Functions

الطالبة : بشرى عصام منكوسة
رقم القيد:172032

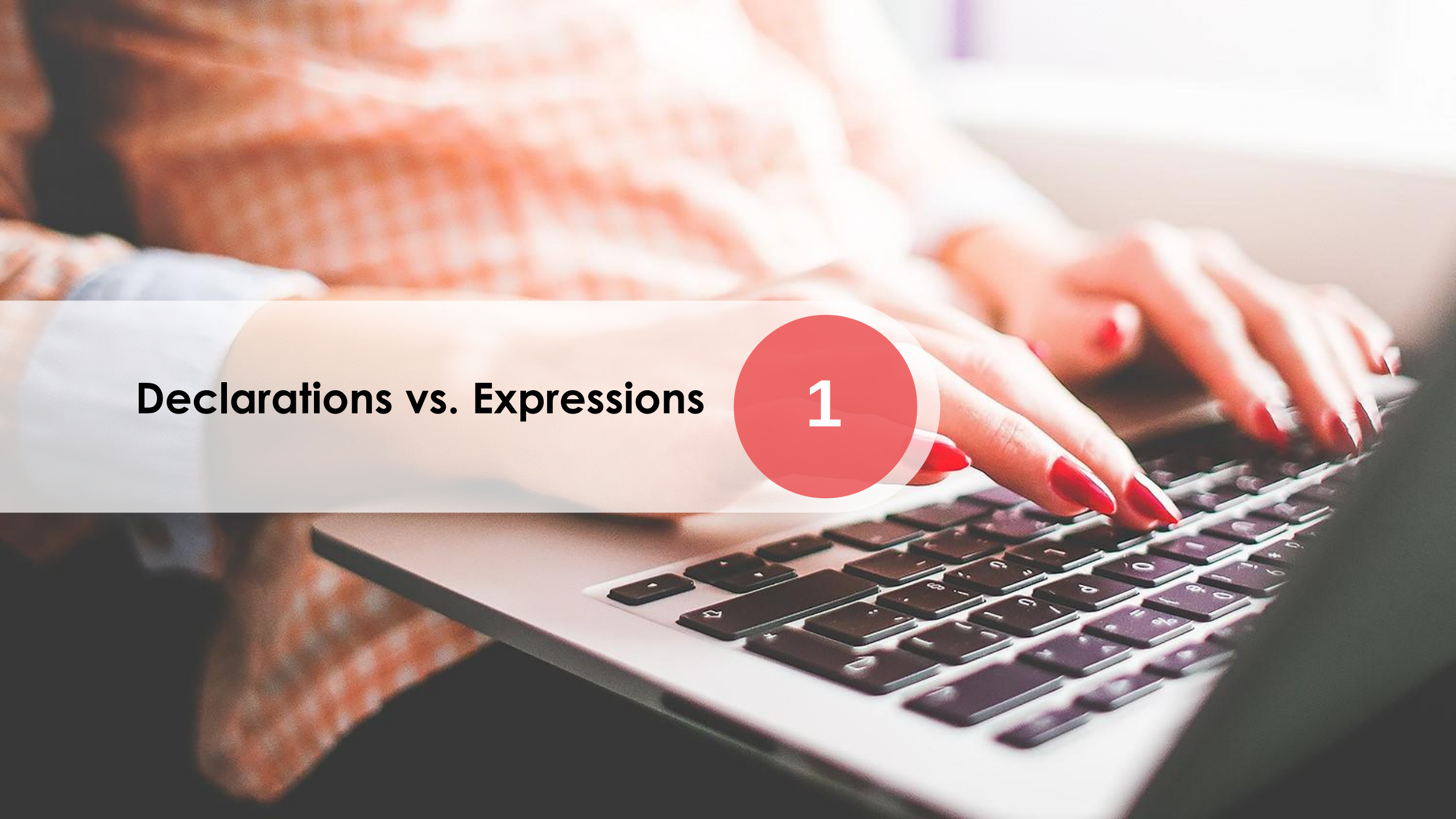
Contents

Functions

- يمكن تعريفها على انها وظيفة او دالة.
- وهي مجموعة من الأوامر (التعليمات) البرمجية الذي لن يتم تنفيذها الا بعد استدعاؤها ، وأيضاً مهمة لإنها تجعل الكود قابلاً لإعادة الاستخدام.
- وتختلف function في لغات البرمجة ولكن في JavaScript نعاملها معاملة object، وما يجعلها مميزة من أي object لأخر هو وجود خاصية داخلية [[Call]].
- لا يمكن الوصول الى الخصائص الداخلية عبر الكود ، بل بتحديد سلوك وخصائص الكود اثناء تنفيذه.
- خاصية [[Call]] فريدة للوظائف تشار الى انه يمكن تنفيذ object .

Declarations vs. Expressions

1



Declarations

اعلان الوظيفة، أي تعريف الوظيفة التي تبدا بالكلمة المفتاحية وتتضمن اسم ومعاملات محددة.

اسم الوظيفة

```
function add(num1, num2)
{

return num1 + num2;

}
```

(معاملات)
Parameters

(return) تحدد القيمة
التي يتم استرجاعها .
محتويات الوظيفة
(num1,num2)

Expressions

- التعبير الوظيفي ، الذي لا يتطلب اسماً بعد الوظيفة ،أي تعتبر الوظائف مجهولة وذلك لأن كائن الوظيفة ليس له اسم.
- يشار عادة الى تعبير الوظائف عبر متغير او خاصية.
- تتطابق تعبيرات مع إعلانات باستثناء الاسم والفاصلة المنقوطة ، تنتهي التعبيرات بفاصلة منقوطة في النهاية كما لو تقوم بتعيين قيمة أخرى.

```
var add =
function(num1, num2)
{
return num1 + num2;
};
```

Add يعين قيمة دالة للمتغير.

