

Раздел

[Основы JavaScript](#)

Навигация по уроку

Функции-«колбэки»

Function Expression в
сравнении с Function
Declaration

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)
[→ Основы JavaScript](#) 1-го октября 2020

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):

```
1 function sayHi() {  
2     alert( "Привет" );  
3 }
```

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

Оно выглядит вот так:

```
1 let sayHi = function() {  
2     alert( "Привет" );  
3 };
```

В коде выше функция создаётся и явно присваивается переменной, как любое другое значение. По сути без разницы, как мы определили функцию, это просто значение, хранимое в переменной `sayHi`.

Смысл обоих примеров кода одинаков: "создать функцию и поместить её значение в переменную `sayHi`".

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

```
1 function sayHi() {  
2     alert( "Привет" );  
3 }  
4  
5 alert( sayHi ); // выведет код функции
```

Обратите внимание, что последняя строка не вызывает функцию `sayHi`, после её имени нет круглых скобок. Существуют языки программирования, в которых любое упоминание имени функции совершает её вызов. JavaScript – не один из них.

В JavaScript функции – это значения, поэтому мы и обращаемся с ними, как со значениями. Код выше выведет строковое представление функции, которое является её исходным кодом.

Конечно, функция – не обычное значение, в том смысле, что мы можем вызвать его при помощи скобок: `sayHi()`.

Но всё же это значение. Поэтому мы можем делать с ним то же самое, что и с любым другим значением.

Мы можем скопировать функцию в другую переменную:

```
1 function sayHi() {    // (1) создаём  
2     alert( "Привет" );  
3 }  
4  
5 let func = sayHi;    // (2) копируем  
6  
7 func(); // Привет    // (3) вызываем копию (работает)!  
8 sayHi(); // Привет    // прежняя тоже работает (поче
```

Раздел

[Основы JavaScript](#)

Навигация по уроку

Функции-«колбэки»

Function Expression в
сравнении с Function
Declaration

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



Давайте подробно разберём всё, что тут произошло:

1. Объявление Function Declaration (1) создало функцию и присвоило её значение переменной 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 быть вызвана с помощью обеих переменных `sayHi()` и `func()`.

Заметим, что мы могли бы использовать и Function Expression для того, чтобы создать `sayHi` в первой строке:

```
1 let sayHi = function() {  
2   alert( "Привет" );  
3 };  
4  
5 let func = sayHi;  
6 // ...
```

Результат был бы таким же.

i 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:

```
1 function sayHi() {  
2   // ...  
3 }  
4  
5 let sayHi = function() {  
6   // ...  
7 };
```

Ответ прост:

- Нет необходимости в ; в конце блоков кода и синтаксических конструкций, которые их используют, таких как `if { ... }`, `for { }`, `function f { }` и т.д.
- Function Expression использует внутри себя инструкции присваивания `let sayHi = ...`; как значение. Это не блок кода, а выражение с присваиванием. Таким образом, точка с запятой не относится непосредственно к Function Expression, она лишь завершает инструкцию.

Функции-«колбэки»

Рассмотрим ещё примеры функциональных выражений и передачи функции как значения.

Давайте напомним 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()`:

Раздел

Основы JavaScript

Навигация по уроку

Функции-«колбэки»

Function Expression в сравнении с Function Declaration

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
1 function ask(question, yes, no) {
2   if (confirm(question)) yes()
3   else no();
4 }
5
6 function showOk() {
7   alert( "Вы согласны." );
8 }
9
10 function showCancel() {
11   alert( "Вы отменили выполнение." );
12 }
13
14 // использование: функции showOk, showCancel передаются
15 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 функции `ask` ещё называют 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:

```
1 function ask(question, yes, no) {
2   if (confirm(question)) yes()
3   else no();
4 }
5
6 ask(
7   "Вы согласны?",
8   function() { alert("Вы согласились."); },
9   function() { alert("Вы отменили выполнение."); }
10 );
```

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.

i 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

Раздел

[Основы JavaScript](#)

Навигация по уроку

Функции-«колбэки»

Function Expression в
сравнении с Function
Declaration

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



Давайте разберём ключевые отличия Function Declaration от Function Expression.

