مقرر الجافا سكريبت الباب الخامس (3) من كتاب
THE PRINCIPLES OF OBJECT ORIENTED JAVASCRIPT

فهم الكائنات

understanding objects

اعداد الطالبة : غدي فوزي التمتام

تحت اشراف : أ. مصطفى قاباج

javascript

2021

المحاور

المقدمة

- الكائنات Object :
في لغة جافا سكربت يمكن تغيير الكائنات أو التعديل عليها في أي وقت من تنفيذ الكود , وليس لديها قيود.
- جزء كبير من لغة جافا سكربت يعتمد على فهم الـ objects , أي فهم كيفية عمل هذه الكائنات هو مفتاح لفهم هذه اللغة.

تعريف الخصائص - Properties Defining

هناك طريقتين لنإشاء الكائنات: باستخدام object constructor منشئ الكائنات , أو باستخدام literal object أي بشكل حرفي .

على السبيل المثـال :

```
var person1 = {
 name: "Nicholas"
};
var person2 = new Object();
person2.name = "Nicholas";
person1.age = "Redacted";
person2.age = "Redacted";
person1.name = "Greg";
person2.name = "Michael";
```

ماذا حدث في المثال السّــابق !

- الكائنين person1 و person2 لديهم خاصية الأسم name .
- تم تعيين خاصية العمر للكائنين ، كما نرى في هذا المثال تستطيع تعيين الخاصية فورا أو حتى لاحقا .
- الكائنات يمكن التعديل في خصائصها في أي وقت.
- 1 هنا تم تغيير قيمة خاصية الأسم لكل الكائنين .
- عندما يتم إضافة خاصية للكائن، جافاسكربت تستخدم طريقة داخلي Method تسمى (put) ، تنشئ ال put بقعة spot تمكننا من حفظ الخاصية بها , لهذا مثلا في هذا المثال عندما تم تعريف خاصيتي الأسم والعمر في كل كائن تم استدعاء الطريقة put .
- عندما تتم إضافة قيمة جديدة لخاصية موجودة , تستخدم عملية تسمى (set) , هذه العملية تستبدل القيمة القديمة للخاصية بأخرى جديدة

حماية الخصائص - Detecting Properties

ولأن الخصائص يمكن اضافتها في أي وقت للكائن، لهذا يكون من الضروري أحيانا ان نتاكد من وجود خاصية معينة في كائن ما، ويحدث ان يستخدم المطورين المبتدئين طرقاً غير صحيحة مثل :

```
// unreliable
if (person1.age) {
 // do something with age
}
```

- مشكلة هذه الطريقة انها قد تخرج لنا نتائج غير صحيحة, فعندما تكون قيمة الخاصية الكائن (string , غير فارغ , رقم غير صفري , أو true) يكون لدينا الناتج true !

- وعلى العكس عندما تكون قيمة الخاصية عبارة عن(NaN, false, 0, undefined, Null, أو string فارغ) تكون النتيجة false !

- وكمثال ؛ اذا كانت لدينا قيمة person1.age عبارة عن 0 اذا الناتج سيكون false رغم ان الخاصية موجودة!

ولكن هناك طريقة أخرى أكثر واقعية وهي طريقة (in) ,

على سبيل المثال!

```
console.log("name" in person1); // true
console.log("age" in person1); // true
console.log("title" in person1); // false
```

- هذه الطريقة تبحث عن اسم خاصية معينة في كائن معين، في حالة وجدتها يكون الناتج true.
ملاحظة : الـ methods هم أيضاً خصائص لعمل وظائف معينة.

```
var person1 = {
 name: "Nicholas",
 sayName: function() {
 console.log(this.name);
 }
};
console.log("sayName" in person1); // true
```

مثـال آخر !