Declarations vs. Expressions 1

- صحيح انهم متشابهان الا انهم يختلفان بطريقة مهمة.
- يتم رفع إعلانات الوظائف الى اعلى (اما الوظيفة او النطاق العام) .
- يحدث رفع لإعلانات الوظائف فقط وذلك بسبب ان اسم الوظيفة مُعرف مسبقاً، ومن جهة أخرى لا يمكن رفع تعبيرات الوظائف لأنه لا يمكن الإشارة الى الوظائف الا من خلال متغير

عند النظر الى الكود كأنه يتسبب في حدوث خطأ ولكنه يعمل جيد، ذلك بسبب محرك Java Script يقوم برفع اعلان الوظيفة الى اعلى وينفذ

```
> function add(num1, num2) {
  return num1 + num2;
  }
  var result = add(5, 5);
```

عندما يتم تحديد الكود يمكننا تحديد وظيفة بعد استخدامها دون حدوث أي أخطاء.

```
> var result = add(5, 5);
  function add(num1, num2) {
  return num1 + num2;
  }
```

Functions as Values

2

Functions as Values

- Java Script تحتوي على وظائف مميزة ومن الدرجة الأولى، أي الوظائف ليست فقط بناء الجملة ولكن ايضاً قيم ، مما يعني ان يمكن تخصيصها للمتغيرات وتخزينها في خصائص الكائنات (objects) أو عناصر المصفوفات وتمريرها كوسيطات(arguments) للوظائف، ايضاً اعادتها من الوظائف.
- يمكن استخدام دالة في اي مكان تستخدم فيه أي قيمة مرجعية أخرى هذا ما يجعل وظائف (Functions) الجافا سكريبت قوية وفريدة ومميزة.

```
function sayHi() {
console.log("Hello!");
}
sayHi();
```

1

```
Hello!
```

```
var sayHi2 = sayHi;
sayHi2();
```

2

```
Hello!
```

في الكود 1 يوجد اعلان للدالة .

في الكود 2 قمت بإنشاء متغير اسمه SayHi2 واعطائه قيمة .

يشير ان كل من الكود 1 و 2 يؤديان نفس الوظيفة ، أي يمكن تنفيذ أي منهما بنفس النتيجة.

Functions as Values

When you keep in mind that functions are objects, a lot of the behavior starts to make sense.

عند الاخذ بعين الاعتبار ان الوظائف هي كائنات (objects) فإن الكثير من السلوك يبدأ في ان يكون له معنى.

المصفوفات(Array):

- تستخدم ()sort الموجودة في مصفوفات جافا سكريبت كمعامل اختياري وتسمى دالة المقارنة ، يتم استدعائها عندما نحتاج الى مقارنة قيمتين في المصفوفة.
- تحول كل عنصر في المصفوفة الى سلسلة ثم تقوم بعملية المقارنة، أي لا يمكننا فرز المصفوفة من الأرقام بدقة دون تحديدها.
- ملاحظات يجب معرفتها:
  - اذا كانت القيمة الأولى اكبر من القيمة الثانية فيجب ان تُرجع الدالة عدد موجب.
  - اذا كانت القيمة الأولى اصغر من القيمة الثانية فيجب ان تُرجع الدالة عدد سالب.
  - اذا كانت القيمتين متساويتين اذاً عندها ترجع الدالة صفر.

Example

دالة المقارنة المستخدمة في الكود1 هي تعبير وظيفي لا يوجد اسم للدالة، انه موجود فقط كمرجع ليتم تمريره الى دالة أخرى مما يجعلها وظيفة مجهولة ، طرح القيمتين يعود النتيجة من دالة المقارنة.

* تم ترتيبهم ترتيب تصاعدي.

في الكود 2 ،دالة الاستدعاء لا تستخدم وظيفة المقارنة يختلف ترتيب المصفوفة عن المعتاد، وذلك بعد العدد 1 يتبعه العدد 10 وهو الناتج من دمج الرقمين 1 و 0 ، والعدد 1 يأتي قبل 2 في ترتيب *يونيكود*

```
> var numbers = [ 1, 5, 8, 4, 7, 10, 2, 6 ];
   numbers.sort(function(first, second) {
  return first - second;
  });
  console.log(numbers);                    1
    ▶ (8) [1, 2, 4, 5, 6, 7, 8, 10]
  undefined
> numbers.sort();
  console.log(numbers);                    2
    ▶ (8) [1, 10, 2, 4, 5, 6, 7, 8]
```

Example

الكود1 :

تم ترتيب الكلمات بناءً على أولوية الأرقام قبل الاحرف الكبيرة في يونيكود(أي الاقام تأتي قبل الاحرف سواء كانت الكبيرة او الصغيرة).

الكود 2:

تم ترتيب الكلمات بناءً على أولوية الاحرف في الترتيب .

