13 декабря 2022 г.

Function Expression

Функция в JavaScript – это не магическая языковая структура, а особого типа значение.

Синтаксис, который мы использовали до этого, называется *Function Declaration* (Объявление Функции):

```
function sayHi() {
  alert( "Привет" );
}
```

Существует ещё один синтаксис создания функций, который называется *Function Expression* (Функциональное Выражение).

Данный синтаксис позволяет нам создавать новую функцию в середине любого выражения.

Это выглядит следующим образом:

```
let sayHi = function() {
  alert( "Привет" );
};
```

Здесь мы можем видеть переменную `sayHi`, получающую значение, новую функцию, созданную как `function() { alert("Привет"); }`.

Поскольку создание функции происходит в контексте выражения присваивания (с правой стороны от `=`), это *Function Expression*.

Обратите внимание, что после ключевого слова `function` нет имени. Для Function Expression допускается его отсутствие.

Здесь мы сразу присваиваем её переменной, так что смысл этих примеров кода один и тот же: "создать функцию и поместить её в переменную `sayHi`".

В более сложных ситуациях, с которыми мы столкнёмся позже, функция может быть создана и немедленно вызвана, или запланирована для дальнейшего выполнения, нигде не сохраняясь, таким образом, оставаясь анонимной.

Функция – это значение

Давайте повторим: независимо от того, как создаётся функция – она является значением. В обоих приведённых выше примерах функция хранится в переменной `sayHi`.

Мы даже можем вывести это значение с помощью `alert`:

```
function sayHi() {
  alert( "Привет" );
}

alert( sayHi ); // выведет код функции
```

Обратите внимание, что последняя строка не вызывает функцию, потому что после `sayHi` нет круглых скобок. Существуют языки программирования, в которых любое упоминание имени функции приводит к её выполнению, но JavaScript к таким не относится.

В JavaScript функция – это значение, поэтому мы можем обращаться с ней как со значением. Приведённый выше код показывает её строковое представление, которое является её исходным кодом.

Конечно, функция – это особое значение, в том смысле, что мы можем вызвать её как `sayHi()`.

Но всё же это значение. Поэтому мы можем работать с ней так же, как и с другими видами значений.

Мы можем скопировать функцию в другую переменную:

```
function sayHi() {   // (1) создаём
  alert( "Привет" );
}

let func = sayHi;    // (2) копируем

func(); // Привет     // (3) вызываем копию (работает)!
sayHi(); // Привет    //     эта тоже все ещё работает (почему бы и нет)
```

Давайте подробно разберём всё, что тут произошло:

1. Объявление Function Declaration `(1)` создаёт функцию и помещает её в переменную с именем `sayHi`.
2. В строке `(2)` мы скопировали её значение в переменную `func`. Обратите внимание (ещё раз): нет круглых скобок после `sayHi`. Если бы они были, то выражение `func = sayHi()` записало бы *результат вызова* `sayHi()` в переменную `func`, а не саму *функцию* `sayHi`.
3. Теперь функция может вызываться как `sayHi()`, так и `func()`.

Мы также могли бы использовать Function Expression для объявления `sayHi` в первой строке:

```
let sayHi = function() { // (1) создаём
  alert( "Привет" );
};

let func = sayHi;
// ...
```

Всё будет работать так же.

Зачем нужна точка с запятой в конце?

У вас мог возникнуть вопрос: Почему в Function Expression ставится точка с запятой `;` на конце, а в Function Declaration нет:

```
function sayHi() {
  // ...
}

let sayHi = function() {
  // ...
};
```

Ответ прост: Function Expression создаётся здесь как `function(...) {...}` внутри выражения присваивания: `let sayHi = …;` . Точку с запятой `;` рекомендуется ставить в конце выражения, она не является частью синтаксиса функции.

Точка с запятой нужна там для более простого присваивания, такого как `let sayHi = 5;` , а также для присваивания функции.

Функции-«колбэки»

Давайте рассмотрим больше примеров передачи функции в виде значения и использования функциональных выражений.