طريقة In تعمل حتى لايجاد الخصائص الموجودة اساساً في كل كائن والتي تسمى prototype properties ولهذا عند البحث عن خاصية باسم غير خاص، أي الاسم يمثل خاصية متواجدة أساسا في كل كائن وليست خاصة بكائن معين، ستكون النتيجة true ولحل هذه المشكلة تستخدم طريقة hasOwnProperty() .

والمثال التالي يوضح الفرق بين الطريقتين :

```
var person1 = {
 name: "Nicholas" ,
 sayName: function() {
 console.log(this.name);
 }
};
console.log("name" in person1); // true
console.log(person1.hasOwnProperty("name")); // true
console.log("toString" in person1); // true
console.log(person1.hasOwnProperty("toString")); // false
```

في هذا المثال الخاصية name هي خاصية خاصة بالكائن person1 ولهذا ستكون النتيجة true في كلا الحالتين.

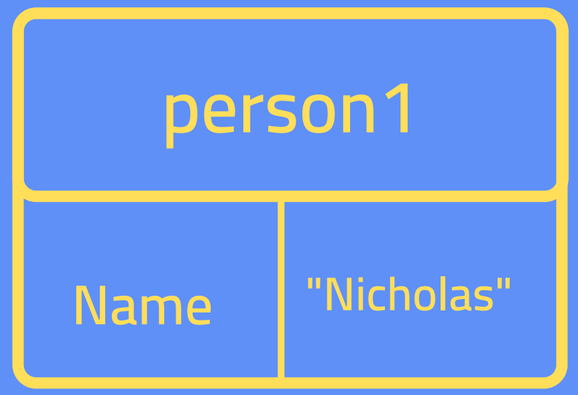
ولكن الخاصية tostring هي خاصية ليست خاصة, لهذا عند البحث عنها باستخدام In ستكون النتيجة true لانها موجودة أساسا في كل الكائنات ولكن عند البحث عنها باستخدام hasOwnProperty تكون النتيجة false لأنها لا تمثل خاصية خاصة بهذا الكائن تحديداً أي لم يتم انشائها لهذا الكائن.

إزالة الخصائص - removing properties

- يمكن إضافة الخصائص للكائنات في أي وقت.
- يمكن أيضا ازالتهم ببساطة.

ملاحظة : وضع قيمة الخاصية كـ Null لايحذف الخاصية, ولكن يجب استخدام المعامل delete, والذي يستدعي العملية المسماة delete .

```
var person1 = {
 name: "Nicholas"
};
console.log("name" in person1); // true
delete person1.name; // true - not output
console.log("name" in person1); // false
console.log(person1.name); // undefined
```



delete person1.name;

التعداد - enumeration

- بشكل افتراضي ، كل الخصائص التي تتم اضافتها الى الكائنات هي enumerable , بمعنى انه يمكنك المرور عليهم باستخدام foor-in loop .
- الخصائص القابلة للتعداد لديهم خصائص داخلية (قابلة للتعداد) تكون موضوعة على true

```
var property;
for (property in object) {
 console.log("Name: " + property);
 console.log("Value: " + object[property]);
}
```

أنواع الخصائص types of properties

هناك نوعان من الخصائص , خصائص البيانات data
وخصائص الملحقات او الاسترجاع properties
accessor properties

خصائص البيانات تحتوي على قيمة مثل خاصية الاسم , والسلوك الافتراضي لـ put method هو انشاء خاصية بيانات , ولكن خصائص الاسترجاع لا تحتوي على قيمة ولاكن بدلا من ذلك تعرف وظيفة function للاستدعاء عندما يتم قراءة الخاصية وتسمى (a gatter) ووظيفة للاستدعاء عندما تتم كتابة الخاصية تسمى (setter) , خصائص الاسترجاع فقط تتطلب اما getter او setter او الاثنين.