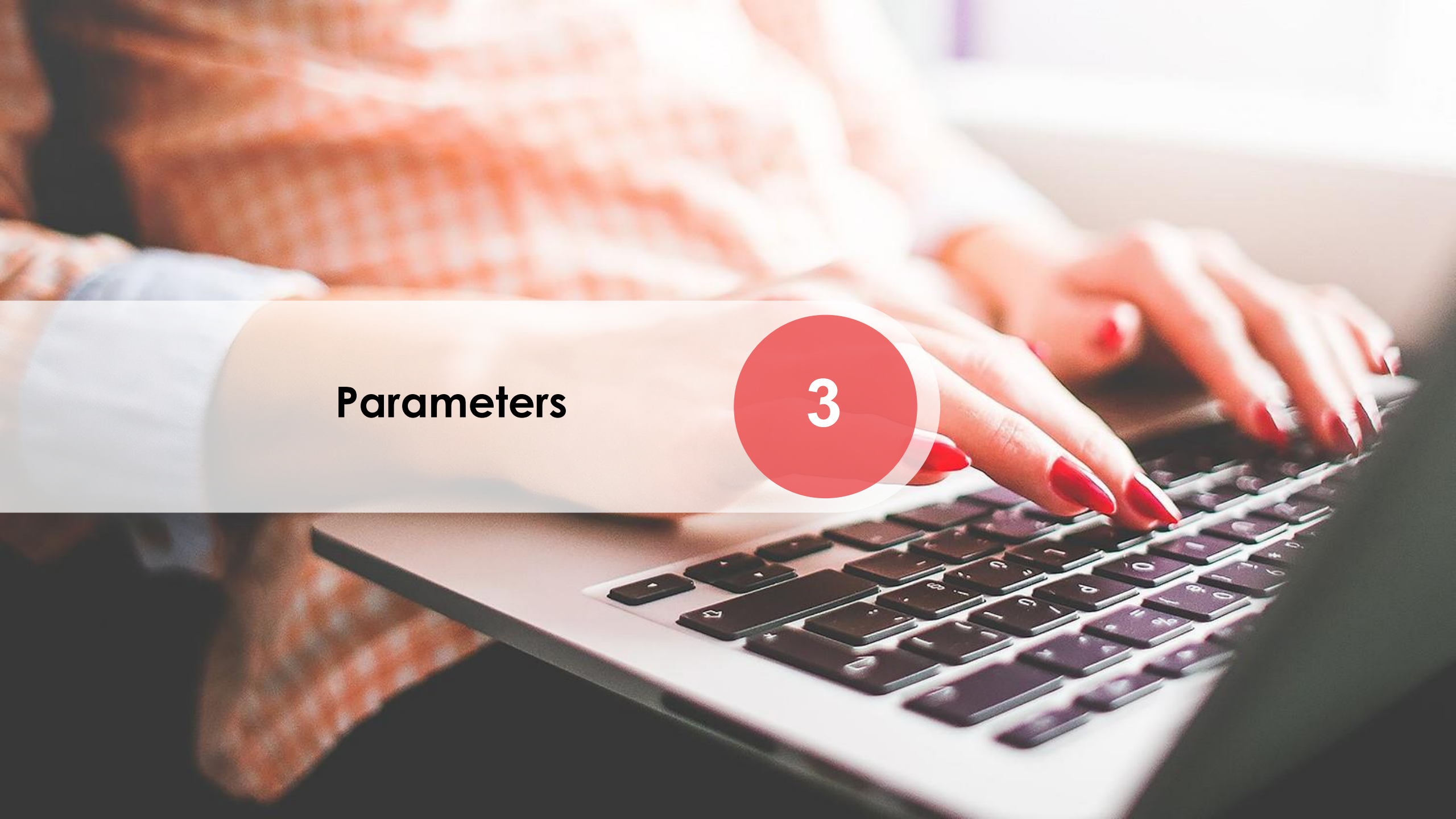
```
> var things = ['word', 'Word', '1 Word', '2 Words'];
  things.sort();
  console.log(things);

  ▶ (4) ["1 Word", "2 Words", "Word", "word"]

< undefined

> var fruit = ['cherries', 'apples', 'bananas'];
  fruit.sort();
  console.log(fruit);

  ▶ (3) ["apples", "bananas", "cherries"]
```



Parameters

3

Parameters

يوجد جانب فريد ومميز من وظائف جافا سكريبت انه يمكننا تمرير أي عدد من معاملات (معلمات) الى أي وظيفة دون التسبب في حدوث أي خطأ ، وذلك لان معاملات(معلمات) يتم تخزينها فعلياً كهيكل يشبه المصفوفة يسمى الوسائط (arguments) .

يمكن ان تحتوي الوسائط على عدد من القيم ،تتم الإشارة الى القيم عبر مؤشرات رقمية، وتوجد خاصية طول لتحديد عدد القيم الموجودة.

كائن الوسائط (The arguments object) متاح تلقائيًا داخل أي دالة. هذا يعني أن المعاملات(معلمات) المسماة في إحدى الوظائف موجودة في الغالب للتسهيل ولا تحد فعليًا من عدد الوسائط التي يمكن أن تقبلها الوظيفة.

The arguments object is not an instance of Array and therefore doesn't have the same methods as an array; Array.isArray(arguments) always returns false.

معاملات الوظيفة (Function parameters) هي الأسماء التي يتم تعريفها في تعريف الوظيفة ، وتُعرف القيم الحقيقية التي تم تمريرها إلى الوظيفة في تعريف الوظيفة باسم الوسائط.

وسيطات الوظيفة (Function arguments) هي القيم الحقيقية التي يتم تمريرها إلى الدالة.

Parameters

- نعرف أن الوظيفة هي في الواقع مجرد كائن ، لذلك يمكن أن يكون لها خصائص.
- تشير خاصية الطول إلى طبيعة الوظيفة ، أو عدد المعلمات التي تتوقعها.
- تعد معرفة طبيعة الوظيفة أمرًا مهمًا في جافا سكريبت لأن الوظائف لن تؤدي إلى خطأ إذا قمت بتمرير عدد كبير جدًا من المعلمات أو القليل جدًا منها.

```
> function reflect(value) {
  return value;
  }
  console.log(reflect("Welcome!"));
  console.log(reflect("Welcome!", 25));
  console.log(reflect.length);1
  reflect = function() {
  return arguments[0];
  };
  console.log(reflect("Welcome!"));
  console.log(reflect("Welcome!", 25));
  console.log(reflect.length);
  Welcome!
  Welcome!
  1
  Welcome!
  Welcome!
  0
```

في المثال قام بتعريف وظيفة reflect باستخدام معلمة واحدة ،وخاصية الطول=1 لان يوجد parameter ، وتنفيذ اسهل بكثير في الفهم.

تم إعادة تعريف الوظيفة بدون parameter فتقوم بإرجاع الوسيطات [0]، يخدم مثل ما خدم الكود الأول ولكن الاختلاف في ان طوله =0.