Давайте напишем функцию `ask(question, yes, no)` с тремя параметрами:

`question`

Текст вопроса

`yes`

Функция, которая будет вызываться, если ответ будет «Yes»

`no`

Функция, которая будет вызываться, если ответ будет «No»

Наша функция должна задать вопрос `question` и, в зависимости от того, как ответит пользователь, вызвать `yes()` или `no()` :

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

function showOk() {
  alert( "Вы согласны." );
}

function showCancel() {
  alert( "Вы отменили выполнение." );
}

```

```
// использование: функции showOk, showCancel передаются в качестве аргументов
ask("Вы согласны?", showOk, showCancel);
```

На практике подобные функции очень полезны. Основное отличие «реальной» функции `ask` от примера выше будет в том, что она использует более сложные способы взаимодействия с пользователем, чем простой вызов `confirm`. В браузерах такие функции обычно отображают красивые диалоговые окна. Но это уже другая история.

Аргументы `showOk` и `showCancel` функции `ask` называются *функциями-колбэками* или просто *колбэками*.

Ключевая идея в том, что мы передаём функцию и ожидаем, что она вызовется обратно (от англ. «call back» – обратный вызов) когда-нибудь позже, если это будет необходимо. В нашем случае, `showOk` становится *колбэком* для ответа «yes», а `showCancel` – для ответа «no».

Мы можем переписать этот пример значительно короче, используя Function Expression:

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Вы согласны?",
  function() { alert("Вы согласились."); },
  function() { alert("Вы отменили выполнение."); }
);
```

Здесь функции объявляются прямо внутри вызова `ask(...)`. У них нет имён, поэтому они называются *анонимными*. Такие функции недоступны снаружи `ask` (потому что они не присвоены переменным), но это как раз то, что нам нужно.

Подобный код, появившийся в нашем скрипте выглядит очень естественно, в духе JavaScript.

Функция – это значение, представляющее «действие»

Обычные значения, такие как строки или числа представляют собой *данные*.

Функции, с другой стороны, можно воспринимать как *действия*.

Мы можем передавать их из переменной в переменную и запускать, когда захотим.

Function Expression в сравнении с Function Declaration

Давайте разберём ключевые отличия Function Declaration от Function Expression.

Во-первых, синтаксис: как отличить их друг от друга в коде.

- *Function Declaration*: функция объявляется отдельной конструкцией «function…» в основном потоке кода.

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

- *Function Expression*: функция, созданная внутри другого выражения или синтаксической конструкции. В данном случае функция создаётся в правой части «выражения присваивания» `=` :

```
// Function Expression
let sum = function(a, b) {
  return a + b;
};
```

Более тонкое отличие состоит в том, *когда* создаётся функция движком JavaScript.

Function Expression создаётся, когда выполнение доходит до него, и затем уже может использоваться.

После того, как поток выполнения достигнет правой части выражения присваивания `let sum = function…` – с этого момента, функция считается созданной и может быть использована (присвоена переменной, вызвана и т.д.).

С Function Declaration всё иначе.

Function Declaration может быть вызвана раньше, чем она объявлена.

Другими словами, когда движок JavaScript *готовится* выполнять скрипт или блок кода, прежде всего он ищет в нём Function Declaration и создаёт все такие функции. Можно считать этот процесс «стадией инициализации».

И только после того, как все объявления Function Declaration будут обработаны, продолжится выполнение.

В результате функции, созданные как Function Declaration, могут быть вызваны раньше своих определений.

Например, так будет работать:

```
sayHi("Вася"); // Привет, Вася

function sayHi(name) {
  alert( `Привет, ${name}` );
}
```

Функция `sayHi` была создана, когда движок JavaScript подготавливал скрипт к выполнению, и такая функция видна повсюду в этом скрипте.

…Если бы это было Function Expression, то такой код вызвал бы ошибку:

```
sayHi("Вася"); // ошибка!

let sayHi = function(name) {  // (*) магии больше нет
  alert( `Привет, ${name}` );
};
```

Функции, объявленные при помощи Function Expression, создаются тогда, когда выполнение доходит до них. Это случится только на строке, помеченной звёздочкой `(*)` . Слишком поздно.

Ещё одна важная особенность Function Declaration заключается в их блочной области видимости.

В строгом режиме, когда Function Declaration находится в блоке `{...}` , функция доступна везде внутри блока. Но не снаружи него.