- هناك طريقة معينة لتعريف خصائص الاسترجاع باستخدام object literal :

```
var person1 = {
 _name: "Nicholas",
get name() {
 console.log("Reading name");
 return this._name;
 },
38 Chapter 3
set name(value) {
 console.log("Setting name to %s", value);
 this._name = value;
 }
};
console.log(person1.name);
// "Reading name" then "Nicholas"
person1.name = "Greg";
console.log(person1.name);
// "Setting name to Greg" then "Greg"
```

- هذا المثال السابق يعرف خاصية استرجاع تسمى name, وهناك خاصية بيانات تسمى name والتي تحتوي على القيمة الفعلية للخاصية.
- هذه العلامة (_) هي علامة متفق عليها تشير الى ان الخاصية المتضمنة لها ستكون private ولكن في هذه الحالة هي لا تزال public

السمات المميزة - property attributes

الصفات الشائعة : common attributes

هناك اثنان من سمات المميزة التي تتم مشاركتها بين خصائص البيانات وخصائص الاسترجاع.

- الأول هو (enumerable)، والذي يحدد إمكانية ان يتم تكرار أو المرور على الصفات.
- والآخر هو (configurable) أو قابلة للتكوين، والذي يحدد ان يتم تغيير الميزة او الصفة
- يمكنك إزالة خاصية (قابلة للتكوين) باستخدام delete ويمكنك التغيير على صفاتها في أي وقت.
- كل الخصائص التي تم تكوينها في كائن ما تكون قابلة للتعداد وقابلة للتكوين بشكل افتراضي.
- لو أردت تغيير صفات خاصية ما, يمكنك استخدام طريقة Object.defineProperty()

```
var person1 = {
name: "Nicholas"
};
Object.defineProperty(person1, "name", {
enumerable: false
});
console.log("name" in person1); // true
console.log(person1.propertyIsEnumerable("name"));
 // false
var properties = Object.keys(person1);
console.log(properties.length); // 0
Object.defineProperty(person1, "name", {
configurable: false
});
// try to delete the Property
delete person1.name;
console.log("name" in person1); // true
console.log(person1.name); // "Nicholas"
Object.defineProperty(person1, "name", {
 // error!!!
 configurable: true
});
```

- هذه الطريقة تقبل استخدام ثلاثة arguments :

1. الكائن الذي يملك الخاصية,
2. أسم الخاصية,
3. الكائن الواصف descriptor والذي يحتوي على السمات التي يجب تعيينها.

- الواصف لديه خصائص بنفس اسم الصفات الداخلية ولكن بدون استخدام الأقواس المربعة.

- على سبيل المثال افترض أنك تريد ان تكون سمة الكائن غير قابلة للتعداد ولا قابلة للتكوين.

- في هذا المثال خاصية الاسم تم تعيينها بشكل اعتيادي، ولكن لاحقاً تم تعديل خاصية التعداد الى false , ولهذا في النقطة 3 كما يوضح المثال، الطريقة propertyIsEnumerable() تكون نتيجتها أو ترجع false.

- بعد ذلك الاسم تم تغييره الى nonconfigurable او غير قابل للتكوين.ولهذا فشلت عملية الحذف لان الخاصية لا يمكن التعديل عليها, نفس الشيئ يحدث في النقطة 5.

- في النقطة 6 يظهر لنا error لانه كما ذكرنا من قبل لا يمكن تعيين الخاصية configurable بعد تحويلها الى nonconfigurable.

سمات خصائص البيانات - data property attributes

- خصائص البيانات تمتلك صفتان اخرتان، لا تمتلكهما خصائص الاسترجاع ، الأولى هي [[value]] أو القيمة ، والتي تحتوي قيمة الخاصية.

- هذه الصفات تُملأ تلقائياً عندما يتم انشاء خاصية في كائن، وكل كل قيم الخاصية تخزن في [[value]] ، حتى وان كانت القيمة هي وظيفة function .

- الصفة الثانية هي [[writeable]] ، وهي قيمة منطقية تشير الى الخاصية التي يمكن الكتابة عليها .