Parameters

في بعض الأوقات يكون استخدام الوسائط (arguments) اكثر فاعلية من تسمية المعاملات(naming parameters)، مثلاً أريد انشاء دالة تقبل أي عدد من المعلمات وترجع مجموعها ، لا يمكنني استخدام المعلمات المسماة لان لا اعرف العدد الذي سنحتاج اليه ، لذلك استخدام(arguments) هو الأفضل.

```
> function sum() {
  var result = 0,
  i = 0,
  len = arguments.length;
  while (i < len) {
  result += arguments[i];
  i++;
  }
  return result;
  }
  console.log(sum(9, 4));
  console.log(sum(3, 5, 6, 8))
  console.log(sum(40));
  console.log(sum());
  13
  22
  40
  0
```

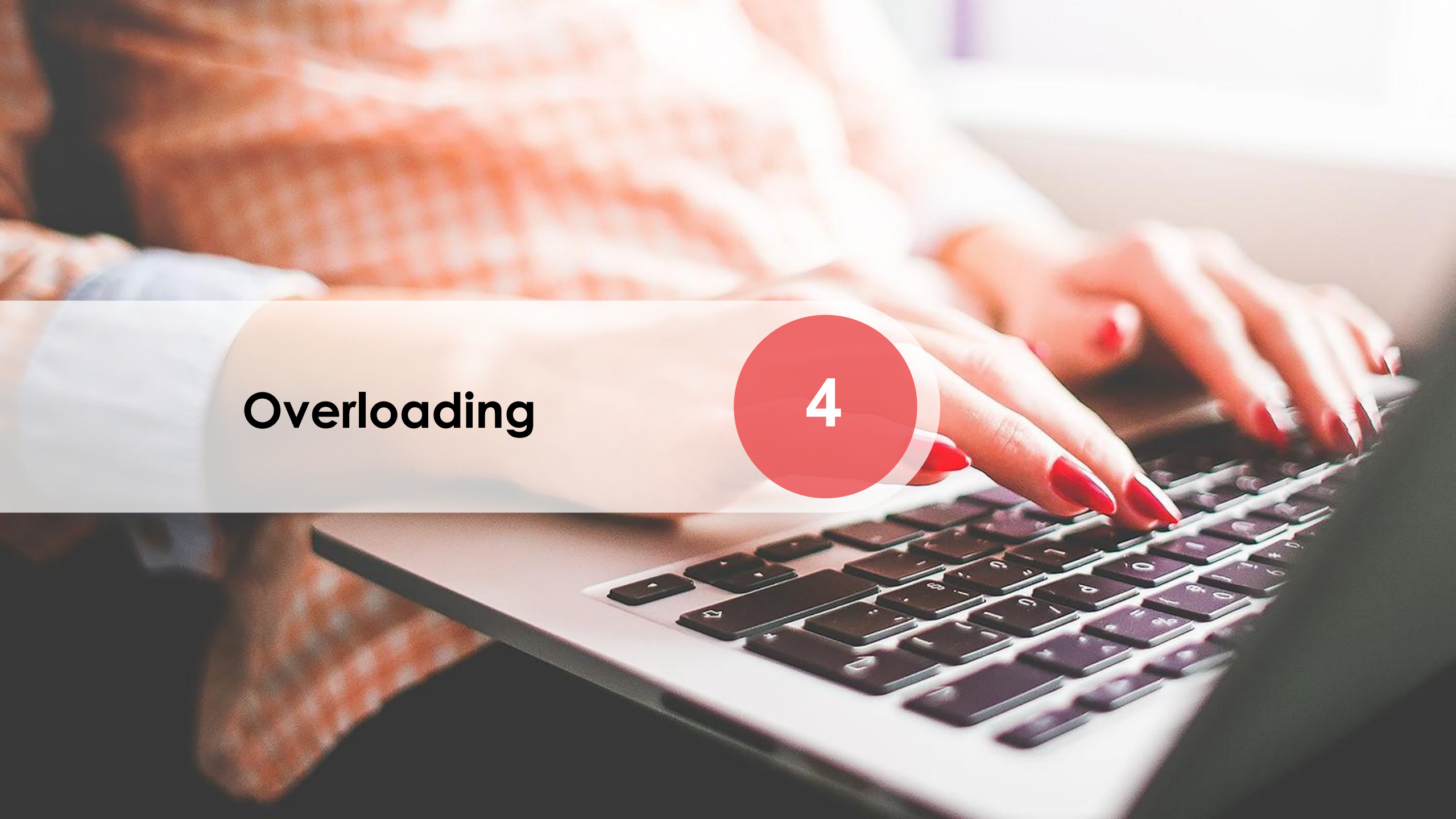
تقبل الدالة sum () أي عدد من المعلمات وتجمعها معًا عن طريق تكرار القيم الموجودة في الوسيطات باستخدام حلقة while.

تعمل الوظيفة حتى عندما لا يتم تمرير أي معلمات ، لأن النتيجة هي تمت تهيئته بقيمة 0.

المخرجات

Overloading

4



Overloading

التحميل الزائد هي قدرة الوظيفة الواحدة على حصول توقيعات متعددة، تدعم أغلب اللغات التحميل الزائد.

يتكون توقيع الوظيفة من اسمها وايضاً الى عدد ونوع المعاملات (المعلمات) التي تتوقعها الوظيفة، بمعنى آخر مثلا يكون لدالة واحدة توقيع واحد يقبل وسيطة سلسلة واحدة ، واخر يقبل وسيطتين رقيمتين، تحدد اللغة اصدار الدالة المطلوب استدعاؤها بناءً على الوسائط التي يتم تمريرها.

```
> function sayMessage(message) {
  console.log(message);
  }
  function sayMessage() {
  console.log("Default message");
  }
  sayMessage("Hello!");

  Default message
```

عند تحديد وظائف متعددة بنفس الاسم ،فإن الوظيفة المكتوبة الأخيرة في التعليمات هي التي يتم استخدامها (أي المخرجات تكون تابعة الأخيرة)،أي تتم إزالة تعريف الوظائف السابقة بالكامل.