Для примера давайте представим, что нам нужно объявить функцию `welcome()` в зависимости от значения переменной `age`, которое мы получим во время выполнения кода. И затем запланируем использовать её когда-нибудь в будущем.

Если мы попробуем использовать Function Declaration, это не заработает так, как задумывалось:

```
let age = prompt("Сколько Вам лет?", 18);

// в зависимости от условия объявляем функцию
if (age < 18) {

  function welcome() {
    alert("Привет!");
  }

} else {

  function welcome() {
    alert("Здравствуйте!");
  }

}

// ...не работает
welcome(); // Error: welcome is not defined
```

Это произошло, так как объявление Function Declaration видимо только внутри блока кода, в котором располагается.

Вот ещё один пример:

```
let age = 16; // возьмём для примера 16

if (age < 18) {
  welcome();               // \   (выполнится)
                           //  |
  function welcome() {     //  |
    alert("Привет!");      //  |  Function Declaration доступно
  }                        //  |  во всём блоке кода, в котором объявлено
                           //  |
  welcome();               // /   (выполнится)

} else {

  function welcome() {
    alert("Здравствуйте!");
  }
}

// здесь фигурная скобка закрывается,
// поэтому Function Declaration, созданные внутри блока кода выше -- недоступн

welcome(); // Ошибка: welcome is not defined
```

Что можно сделать, чтобы `welcome` была видима снаружи `if`?

Верным подходом будет воспользоваться функцией, объявленной при помощи Function Expression, и присвоить значение `welcome` переменной, объявленной снаружи `if`, что обеспечит нам нужную видимость.

Такой код заработает, как ожидалось:

```
let age = prompt("Сколько Вам лет?", 18);

let welcome;

if (age < 18) {

  welcome = function() {
    alert("Привет!");
  };

} else {

  welcome = function() {
    alert("Здравствуйте!");
  };

}

welcome(); // теперь всё в порядке
```

Или мы могли бы упростить это ещё сильнее, используя условный оператор `?`:

```
let age = prompt("Сколько Вам лет?", 18);

let welcome = (age < 18) ?
  function() { alert("Привет!"); } :
  function() { alert("Здравствуйте!"); };

welcome(); // теперь всё в порядке
```

Когда использовать Function Declaration, а когда Function Expression?

Как правило, если нам понадобилась функция, в первую очередь нужно рассматривать синтаксис Function Declaration, который мы использовали до этого. Он даёт нам больше свободы в том, как мы можем организовывать код. Функции, объявленные таким образом, можно вызывать до их объявления.

Также функции вида `function f(…) {…}` чуть более заметны в коде, чем `let f = function(…) {…}`. Function Declaration легче «ловятся глазами».

…Но если Function Declaration нам не подходит по какой-то причине, или нам нужно условное объявление (мы рассмотрели это в примере выше), то следует использовать Function Expression.

Итого

- Функции – это значения. Они могут быть присвоены, скопированы или объявлены в любом месте кода.

- Если функция объявлена как отдельная инструкция в основном потоке кода, то это “Function Declaration”.
- Если функция была создана как часть выражения, то это “Function Expression”.
- Function Declaration обрабатываются перед выполнением блока кода. Они видны во всём блоке.
- Функции, объявленные при помощи Function Expression, создаются только когда поток выполнения достигает их.

В большинстве случаев, когда нам нужно объявить функцию, Function Declaration предпочтительнее, т.к функция будет видна до своего объявления в коде. Это даёт нам больше гибкости в организации кода, и, как правило, делает его более читабельным.

Исходя из этого, мы должны использовать Function Expression только тогда, когда Function Declaration не подходит для нашей задачи. Мы рассмотрели несколько таких примеров в этой главе, и увидим ещё больше в будущем.

Предыдущий урок | Следующий урок

Поделиться

Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen...)
- Если что-то непонятно в статье — пишите, что именно и с какого места.

238 Комментариев

Войти

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ ИЛИ ЧЕРЕЗ DISQUS ?

Имя

27 Share

Лучшие Новые Старые

AlexMarkoff
5 дней назад edited

Сорян, но чет формат изучения совсем не мое, некоторые вещи через одно место объяснены а так как я тупой то пойду лучше через ютуб учить.

2 0 Ответить

Denis Butyrskiy
13 дней назад

Ребят, всем привет! Изучаю JavaScript.

