Язык JavaScript (основы: функции в JavaScript)

ФУНКЦИЯ

В общем смысле, это "подпрограмма", которую можно вызывать из внешнего (или внутреннего, в случае рекурсии) по отношению к функции кода.

Функции позволяют избежать избыточности кода, т.к. функцию создают один раз, а вызывать ее на выполнение можно многократно из разных точек программы.

Часто используемые функции можно объединять в библиотеки.

ФУНКЦИЯ

Функция состоит из последовательности инструкций, называемой телом функции.

Значения могут быть переданы в функцию, а функция вернёт значение.

В JavaScript функции являются объектами - Function

ИМЕНОВАННЫЕ ФУНКЦИИ FUNCTION DECLARATION

Именованные функции доступны везде в области видимости т.е. к ней можно обращаться и до ее создания и после, т.к. они создаются интерпретатором до выполнения кода.

Объявление функции

```
function имяФункции(параметр1, ..., параметрN="значение по-умолчанию" ) {
    тело функции;
    [return;]
}
```

Вызов функции - последовательное выполнение кода из тела функции

```
имяФункции(значение1, …, значениеN);
```

ФУНКЦИИ В БЛОКЕ

При использовании строгого стандарта ‘use strict’;, функция, объявленная в блоке, работает только в блоке. Вне блока ее вызов приведет к ошибке.

```
'use strict';
if (true) {
    function printHello(name) {
        console.log('Hello ' + name);
    }
    printHello('Grisha'); //работает Hello Grisha;
}
printHello(); /* ошибка: Uncaught ReferenceError:
printHello is not defined at arrays.js:9 */
```

ИМЕНА ФУНКЦИЙ И ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

На имена функций распространяются те же правила, что и на имена переменных. Имя функции должно быть глаголом и отображать то, что она делает, т.к. функция – это действие. Функция не должна выполнять больше одного действия.

Возвращаемое значение RETURN.

Функция всегда возвращает значение. Инструкция return указывает, что именно должна вернуть функция. После того, как отработает return функция завершается, код после return выполнен не будет. Функция может вернуть только одно значение. Если return не указан, функция вернет значение по-умолчанию:

- undefined;
- значение this (если функция была создана через оператор new)

ПАРАМЕТРЫ ФУНКЦИИ

При вызове функции ей можно передать данные, которые та использует в зависимости от ее логики.

```
function printHello(name) {
    console.log('Hello ' + name);
}
printHello('Grisha'); // Hello Grisha
```

Функцию можно вызвать с любым количеством аргументов. При объявлении функции необязательные аргументы, как правило, располагают в конце списка. Если параметр не передан при вызове – он считается равным undefined.

ПАРАМЕТРЫ ПО УМОЛЧАНИЮ

можно задать в теле функции:

```
function printUsersInfo(name, surname) {
    surname = surname || 'фамилия не известна';
    console.log('Пользователь: ' + name + ' ' + surname);
}
printUsersInfo('Grisha');
```

Нельзя задать по умолчанию тип данных. Проверку на тип данных нужно делать внутри функции.

а можно “возможность ES6/ES2015”

```
function printUsersInfo(name, surname='фамилия не известна')
{
    console.log('Пользователь: ' + name + ' ' + surname);
}
printUsersInfo('Grisha');
```

Параметры по умолчанию могут быть не только значениями, но и выражениями.

ПЕРЕМЕННОЕ КОЛИЧЕСТВО АРГУМЕНТОВ ARGUMENTS

В JavaScript любая функция может быть вызвана с произвольным количеством аргументов.

Все значения, переданные в функцию при вызове, находятся в объекте arguments (не имеет значения, есть ли они списке параметров и используются ли они в теле функции).

Доступ к элементам arguments осуществляется через [ ], arguments можно перебрать в цикле for. Но массивом arguments не является!

ОПЕРАТОР SPREAD ВМЕСТО ARGUMENTS

“возможность ES6/ES2015”

Чтобы получить именно массив аргументов можно использовать оператор … .

```
function printUsersInfo(name, surname, ...other_info) {
   console.log('Пользователь: ' + name + ' ' + surname + ' ' + other_info);
}
printUsersInfo('Гриша', 'Петров', 26, '+79991234455'); /* Пользователь: Гриша Петров 26,+79991234455 */
```

В other_info попадёт массив всех аргументов, начиная с третьего. other_info – настоящий массив, со всеми доступными массивам методами, в отличие от arguments. Spread должен быть в конце, т.к. в него попадают оставшиеся аргументы.

ОПЕРАТОР SPREAD ПРИ ВЫЗОВЕ ФУНКЦИИ

“возможность ES6/ES2015”

Оператор spread при вызове функции используется для передачи массива параметров как списка.

```
function sum(x, y, z) {
    return x + y + z;
}
let numbers = [1, 2, 3];
let result = sum(...numbers);
console.log(result); // 6
```

ОБЪЯВЛЕНИЕ ФУНКЦИИ FUNCTION EXPRESSION

Функции могут быть созданы внутри выражения, их можно присвоить переменной. Такие функции как правило анонимны. Такие функции можно вызвать только после объявления.

Объявление анонимных функций.

```
let имяПеременной = function(парам1, ... ,
парамN=значение по умолчанию) {
    тело функции;
    [return;] };
```

Вызов функции

```
имяПеременной(значение1, …, значениеN);
```

ФУНКЦИИ ЧЕРЕЗ =>

Вместо

```
let plus = function(a, b) {
    return a + b;
};
plus(2,4);
```

можно записать “возможность ES6/ES2015”

```
let plus = (a, b) => a+b;
console.log(plus(2,4));
```

ФУНКЦИИ ЧЕРЕЗ =>

```
let plus = (a, b) => a+b;
```

Слева от => находится аргумент, справа – выражение, которое нужно вернуть. Если аргумент один, скобки использовать не нужно, если аргументов несколько, их оборачивают в скобки. Если нужно задать функцию без аргументов, то используются пустые скобки:

```
let random = () => Math.random();
```

Когда тело функции достаточно большое, то можно его обернуть в фигурные скобки {…}. Такая функция должна делать явный return.

Функции-стрелки не имеют своего this.

Функции-стрелки не имеют своего arguments.

ЧТО ПОЧИТАТЬ ДОПОЛНИТЕЛЬНО

https://learn.javascript.ru/function-basics - Функции
https://learn.javascript.ru/function-expressions - Function Expression
https://learn.javascript.ru/arrow-functions-basics - Функции-стрелки
https://learn.javascript.ru/rest-parameters-spread-operator - Остаточные параметры и оператор расширения