUR TANK'S A LITTLE FULL

Let's build another method that removes any object from our aquarium



```
var aquarium = {  
  🐠 Nemo: { type: "fish", species: "clownfish", length: 3.7 },  
  🐡 Dory: { type: "fish", species: "blue tang", length: 6.2 },  
  🌟 Peach: { type: "echinoderm", species: "starfish", length: 5.3 },  
  🏰 "Coral Castle": { type: "environment", material: "coquina", moves: false },  
  addCritter: function ( name, type, species, length ){  
    this[name] = {type: type, species: species, length: length};  
  },  
  🐟 Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }  
};
```



```
aquarium.takeOut = function ( name ) {  
  this[name].name = name;  
  var temp = this[name];  
  delete this[name];  
  return temp;  
};
```

```
var toy = aquarium.takeOut("Dragon Statue");  
console.log( toy );
```

➡ Object {type: "environment", material: "coquina", moves: false, name: "Dragon Statue"}



*See the Shimmering*

**OCEAN OF OBJECTS**

# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = { 🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
Nemo: { type: "fish", species: "clownfish", length: 3.7 }
```

# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = { 🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
Nemo: { type: "fish", species: "clownfish", length: 3.7 }  
Dory: { type: "fish", species: "blue tang", length: 6.2 }
```



# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



Nemo: { type: "fish", species: "clownfish", length: 3.7 }

Dory: { type: "fish", species: "blue tang", length: 6.2 }

Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }



# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



Nemo: { type: "fish", species: "clownfish", length: 3.7 }

Dory: { type: "fish", species: "blue tang", length: 6.2 }

Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }

Peach: { type: "echinoderm", species: "starfish", length: 5.3 }

# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



Nemo: { type: "fish", species: "clownfish", length: 3.7 }

Dory: { type: "fish", species: "blue tang", length: 6.2 }

Bubbles: { type: "fish", species: "yellow tang", length: 5.6 }

Peach: { type: "echinoderm", species: "starfish", length: 5.3 }

"Coral Castle": { type: "environment", material: "coquina", moves: false }

# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {, , , , , addCriticter, takeOut};
```



```
addCriticter: function ( name, type, species, length ){  
  this[name] = {type: type, species: species, length: length};  
}
```

# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
addCritter: function ( name, type, species, length ){
  this[name] = {type: type, species: species, length: length};
}
takeOut: function ( name ){
  this[name].name = name;
  var temp = this[name];
  delete this[name];
  return temp;
}
```

# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {, , , , , addCritter, takeOut};
```



= 3 fish total



# COUNTING FISH IN OUR TANK

What if we wanted to know how many fish our tank has at any given time?

```
var aquarium = {, , , , , addCriticter, takeOut};
```



= 3 fish total



```
aquarium.length;
```

→ undefined

Hmm, uh oh. Generic Objects don't have a native `length` like Arrays and Strings do, so we can't use that in a loop format in order to get to each property.

# ENUMERATION WITH THE FOR-IN LOOP

The for-in loop allows us to access each enumerable property in turn.

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
for ( var key in aquarium ) {  
    }  
}
```

The `in` keyword looks "in" the Object to its right and finds each enumerable property in turn. Think of it like accessing each index of an Array.



# ENUMERATION WITH THE FOR-IN LOOP

The for-in loop allows us to access each enumerable property in turn.

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
for ( var key in aquarium ) {  
    console.log(key);  
}
```

Logging out each  
property produces only  
their names as strings.

- ➔ Nemo
- ➔ Dory
- ➔ Bubbles
- ➔ Peach
- ➔ Coral Castle
- ➔ addCritter
- ➔ takeOut

key in aquarium

# ENUMERATION WITH THE FOR-IN LOOP

The for-in loop allows us to access each enumerable property in turn.

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
for ( var key in aquarium ) {  
    console.log(key);  
}
```

Logging out each  
property produces only  
their names as strings.

- Nemo
- Dory
- Bubbles
- Peach
- Coral Castle
- addCritter
- takeOut

key in aquarium

# ENUMERATION WITH THE FOR-IN LOOP

Now we need a way to determine which properties in 'aquarium' are fish!

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
for ( var key in aquarium ) {  
  
}
```

# ENUMERATION WITH THE FOR-IN LOOP

Now we need a way to determine which properties in 'aquarium' are fish!

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
var numFish = 0;
for ( var key in aquarium ) {
    if ( aquarium[key].type == "fish" ) {
    }
}
```

Since **key** contains the string name of a property, we can use it in a set of brackets as an expression.

# ENUMERATION WITH THE FOR-IN LOOP

Now we need a way to determine which properties in 'aquarium' are fish!