Overloading

في جافا سكريبت يمكن ان تقبل الوظائف أي عدد من معاملات (معلمات) ،وأنواع التي تأخذها الوظيفة غير محددة، هذا يدل على ان وظائف جافا سكريبت ليس لها توقيعات ()،أي عدم وجود تواقيع الوظيفة ايضاً عدم وجود تحميل زائد على الوظيفة.

أي يمكن استرداد عدد المعاملات (المعلمات) التي تم تمريرها باستخدام كائن الوسيطات (arguments object).

```
function myFirst(message){
    if(arguments.length===0){
        message ="Hello";}
        console.log(message);
    }
      myFirst("goodbye");
```

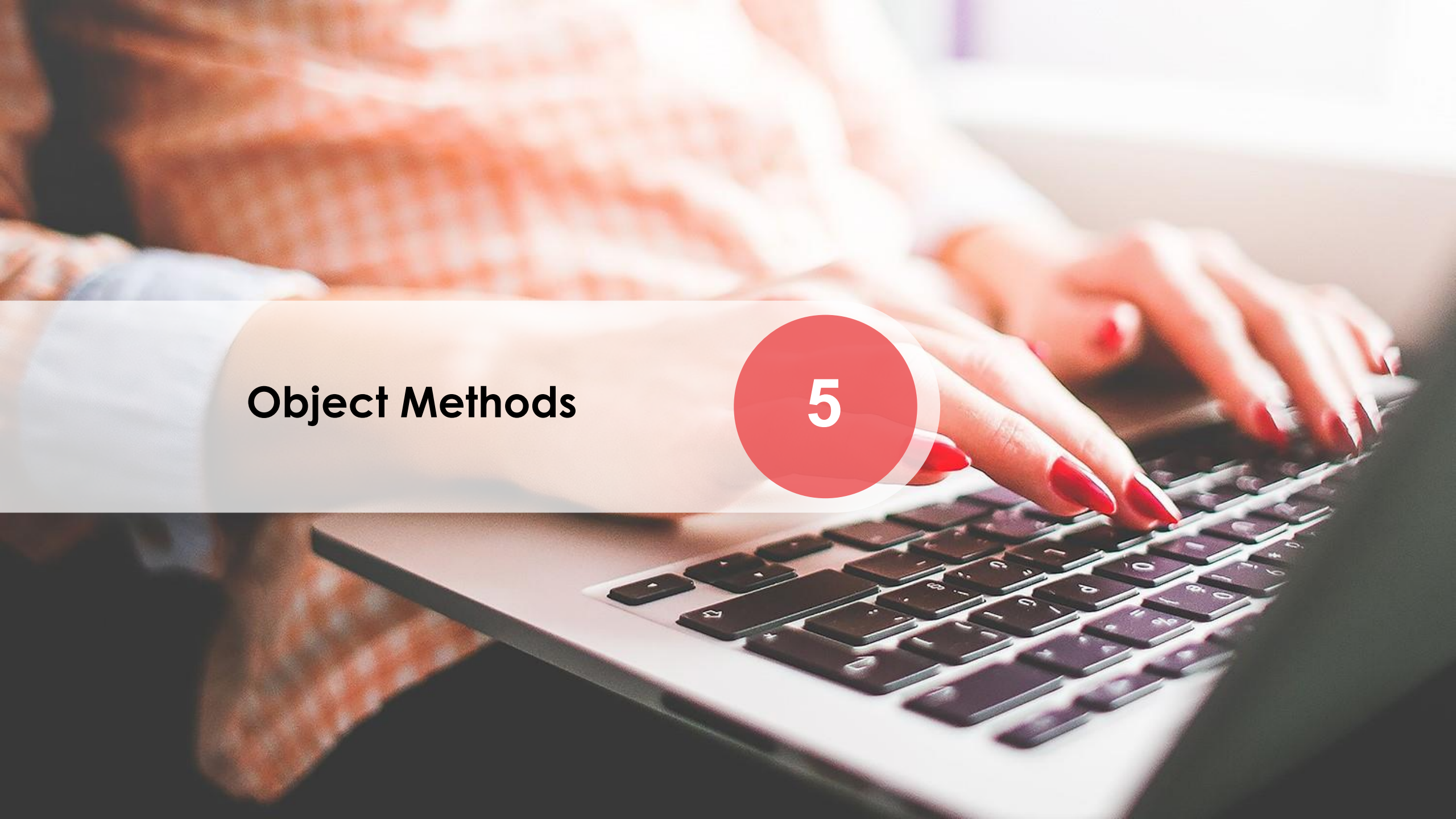
```
goodbye
```

outputs

تتصرف وظيفة myFirst بشكل مختلف بناءً على عدد المعاملات(معلمات) التي تم تمريرها، إذا لم يتم تمرير أي معلمات (arguments. Length === 0) فيتم استخدام رسالة افتراضية ،هذا اكثر تعقيداً من التحميل الزائد للوظائف في اللغات الأخرى ،ولكن النتيجة نفسها .

