Function Expression

(Функциональное Выражение)
Анонимные функции

Функциональное Выражение. Синтаксис

Декларация Функции:

```
function имяФункции( Параметры ) {
    ТелоФункции }
```

Функциональное Выражение:

```
function [имяФункции] (Параметры ) {
    ТелоФункции };
```

У функционального выражения может **отсутствовать имя**. Т.е. сразу после ключевого слова `function` идут круглые скобки, а в них параметры. Функциональные выражения без имени называются **анонимными функциями**.

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

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

Функция в JavaScript – это особого типа значение!

Функциональное выражение — это объявление функции в контексте какого-либо выражения!

Оператор присваивания ожидает справа выражение, именно поэтому функция становится частью выражения.

Function Expression & Function Declaration

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.). То есть нельзя вызвать функцию до ее объявления!

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.

Вызов

```
let showHello = function() {  // (*)
    alert( "Привет" );
};
```

```
//переменная showHello, получает значение -  функцию function showHello(){alert("Привет" );};
```

```
showHello;
```

```
showHello();  //вызов функции должен идти строго после ее объявления
```

```
//Функции, объявленные при помощи Function Expression, создаются тогда, когда выполнение доходит до них (*) .
```

Функция - это значение

Поэтому мы можем работать с ней так же, как и с другими видами значений. Мы можем скопировать функцию в другую переменную:

```
let display = showHello;
```

```
display();
```

Сумма

```
let sum= function(x, y) {
    alert( x + y);
}
sum(2,3)
```

Произведение

```
//  функциональное выражение - вернуть занчение
let mult = function(x, y) {
    return x * y;
}
mult(2,3);
```

Вычислить, с использованием функций

y = (20+5)+(3*4)+(40+6)/(5*6)+2^3

y= sum(20,5)+mult(3,4)+sum(40,6)/mult(5,6)

Проверка четного числа

```
let checkChet = function(n) {
            if (n % 2 == 0) return false;
        return true;
    }

    let num = +prompt('num>', '');
    if (checkChet(num)) alert("четное")
    else alert("нечетное");
```

Проверка простого числа

```
let checkPrime = function(n) {
        for (let i = 2; i < n; i++) {
            if (n % i == 0) return false;
        }
        return true;
    }

    let num = +prompt('num>', '');
    if (checkPrime(num)) alert("простое")
    else alert("составное");
```

Если число четное и простое,

необходимо его увеличить до нечетного

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.

Функции-«колбэки»

Колбэк функция (от английского callback function - функция обратного вызова**)** – это обычная функция, которая просто вызывается внутри другой функции.

Функции могут выступать в качестве параметров других функций:

```
 function hello() {
            return 'Hello, world!';
        }
  alert(hello);//выводит содержимое функции,  значит , нам
ничего не мешает передать функцию в виде параметра для других
функций:

 function display(callback) {
            return callback(); // Вызываем функцию, переданную
в качестве параметра
        }
alert(display(hello));
```

Аргумент *hello* функции *display называется* функцией-колбэк или просто колбэк .

Функция обратного вызова — функция, предназначенная для отложенного выполнения. Проще говоря, она должна быть выполнена после завершения работы другой функции.

```
//приветсвие и прощание должны следовать друг за другом,
ни смотря ни на что
function sayName(name) {
            alert(`Привет ${name} `);
        }

function alertBy() {
            alert(`Пока`);
         }
sayName('Вася',);
alertBy();
```

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

    function alertBy() {
        alert(`Пока`);
    }
    sayName('Вася', alertBy);
```

Аргумент `alertBy` функции **`sayName`** будет колбэком.

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

 function alertBy(name) {
        alert(`Пока ${name} `);
    }
    sayName('Вася', alertBy);
```

Аргумент `alertBy` функции **`sayName`** называется функцией-колбэк или просто колбэк .

Коллбэки позволяют нам быть уверенными в том, что определенный код не начнет исполнение до того момента, пока другой код не завершит исполнение.

```
function sayName(name, callback) {
        alert(`Привет ${name} `);
        setTimeout(callback(name), 3000);
    }

 function alertBy(name) {
        alert(`Пока ${name} `);
    }
    sayName('Вася', alertBy);
```

Аргумент `alertBy` функции **`sayName`** называется функцией-колбэк или просто колбэк .

Коллбэки позволяют нам быть уверенными в том, что определенный код не начнет исполнение до того момента, пока другой код не завершит исполнение.

```
  function readBook() {
            alert('Читай книгу  JS');
        }

 function sayName(name, callback) {
            alert(`Привет ${name} `);
            setTimeout(callback, 3000);
        }

 function alertBy() {
            alert(`Пока  `);
        }
sayName('Вася', alertBy);
readBook();
```

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


sayName('Вася', function(name) {
        alert(`Пока ${name} `);} )
```

Вычисления

```
function sum(x, y){
    return x + y;
}

function mult(x, y){
    return x * y;
}

function operation(x, y, nameoperation){

    let result = nameoperation(x, y);
    alert(result);
}
```

```
alert(sum(10, 6));   // 16
operation(10, 6, sum);  // 16


alert(mult(10, 6));  // 4
operation(10, 6, mult); // 4
```

Аргументы *sum* и *mult* функции *operation* будут колбэками.

Стрелочные функции в JS

```
let func = (arg1, arg2, ...argN) => expression;
```

Это создаёт функцию `func`, которая принимает аргументы `arg1..argN`, затем вычисляет `expression` в правой части с их использованием и возвращает результат. Краткая запись функции выражения

```
let func = function(arg1, arg2, ...argN) {

  return expression;

};
```

Найти сумму

```
let sum= function(x, y) {
    return x + y;
}
```

```
let sum = (x, y) => x + y;
```

Найти произведение

```
let mult= function(x, y) {
    return x * y;
}
```

```
let mult = (x, y) => x * y;
```

Найти квадрат числа

```
let sqr = function(x) {
    return x * x;
}
```

```
let sqr = (x) => x * x;
```

```
//если один аргумент, то можно без скобок
let sqr = x => x * x;
```

Вывести надпись

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

```
//если нет аргумента, то   скобки обязательно! пустые
let showHello = () =>  alert( "Привет" );
```

меньшее из 2 чисел

Меньшее из трех чисел

```
const min = (num1, num2, num3) => {
  let c = num1;
   if (c > num2){c = num2 } else {num2 = c;}
   if (c > num3){c = num3} else {num3 = c;}
 return c;}

alert(min(10, 3, 2))
```

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

 let welcome = (age < 18) ?
 () => alert('Ты еще мал') :
 () => alert("Добро пожаловать");

 welcome();
```

Возвращение функции из функции

Одна функция может возвращать другую функцию:

```
function menu(n){
    if(n==1) return function(x, y){ return x + y;}
    else if(n==2) return function(x, y){ return x - y;}
    else if(n==3) return function(x, y){ return x * y;}
    return function(){ return 0;}
}
// выбираем первый пункт - сложение const action = function(x, y){ return x + y;}
const action = menu(1);
// выполняем функцию и получаем результат в константу result
const result = action(2, 5);
console.log(result);        // 7
```