Главная цель⚡ - «С нуля до Junior Frontend Developer за 7 месяцев». На протяжении этих 7 месяцев буду изучать JS и записывать еженедельные **видеоотчёты** об этом.

Всё здесь:
YouTube по названию канала **Denis Butyrskiy**
ТГ @itway_chat - обсуждаем всё, что касается обучения

Давайте объединяться, общаться, делиться друг с другом успехами и неудачами. Это сильно помогает пройти сложные моменты, уже проверено. Под каждым видео есть вся нужная информация👌

1 0 Ответить

KK

Katya Klep
19 дней назад

В первой задаче function ask(question,yes,no){
If (confirm(question)) yes()
else no();
} Как понять если вслух прочитать получается "если задаем вопрос получаем ответ yes, в противном случае no".? И почему после yes no стоят скобки?

1 0 Ответить

Dima Kim → Katya Klep
19 дней назад

//создается функция ask(question, yes, no), где question, yes, no - переменные этой функции
/* if (confirm(question)) yes() else no(); - короткая запись условия, если функция confirm(question)- а это виндов функция модального окна, которая возвращает true если нажата ок, и false если cancel, если true вызывается функция yes() иначе если вы нажали

cancel и функция confirm вернула false то сработает функция no()

вызова нужно указать ();
про alert я надеюсь не надо рассказывать что это тоже модальное окно виндовс
*/

function ask(question, yes, no) {
if (confirm(question)) yes()
else no();
}

показать больше

2 0 Ответить

Akylbek → Dima Kim
19 дней назад

Thanks!

0 0 Ответить

KK **Katya Klep** → Dima Kim
19 дней назад

Спасибо огромное!)))

1 0 Ответить

KK **Katya Klep** → Katya Klep
19 дней назад

И еще на деле все это на русском показывается, как так? Если мы пишем yes, no на англ

0 0 Ответить

P **PVG**
20 дней назад

Function Expression - очень похоже на делегат в C#. Только в JS в одну переменную не получится запихнуть вызов нескольких функций или возможно я об этом пока не знаю.

0 0 Ответить

Road to WEB 2023
22 дня назад

✅ функции expression
давайте вместе ботать, пытаюсь вкатиться в веб-разработку за январь, roadtoweb2023

0 0 Ответить

PP **Paul Pumphut**
месяц назад

Хмм.... пока все еще не понял зачем нужны Function Expression если есть Function Declaration.

Вроде же должно на все хватать и удобно везде использовать Function Declaration.

1 2 Ответить

Atis → Paul Pumphut
месяц назад edited

Ничего, в будущем когда дойдешь до событий поймешь в чем весь кайф. Сейчас главное знать, что так можно

2 0 Ответить

Admin
3 месяца назад

Сообщество школы фронтенд разработчиков, где вам всегда помогут и расскажут, почему что-то не работает в вашем коде)

Найти легко в тг - @babyfrontent

0 1 Ответить

Tazor
3 месяца назад

Я так понял, браузер понимает на что нажал пользователь за счет этого куска кода:

```
if (confirm(question)) yes()
else no()
```

Или это не так?

0 0 Ответить

Kiyotaka → Tazor
3 месяца назад

confirm() - это метод который возвращает значения true (при нажатии на ок) или false (при нажатии отмена или escape)

5 0 Ответить

Комментарий был удален.

Kiyotaka → Guest
3 месяца назад

я не понимаю, зачем вы попытались меня исправить. confirm() как и alert() как и prompt() - это МЕТОД объекта Window. Есть ли смысл обьяснять отличия понятий функции и метода дальше?))

1 0 Ответить

Vladimir Degtyarev → Kiyotaka
месяц назад

Есть)
С чего это метод вдруг стал не функцией?

0 0 Ответить

En Kratia → Kiyotaka

2 месяца назад

глупость написал, прошу прощения.

0 0 Ответить

Candice

3 месяца назад

Wow. Looks good.

0 0 Ответить

Alexander Mandrov

4 месяца назад

Обстоятельства и желания сподвигли меня на создание собственного канала. Там я рассказываю о жизни и работе, о том, как стал программистом в 20 лет, коротко и по делу.

ТГ @unsleeping706

0 1 Ответить

Загрузить ещё комментарии