ملاحظة: كل الخصائص هم writable بشكل افتراضي مالم يعيّن عكس ذلك.

- اذاً الآن يمكنك تعريف خاصية بيانات باستخدام Object.defineProperty() حتى وان كانت الخاصية لم توجد بعد.

على سبيل المثال:

```
var person1 = {
name: "Nicholas"
};
```

سبق وتحدثنا عن إمكانية تعيين خاصية name في كائن معين بهذه الطريقة, المثال التالي يضمن نفس النتيجة.

- في هذا المثال اولاً Object.defineProperty() يتأكد من وجود الخصائص في الكائن، في حالة عدم وجودهم ينشئ الخاصية, في هذا المثال خاصية name لم تكن موجودة.

- عندما تعرّف خاصية جديدة باستخدام Object.defineProperty() , من المهم تحديد كل الصفات, لأن طبيعة الصفات المنطقية تكون على الحالة false بشكل افتراضي.

```
var person1 = {};
Object.defineProperty(person1, "name", {
 value: "Nicholas",
 enumerable: true,
 configurable: true,
 writable: true
});
```

في المثال التالي، يتم انشاء خاصية name والتي تكون nonenumerable و nonconfigurable و nonwritable لأنه لم يتم وضع قيمة true لأي من هذه الصفات من قبل.

```
var person1 = {};
Object.defineProperty(person1, "name", {
 value: "Nicholas"
});
console.log("name" in person1); // true
console.log(person1.propertyIsEnumerable("name")); // false
delete person1.name;
console.log("name" in person1); // true
person1.name = "Greg";
console.log(person1.name); // "Nicholas"
```

لهذا لن تستطيع عمل أي عملية على الخاصية عدا قراءة قيمتها.

accessor property attributes

- خصائص الاسترجاع أيضاً لديها صفتان اضافيتان.

ولأنه لا يوجد أي قيمة مخزنة لخصائص الاسترجاع، ليس هناك أي حاجة لـ [[value]] أو [[writable]]. بدلاً من ذلك خصائص لاسترجاع لديها [[Get]] و [[Set]]، واللتان تضمان وظائف مثل setter و getter

- ملاحظة : كل ما عليك هو تعريف واحدة فقط من هذه الصفات لإنشاء الخاصية، وان حاولت انشاء خاصية مع كلا خصائص البيانات والاسترجاع سيكون الناتج error.

- ميزة استخدام صفات خصاص الاسترجاع بدلاً من كائن التدوين الحرفي (object literal notation) لتعريف خصائص الاسترجاع، هي انه يمكنك أيضاً تعريف تلك الخصائص مع كائنات موجودة من قبل.

- عندما تريد أن تستخدم كائن التدوين الحرفي، يجب عليك أن تعرّف خصائص الاسترجاع عند تكوين الكائن.

- مثل خصائص البيانات، تستطيع أيضاً أن تحدد أي خصائص الاسترجاع ستكون configurable أو enumerable.

على سبيل المثال

```
var person1 = {
 _name: "Nicholas"
};
Object.defineProperty(person1, "name", {
 get: function() {
 console.log("Reading name");
 return this._name;
 },
 set: function(value) {
 console.log("Setting name to %s", value);
 this._name = value;
 },
 enumerable: true,
 configurable: true
});
```

```
var person1 = {
 _name: "Nicholas",
 get name() {
 console.log("Reading name");
 return this._name;
 },
 set name(value) {
 console.log("Setting name to %s", value);
 this._name = value;
 }
};
```

هذا الكود يمكن أيضاً كتابته بهذه الطريقة:

```
var person1 = {
 _name: "Nicholas"
};
Object.defineProperty(person1, "name", {
 get: function() {
 console.log("Reading name");
 return this._name;
 }
});
console.log("name" in person1); // true
console.log(person1.propertyIsEnumerable("name"));
// false
delete person1.name;
console.log("name" in person1); // true
person1.name = "Greg";
console.log(person1.name); // "Nicholas"
```