Object Methods

5



Object Methods

يمكنك إضافة وإزالة الخصائص من الأشياء في أي وقت. عندما تكون قيمة الخاصية دالة في الواقع ، تعتبر الخاصية طريقة. يمكنك إضافة طريقة إلى كائن بنفس الطريقة التي تضيف بها خاصية. على سبيل المثال ، في الكود التالي ، يتم تعيين كائن حرفي إلى متغير الشخص بخاصية اسم وطريقة تسمى sayName

```
> var person = {
  name: "Ali",
  sayName: function() {
  console.log(person.name);
  }
  };
  person.sayName();
```

```
Ali
```

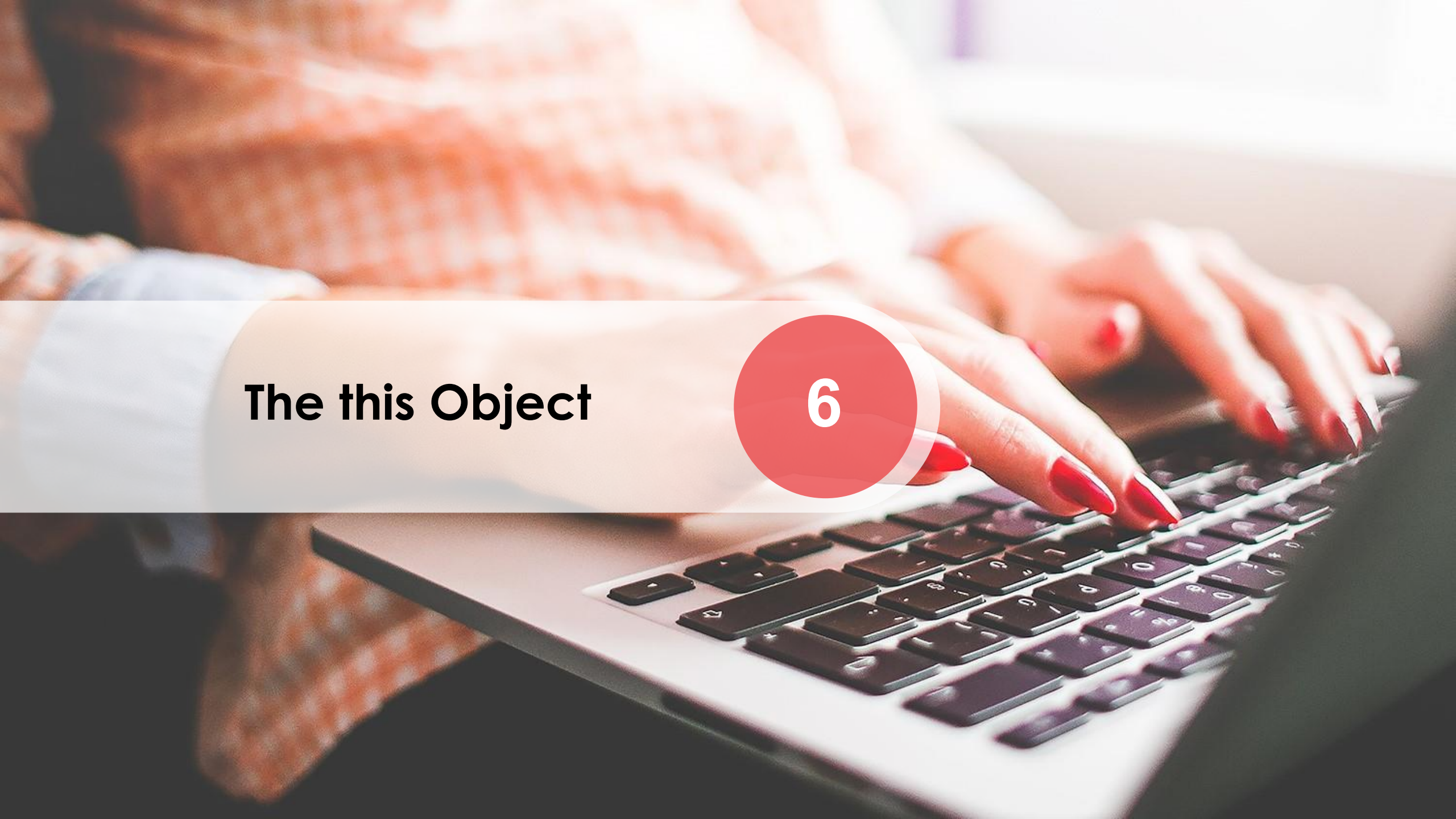
ان بناء الجملة لخاصية بيانات وطريقة هي بالضبط المعرف نفسه متبوعاً بنقطتين والقيمة.

طرق جافا سكريبت هي إجراءات يمكن تنفيذها على الكائنات.

تشير طريقة sayName () إلى person.name مباشرةً ، مما يؤدي الى انشاء اقتران وثيق بين الأسلوب والكائن.

The this Object

6



The this Object

- تتصرف هذا الكلمة (this) الخاصة بوظيفة في جافا سكريبت مقارنة باللغات الأخرى ، وتحتوي على بعض الاختلافات.
- عندما يتم استدعاء دالة أثناء إرفاقها بكائن ، فإن قيمة هذا تساوي هذا الكائن افتراضيًا. لذلك ، بدلاً من الإشارة مباشرة إلى كائن داخل طريقة ، يمكنك الرجوع إلى هذا بدلاً من ذلك.

```
> var person = {
  name: "Ali",
  sayName: function() {
  console.log(this.name);
  }
  };
  person.sayName();
  Ali
```

ربما لاحظت شيئًا غريبًا، تستخدم طريقة say Name () بشكل صريح person.name، مما يؤدي إلى علاقة وثيقة بين الأسلوب والكائن. لمجموعة متنوعة من الأسباب ، هذا أمر إشكالي. للبدء ، إذا قمت بتغيير اسم المتغير ، يجب أن تتذكر أيضًا تعديل مرجع الأسلوب إلى هذا الاسم.

ثانيًا ، تجعل هذه التبعية الوثيقة من المستحيل إعادة استخدام نفس الوظيفة عبر كائنات متعددة. لحسن الحظ ، فإن JavaScriptلديها حل لهذه المشكلة .
في JavaScript، يحتوي كل نطاق على هذا الكائن الذي يمثل كائن استدعاء الوظيفة. هذا هو الكائن العام في النطاق العام (نافذة في متصفحات الويب).