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
var numFish = 0;
for ( var key in aquarium ) {
    if ( aquarium[key].type == "fish" ) {
    }
}
```

Once we've accessed the Object that `key` refers to, we can seek its own `type` property, and check to see if the current Object is a `fish`.

# ENUMERATION WITH THE FOR-IN LOOP

Now we need a way to determine which properties in 'aquarium' are fish!

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
var numFish = 0;
for ( var key in aquarium ) {
    if ( aquarium[key].type == "fish" ) {
        numFish++;
    }
}
```

```
aquarium["addCritter"].type;
```

→ undefined

```
undefined == "fish";
```

→ false

?

# ENUMERATION WITH THE FOR-IN LOOP

Now we need a way to determine which properties in 'aquarium' are fish!

```
var aquarium = {🐠,🐡,🐟,🌟,🏰,addCritter, takeOut};
```



```
var numFish = 0;
for ( var key in aquarium ) {
    if ( aquarium[key].type == "fish" ) {
        numFish++;
    }
}
```

```
console.log(numFish);
```

➡ 3

Current Property	aquarium[property]	has .type?	aquarium[property].type	type == fish?	numFish
Nemo	aquarium["Nemo"]	YES	"fish"	TRUE	1
Dory	aquarium["Dory"]	YES	"fish"	TRUE	2
Bubbles	aquarium["Bubbles"]	YES	"fish"	TRUE	3
Peach	aquarium["Peach"]	YES	"echinoderm"	FALSE	3
Coral	aquarium["Coral Castle"]	YES	"environment"	FALSE	3
addCritter	aquarium["addCritter"]	no	undefined	FALSE	3
takeOut	aquarium["takeOut"]	no	undefined	FALSE	3



# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
var numFish = 0;
for ( var key in aquarium ) {
    if ( aquarium[key].type == "fish" ) {
        numFish++;
    }
}
```

# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in aquarium ) {  
        if ( aquarium[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
}
```

# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in      ) {  
        if ( [key].type == "fish" ) {  
            numFish++;  
        }  
    }  
}
```

# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in this ) {  
        if ( this[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
}
```

Remember, since `countFish` will be "owned" by `aquarium`, it will use the `this` keyword to refer to it as an owner Object.

# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in this ) {  
        if ( this[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
    return numFish;  
}
```

# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCriticter, takeOut, countFish};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in this ) {  
        if ( this[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
    return numFish;  
}
```

```
aquarium.countFish();
```

→ 3

# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐡, 🐟, 🌟, 🏰, addCritter, takeOut, countFish};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in this ) {  
        if ( this[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
    return numFish;  
}
```

```
var poorDory = aquarium.takeOut("Dory");
```



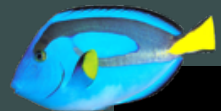
# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {, , , , addCriticter, takeOut, countFish};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in this ) {  
        if ( this[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
    return numFish;  
}
```



```
var poorDory = aquarium.takeOut("Dory");
```

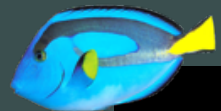
# ADDING OUR FISH COUNTER TO THE AQUARIUM

We'll need to build a function property using our loop

```
var aquarium = {🐠, 🐟, 🌟, 🏰, addCritter, takeOut, countFish};
```



```
aquarium.countFish = function ( ) {  
    var numFish = 0;  
    for ( var key in this ) {  
        if ( this[key].type == "fish" ) {  
            numFish++;  
        }  
    }  
    return numFish;  
}
```



```
var poorDory = aquarium.takeOut("Dory");
```

→ true

```
aquarium.countFish();
```

→ 2



*See the Shimmering*

**OCEAN OF OBJECTS**



*Welcome to*

THE PROTOTYPE PLAINS



LEVEL 5

THE PROTOTYPE PLAINS

# SURPRISE!

The Objects we've built so far have secret properties that we never saw!



valueOf

constructor

toLocaleString

toString

isPrototypeOf

propertyIsEnumerable

hasOwnProperty



# WHERE DID ALL OF THESE PROPERTIES COME FROM?

All of these Objects have a mysterious “parent” object that gives them properties



valueOf

constructor

toLocaleString

toString

isPrototypeOf

propertyIsEnumerable

hasOwnProperty





# WHERE DID ALL OF THESE PROPERTIES COME FROM?

All of these Objects have a mysterious “parent” object that gives them properties

valueOf

constructor

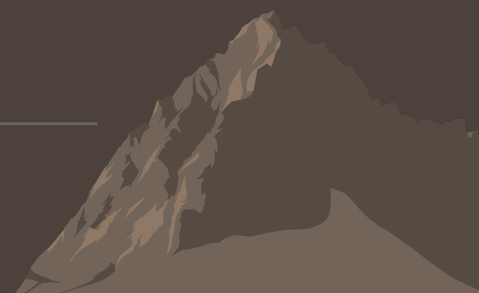
toLocaleString

toString

isPrototypeOf

propertyIsEnumerable

hasOwnProperty



# THE OBJECT'S PARENT IS CALLED ITS “PROTOTYPE”

All of those mysterious properties belong to and come from the Object's prototype