- لاحظ أن المفاتيح get و set في الكائن يتم تمريرها خلال Object.defineProperty() هي خصائص بيانات تحتوي على وظيفة function ولا يمكنك استخدام صيغة الكائن الحرفي هنا.
- وضع أو تحديد الصفات الأخرى ([[enumerable]]&[[configurable]]) تسمح لك بتغيير كيفية عمل خاصية الاسترجاع ,

على سبيل المثال :

يمكنك انشاء خاصية غير قابلة للتكوين وغير قابلة للتعداد ولا الكتابة بهذه الطريقة:

في هذا المثال : الخاصية name هي خاصية استرجاع مع فقط getter , أي بدون setter أو أي خصائص أخرى، لهذا القيمة يمكن فقط قراءتها ولكن بدون تغييرها.

defining multiple properties - تعريف عدة خصائص

- يمكن تعريف عدة خصائص في الكائن، ولكن باستخدام Object.defineProperties بدلا من Object.defineProperty

- هذه الطريقة تقبل استخدام اثنان من الـ argument , الكائن الذي سنعمل عليه والكائن الذي يحتوي كل معلومات الخاصية.

- مفاتيح الـ argumument الثاني هم أسماء الخصائص والقيم هم واصف الكائنات التي تعرف الصفات لتلك الخصائص .

المثال التالي يعرف خاصيتين :

```
var person1 = {};
Object.defineProperties(person1, {
1 // data property to store data
 _name: {
 value: "Nicholas",
 enumerable: true,
 configurable: true,
 writable: true
2 }, // accessor property
 name: {
 get: function() {
 console.log("Reading name");
 return this._name;
 },
 set: function(value) {
 console.log("Setting name to %s", value);
 this._name = value;
 },
 enumerable: true,
 configurable: true
 } });
```

هذا المثال يعرف _name كخاصية بيانات لاحتواء معلومات (1) nameg كخاصية استرجاع (2).

-يمكنك تعريف أي عدد من الخصائص بنفس الطريقة, ويمكنك أيضاً تغيير خصائص موجودة وانشاء خصائص جديدة في نفس الوقت, بنفس تأثير استدعاء Object.defineProperty أكثر من مرة.

استرداد صفات خاصية - retrieving property attributes

- لو احتجت لجلب صفات خاصية ما, يمكنك فعلها باستخدام Object.getOwnPropertyDescriptor().
- هذه الطريقة تعمل فقط مع الخصائص الخاصة own properties.
- هذه الطريقة تقبل اثنان من ال arguments الكائن الذي سنعمل عليه واسم الخاصية التي سيتم استردادها.
- لو كانت الخاصية موجودة, يجب أن تتلقى واصف الكائن مع أربعة خصائص وهي: configurable, enumerable, والخاصيتان الأخيرتان مناسبتان لنوع الخاصية.

```
var person1 = {
 name: "Nicholas"
};
var descriptor = Object.getOwnPropertyDescriptor(person1,
"name");
console.log(descriptor.enumerable); // true
console.log(descriptor.configurable); // true
console.log(descriptor.writable); // true
console.log(descriptor.value); // "Nicholas"
```

- المثال التالي يوضح كيفية انشاء خاصية وتفقد صفاتها :

عند استدعاء Object.getOwnPropertyDescriptor() يتم ارجاع كائن مع enumerable, configurable, writable, و value, مع أن لم يتم تعريفهم خلال Object.defineProperty()

منع تعديل الكائن - preventing object modification

- الكائنات مثل الخصائص , لديها صفات داخلية والتي تتحكم في سلوكها.
- واحدة من تلك الصفات هي [[Extensible]] أو موسعة,وهي قيمة منطقية تبين ما اذا كان الكائن يمكن التعديل عليه.
- كل الكائنات التي يتم انشائها هي موسعة بشكل افتراضي، بمعنى يمكن إضافة خصائص جديدة للكائن في أي وقت.
- بوضع الصفة [[Extensible]] على الوضع false, يمكنك منع إضافة خصائص جديدة للكائن.