The this Object

```
> function sayNameForAll() {
  console.log(this.name);
  }
  var person1 = {
  name: "Mohammed",
  sayName: sayNameForAll
  };
  var person2 = {
  name: "Esam",
  sayName: sayNameForAll
  };
  var name = "Hani";
  person1.sayName();
  person2.sayName();
  sayNameForAll();
  Mohammed
  Esam
  Hani
```

- في هذا المثال ، اولاً يتم تعريف دالة تسمى sayName ومن ثم انشاء اثنين من الكائنات الحرفية يعينان sayName ليكونا مساوياً لوظيفة sayNameForAll .
- الدوال هي مجرد قيم مرجعية ، يمكن تعيينها كقيم خاصة على أي عدد من الكائنات .
- عندما يتم استدعاء sayName على person1 فتكون المخرجات "Mohammed" ، وعند استدعاؤها على person2 فتكون المخرجات " Essam "، يعرف الجزء الأخير من هذا المثال متغيرًا شاملاً يسمى name.عندما يتم استدعاء sayNameForAll () مباشرة ، فإنه ينتج "Hani" ، لأن المتغير العام يعتبر خاصية للكائن العام.

Changing this

- في JavaScript، تعد القدرة على استخدام هذه القيمة من الوظائف ومعالجتها أمراً ضرورياً للبرمجة الجيدة الموجهة للكائنات.
- يمكن استخدام الوظائف في مجموعة متنوعة من المواقف، ويجب أن تكون قادرة على العمل فيها جميعاً، وعلى الرغم من حقيقة أن هذا عادةً ما يتم تعيينه تلقائياً،
- إلا أنه يمكنك تعديل قيمته لتحقيق أهداف مختلفة.
- يمكنك تعديل قيمة هذا باستخدام واحدة من ثلاث طرق للوظائف .
- ضع في اعتبارك ان الوظائف هي كائنات(objects)، ومثلما يمكن ان يكون للكائنات طرق ، يمكن للوظائف كذلك.

The call() Method

- الطريقة الأولى لتغيير الامر هي call()، والتي تنفذ الوظيفة مع هذه القيمة والمعاملات المعطاة (المعلمات،)، القيمة التي يجب ان يكون مساوياً لها عند تنفيذ الوظيفة هي المعلمة الأولى ل call().
- المعلمات (المعاملات) التي يجب توفيرها في الوظيفة، قمت بتغير say NameForAll() لكي تأخذ معلمة.

```
> function sayNameForAll(label) {
  console.log(label + ":" + this.name);
  }
  var person1 = {
  name: "Mohammed"
  };
  var person2 = {
  name: "Esam"
  };
  var name = "Hani";
  sayNameForAll.call(this, "global");
  sayNameForAll.call(person1, "person1");
  sayNameForAll.call(person2, "person2");
  global:Hani
  person1:Mohammed
  person2:Esam
```

يقبل sayNameForAll () معلمة واحدة ، تستخدم كتسمية لقيم الإخراج في المثال ، يتم استدعاء الوظيفة ثلاث مرات ،بحكم لأنه يتم الوصول الى الوظيفة ككائن (object) بدلاً من رمز لتنفيذه، لا توجد أقواس بعد اسم الوظيفة .
يتم استدعاء الوظيفة المتطابقة مرتين أخريين ، مرة واحدة لكل من الشخصين.نظرًا لاستخدام طريقة call () ، فلن تضطر إلى إضافة الوظيفة إلى كل كائن يدويًا ؛ بدلاً من ذلك ، تقوم بتوفير قيمة هذا بدلاً من السماح لمحرك JavaScriptبالقيام بذلك نيابةً عنك.

The apply( ) Method

- apply() هو طريقة الوظيفة الثانية التي يمكنك استخدامها لمعالجة this() .
- يتطابق أسلوب apply() مع call() فيما عدا انه لا يقبل سوى معلمتين (parameters):
- القيمة الخاصة بذلك والمصفوفة او الكائن object يشبه المصفوفة مع المعلمات لإرساله الى الوظيفة (أي انه يمكن استخدام كائن وسيطات arguments باعتباره المعلمة الثانية.
- نتيجة لذلك ، بدلاً من تحديد كل معلمة بشكل منفصل باستخدام call () ، يمكنك فقط إرسال المصفوفات ل apply()كوسيطة ثانية .خلاف ذلك ، استدعاء apply() و call() لهما نفس السلوك.

The bind( ) Method

Bind() هي الوظيفة الثالثة للتعديل، وهي تختلف عن الطريقتين السابقتين ، هذه القيمة للوظيفة الجديدة هي المعلمة الأولىbind() .
يتم تسمية جميع الوسائط الأخرى بالمعلمات التي يجب تعيينها في الوظيفة الجديدة بشكل دائم ،لا يزال من الممكن تمرير أي معلمات لم يتم تعيينها بشكل دائم لاحقاً.

```
function sayNameForAll(label) {
console.log(label + ":" + this.name);
}
var person1 = {
name: "Nicholas"
};
var person2 = {
name: "Greg"
};
var sayNameForPerson1 = sayNameForAll.bind(person1);
sayNameForPerson1("person1");
var sayNameForPerson2 = sayNameForAll.bind(person2, "person2");
sayNameForPerson2();
person2.sayName = sayNameForPerson1;
person2.sayName("person2");
person1:Nicholas
person2:Greg
person2:Nicholas
undefined
```

نظرًا لعدم احتواء sayNameForPerson1 () على معلمات ، لا يزال يتعين عليك توفير تسمية للنتيجة.
لا تقوم الطريقة sayNameForPerson2 () فقط بربط هذا بـ person2، ولكنها تربط أيضًا المعلمة الأولى بـ "person2".
نتيجة لذلك ، يمكن استدعاء sayNameForPerson2 () دون أي وسيطات أخرى.
في القسم الأخير من هذا المثال ، تم تطبيق sayNameForPerson1 () على person2 بالاسم sayName.
نظرًا لأن الوظيفة مرتبطة ، على الرغم من أن sayNameForPerson1 أصبحت الآن دالة على person2، فإن قيمة هذا لا تتغير. لا يزال يتم إرجاع قيمة person1.name بواسطة الطريقة.

