**Function Expression**

Функция в JavaScript это особого типа значение.

***Function Declaration* (Объявление Функции):**

function sayHi() {

alert( "Привет" );

}

Существует ещё один синтаксис создания функций - Function Expression

Данный синтаксис позволяет создавать новую функцию в середине любого выражения.

**Function Expression (Функциональное Выражение)**

let sayHi = function() {

alert( "Привет" );

};

Здесь переменная sayHi, получает значение, новую функцию, созданную как function() { alert("Привет"); }.

Т.к. создание функции происходит в контексте выражения присваивания (с правой стороны от =), это Function Expression.

После ключевого слова function нет имени. Для Function Expression допускается его отсутствие.

**Функция – это значение**

**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.

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

Но, т.к В JavaScript функция – это значение мы можем обращаться с ней как со значением.

Приведённый код показывает её строковое представление, которое является её исходным кодом.

function sayHi() {

alert( "Привет" );

}

alert( sayHi ); // выведет код функции

Так как после sayHi нет круглых скобок последняя строка не вызывает функцию.

Можно 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)

В строке (2) мы скопировали значение функции 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.

**Зачем нужна точка с запятой в конце?**

Почему в Function Expression ставится точка с запятой ; на конце, а в Function Declaration нет:

function sayHi() {

// ...

}

let sayHi = function() {

// ...

};

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»**

Колбэк-функция (или обратный вызов) - это функция, переданная в другую функцию в качестве аргумента, которая затем вызывается по завершению какого-либо действия.

Например есть функция 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»

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

ask("Вы согласны?", showOk, showCancel);

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

можно 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.

**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;

};

**Во-вторых время, 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. ).

**Function Declaration может быть вызвана раньше, чем она объявлена.**

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}` );

}

**блочная области видимости Function Declaration**

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

например нужно объявить функцию welcome() в зависимости от значения переменной age, которое будет во время выполнения кода. И затем запланируем использовать её когда-нибудь в будущем.

Если 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.

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

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

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 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.