هناك ثلاثة طرق لإنجاز هذا الأمر:

preventing extensions

طريقة واحدة لانشاء كائن يكون nonextensible
أو غير موسع، هي باستخدام
Object.preventExtensions() .
هذه الطريقة تقبل حجة واحدة أو argument،
والذي هو الكائن الذي تريد أن تجعله
Nonextensible.
عندما تستخدم هذه الطرقة على الكائن، لن
تتمكن من إضافة أي خصائص جديدة لاحقاً.

```
var person1 = {
 name: "Nicholas"
};
 console.log(Object.isExtensible(person1));
 // true
Object.preventExtensions(person1);
console.log(Object.isExtensible(person1));
 // false
person1.sayName = function() {
 console.log(this.name);
};
console.log("sayName" in person1); // false
```

بعد انشاء الكائن person1، هذا المثال يتحقق من صفة ال [[Extensible]] للكائن قبل جعله unchangeable
أو غير قابل للتغيير. والأن هذا الكائن هو غير موسع، لهذا sayName() لن يتم ا ضافتها ابداً.

ختم الكائن - sealing objects

- الطريقة الثانية لانشاء كائن غير موسع هو seal أو ختم للكائن.
- الكائن المختوم هو غير موسع، وكل خصائصه غير قابلة للتكوين أو nonconfigurable.
- هذا يعني أنه ليس فقط لا يمكنك إضافة خصائص جديدة للكائن، انما أيضاً لا يمكنك إزالة خصائص أو تغيير صفاتهم من خصائص بيانات لخصائص استرجاع أو العكس.
- لو كان الكائن مختوم، يمكنك فقط القراءة أو الكتابة على خصائصه.
- يمكنك استخدام طريقة Object.seal() , على الكائن لختمه.
- وعندما يحدث هذا صفة ال [[extensible]] تكون على الوضع false، وكل الخصائص لديهم صفات تكون [[configurable]]، موضوعة على الوضع false.
- يمكنك التحقق اذا ما كان الكائن مختوم أم لا باستخدام Object.isSealed() .

```
var person1 = {
 name: "Nicholas"
};
console.log(Object.isExtensible(person1)); // true
console.log(Object.isSealed(person1)); // false
Object.seal(person1);
v console.log(Object.isExtensible(person1)); // false
console.log(Object.isSealed(person1)); // true
person1.sayName = function() {
 console.log(this.name);
};
console.log("sayName" in person1); // false
person1.name = "Greg";
console.log(person1.name); // "Greg"
delete person1.name;
console.log("name" in person1); // true
console.log(person1.name); // "Greg"
var descriptor = Object.getOwnProperty
Descriptor(person1, "name");
console.log(descriptor.configurable); // false
```

كما يوضح المثال التالي

هذا المثال يختم على الكائن person1 لهذا لا يمكنك إضافة او إزالة خصائص.

-وعندما تكون كل الكائنات المختومة nonextinsible, لهذا Object.isExtensible() , ترجع النتيجة false.