constructor

valueOf

toLocaleString

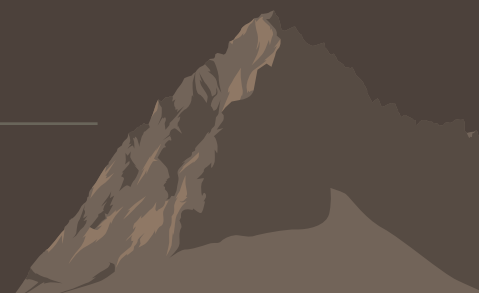
toString

OBJECT  
PROTOTYPE

isPrototypeOf

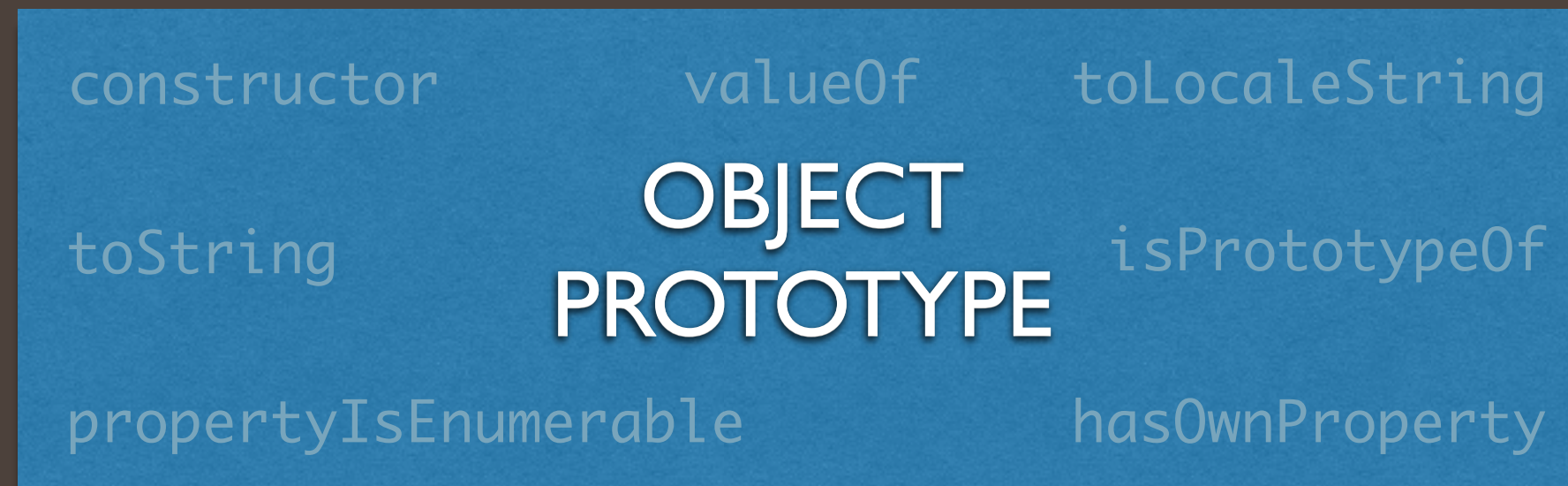
propertyIsEnumerable

hasOwnProperty



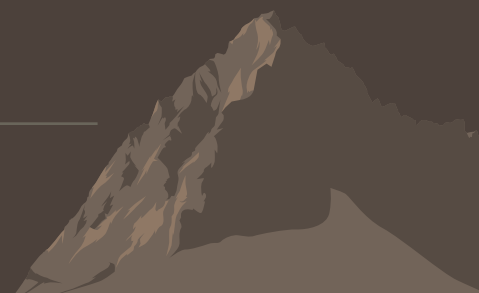
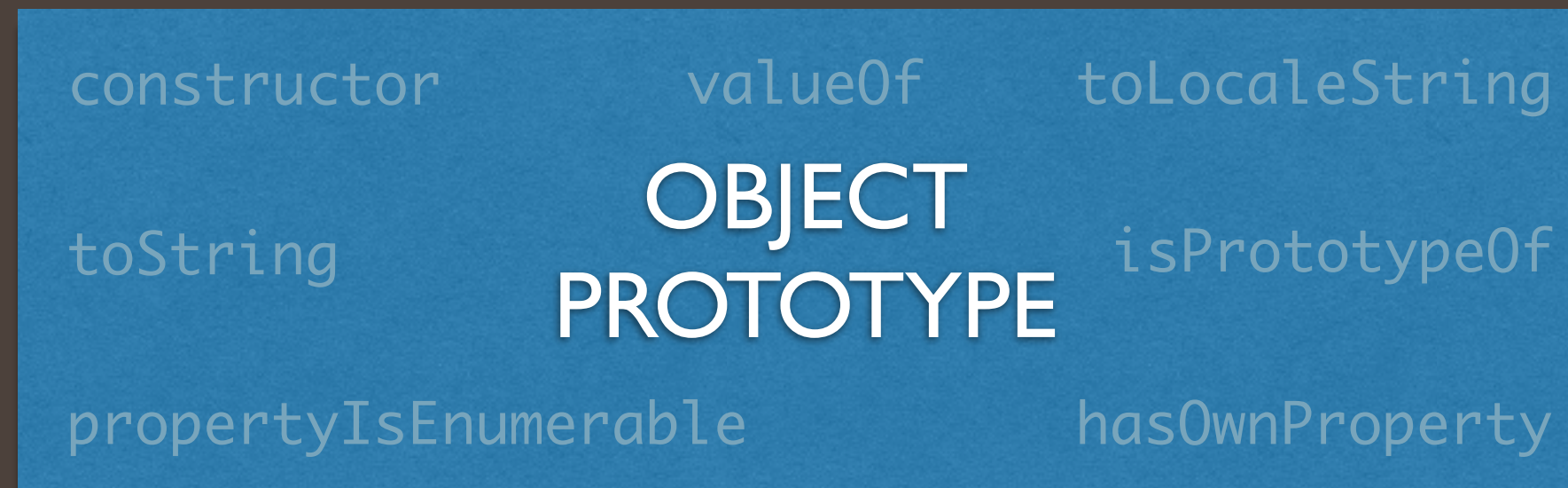
# THE OBJECT'S PARENT IS CALLED ITS “PROTOTYPE”

All of those mysterious properties belong to and come from the Object's prototype