Во-первых, синтаксис: как определить, что есть что в коде.

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

```
1 // Function Declaration
2 function sum(a, b) {
3   return a + b;
4 }
```

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

```
1 // Function Expression
2 let sum = function(a, b) {
3   return a + b;
4 };
```

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 можно использовать во всем скрипте (или блоке кода, если функция объявлена в блоке).

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:

```
1 sayHi("Вася"); // Привет, Вася
2
3 function sayHi(name) {
4   alert( `Привет, ${name}` );
5 }
```

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 вызовет ошибку:

```
1 sayHi("Вася"); // ошибка!
2
3 let sayHi = function(name) { // (*) магии больше нет
4   alert( `Привет, ${name}` );
5 };
```

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

Раздел

[Основы JavaScript](#)

Навигация по уроку

Функции-«колбэки»

Function Expression в
сравнении с Function
Declaration

Итого

Комментарии

Поделиться



Редактировать на GitHub



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 создать функцию `welcome()` в зависимости от значения переменной `age`, которое мы получим во время выполнения кода. И затем запланируем использовать её когда-нибудь в будущем.

Такой код, использующий Function Declaration, работать не будет:

```
1 let age = prompt("Сколько Вам лет?", 18);
2
3 // в зависимости от условия объявляем функцию
4 if (age < 18) {
5
6     function welcome() {
7         alert("Привет!");
8     }
9
10 } else {
11
12     function welcome() {
13         alert("Здравствуйте!");
14     }
15
16 }
17
18 // ...не работает
19 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:

```
1 let age = 16; // присвоим для примера 16
2
3 if (age < 18) {
4     welcome(); // \ (выполнится)
5                 // |
6     function welcome() { // |
7         alert("Привет!"); // | Function Declaration,
8     } // | во всём блоке кода, в
9         // |
10    welcome(); // / (выполнится)
11
12 } else {
13
14     function welcome() {
15         alert("Здравствуйте!");
16     }
17 }
18
19 // здесь фигурная скобка закрывается,
20 // поэтому Function Declaration, созданные внутри блока
21
22 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 работает, как ожидалось:

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

Раздел

Основы JavaScript

Навигация по уроку

Функции-«колбэки»

Function Expression в
сравнении с Function
Declaration

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
2
3 let welcome;
4
5 if (age < 18) {
6
7     welcome = function() {
8         alert("Привет!");
9     };
10
11 } else {
12
13     welcome = function() {
14         alert("Здравствуйте!");
15     };
16
17 }
18
19 welcome(); // теперь всё в порядке
```

Можно упростить этот код ещё сильнее, используя условный оператор `?:`:

```
1 let age = prompt("Сколько Вам лет?", 18);
2
3 let welcome = (age < 18) ?
4     function() { alert("Привет!"); } :
5     function() { alert("Здравствуйте!"); };
6
7 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 (мы рассмотрели это в примере выше), то можно использовать объявление при помощи Function Expression.

Итого

- Функции – это значения. Они могут быть присвоены, скопированы или объявлены в другом месте кода.
- Если функция объявлена как отдельная инструкция в основном потоке кода, то это Function Declaration.
- Если функция была создана как часть выражения, то считается, что эта функция объявлена при помощи 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 создать функцию, предпочтительно использовать Function Declaration, т.к. функция будет видима до своего объявления в коде. Это позволяет более гибко организовывать код, и улучшает его читаемость.

Таким образом, мы должны прибегать к объявлению функций при помощи Function Expression в случае, когда синтаксис Function Declaration не

подходит для нашей задачи. Мы рассмотрели несколько таких примеров в этой главе, и рассмотрим их ещё больше в будущем.

Раздел

[Основы JavaScript](#)

Навигация по уроку

Функции-«колбэки»

Function Expression в
сравнении с Function
Declaration

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



Проводим [курсы по JavaScript и фреймворкам](#).



Комментарии

перед тем как писать...

© 2007–2020 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфи](#)