تجميد الكائنات freezing Objects

- آخر طريقة لانشاء كائن غير موسع هو بتجميده.
- لو كان الكائن مجمد, لا يمكنك إضافة او إزالة خصائص، ولهذا لا يمكنك تغيير أنواع الخصائص، ولا يمكنك الكتابة لأي خصائص بيانات.
- الكائن المجمد هو كائن مختوم, عندما خصائص البيانات تكون أيضا فقط للقراءة.
- الكائنات المجمدة لا يمكن جعلها غير مجمدة, لهذا تظل في الحالة التي كانت عليها قبل أن تكون مجمدة.
- يمكنك تجميد الكائن باستخدام Object.freeze() , و تحديد اذا ما كان الكائن مجمد باستخدام Object.isFrozen().

```
var person1 = {
 name: "Nicholas"
};
console.log(Object.isExtensible(person1)); // true
console.log(Object.isSealed(person1)); // false
console.log(Object.isFrozen(person1)); // false
u Object.freeze(person1);
v console.log(Object.isExtensible(person1)); // false
w console.log(Object.isSealed(person1)); // true
console.log(Object.isFrozen(person1)); // true
person1.sayName = function() {
 console.log(this.name);
};
console.log("sayName" in person1); // false
x person1.name = "Greg";
console.log(person1.name); // "Nicholas"
delete person1.name;
console.log("name" in person1); // true
console.log(person1.name); // "Nicholas"
var descriptor =
Object.getOwnPropertyDescriptor(person1, "name");
console.log(descriptor.configurable); // false
console.log(descriptor.writable); // false
```

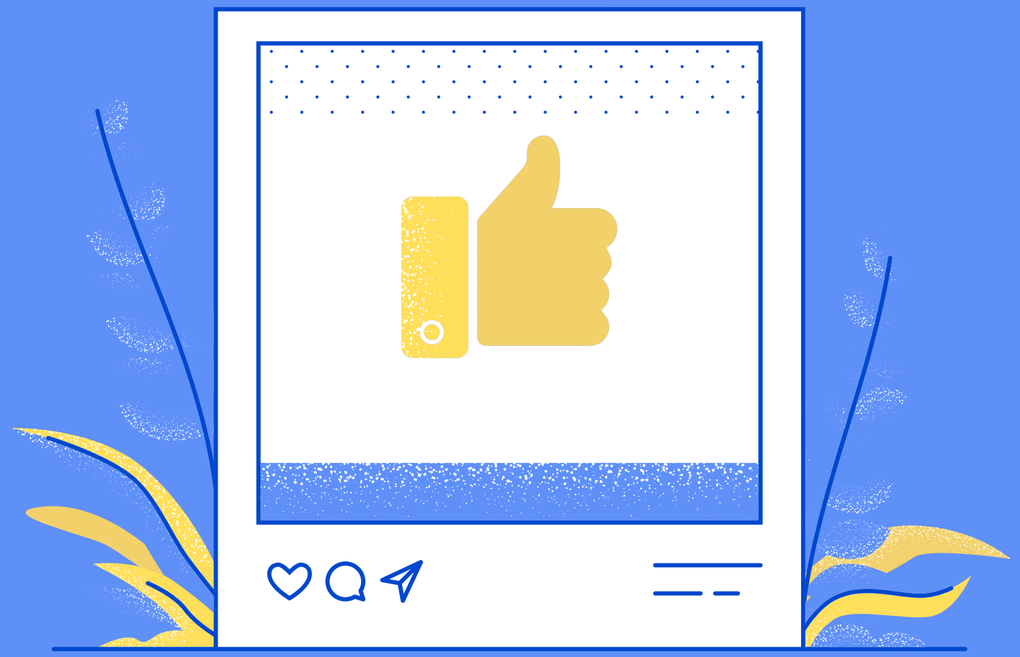
كما يوضح المثال التالي

في هذا المثال person1 هو مجمد.
الكائنات المجمدة هي أيضاً تعتبر غير موسعة ومختومة, لهذا
Object.isExtensible() , يرجع false.

والعكس عند Object.isSealed() .

خاصية الاسم لا يمكن تغييرها لهذا حتى عند اسناد اليها القيمة "greg",
العملية تفشل.

فهم الكائنات
understanding objects



مقرر الجافا سكريبت الباب الخامس (3) من كتاب
THE PRINCIPLES OF OBJECT ORIENTED JAVASCRIPT

اعداد الطالبة : غدي فوزي التمتام

تحت اشراف : أ. مصطفى قاباج

javascript

2021