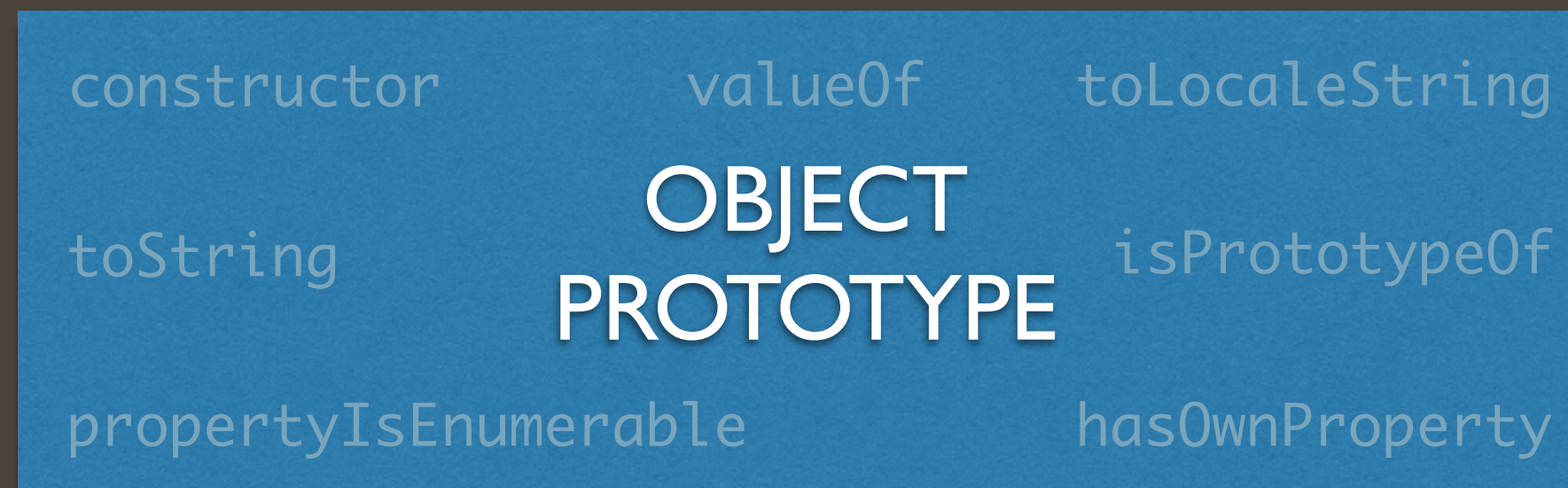
# THE OBJECT'S PARENT IS CALLED ITS “PROTOTYPE”

When a generic Object is created, its prototype passes it many important properties



# THE OBJECT'S PARENT IS CALLED ITS “PROTOTYPE”

When a generic Object is created, its prototype passes it many important properties



```
var aquarium = { 🐠, 🐟, 🏰, 🐉, addCriticter, takeOut, countFish };
```