Summary

نظرًا لأن وظائف JavaScriptهي أيضًا كائنات ، يمكن الوصول إليها ونسخها والكتابة فوقها ومعاملتها تمامًا مثل أي قيمة كائن أخرى .إن أهم تمييز بين وظيفة JavaScriptوالكائنات الأخرى هو وجود خاصية داخلية خاصة تسمى [[Call]]، والتي تحتوي على تعليمات تنفيذ الوظيفة.

يبحث عامل التشغيل type ofعن كائن لهذه الخاصية الداخلية ويعيد "الوظيفة" إذا تم العثور عليها.

التعريفات والتعبيرات هما نوعان من القيم الحرفية للوظيفة .

توجد إعلانات الوظائف في الجزء العلوي من السياق الذي تم الإعلان عنه فيه ، مع وجود اسم الوظيفة على يمين الكلمة الأساسية للوظيفة.

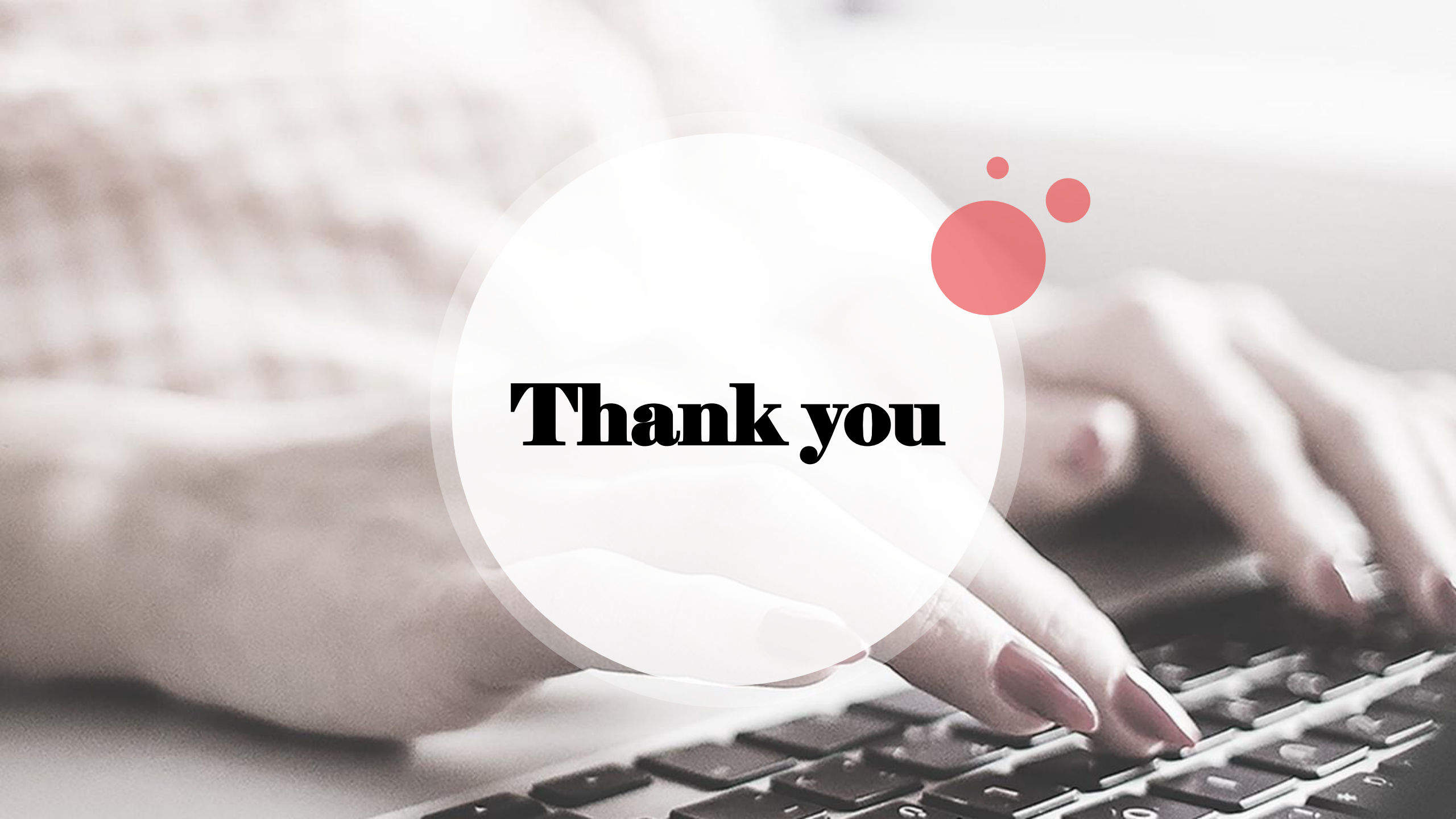
عندما يمكن أيضًا استخدام قيم بديلة ، مثل تعبيرات التخصيص أو معلمات الوظيفة أو نتيجة إرجاع دالة أخرى ، يتم استخدام تعبيرات الوظائف .

يوجد مُنشئ الدالة لأن الوظائف هي كائنات .

يمكنك استخدام مُنشئ الوظيفة لإنشاء وظائف جديدة ، ولكن لا يُنصح بذلك لأنه قد يجعل الكود الخاص بك أكثر صعوبة في الفهم وتصحيح الأخطاء.

ومع ذلك ، فمن شبه المؤكد أنك ستواجهها في ظروف لا يُعرف فيها الشكل الحقيقي للوظيفة حتى وقت التشغيل .

لفهم كيفية عمل البرمجة الشيئية في JavaScript، ستحتاج إلى فهم قوي للوظائف،نظرًا لأن JavaScriptليس لديها فكرة عن فئة ، يجب أن تعتمد على الوظائف والكائنات الأخرى لأداء التجميع والوراثة.


Thank you