Функции JavaScript

Функции являются другими фундаментальными блоками JavaScript. Функция является процедурой JavaScript – набором операторов, выполняющим специфическую задачу. Чтобы использовать функцию,
Вы обязаны сначала определить её; затем скрипт может вызвать эту функцию.

Определение функций

Определение функции состоит из ключевого слова **function** и

- Имени функции.
- Списка аргументов, заключённых в скобки и разделяемых запятыми.
- Операторов JavaScript, составляющих содержание функции, заключённых в фигурные скобки { }. Операторы функции могут могут содержать вызовы других функций, определённых в текущем приложении.

Общий вид

- function имя_функции (параметры) { инструкции }

Например, следующий код определяет простую функцию под названием **square**:

- function square(number) { return number * number; }

- Функция square принимает один аргумент -**number**. Функция состоит из одного оператора, который возвращает квадрат аргумента функции. Оператор **return** специфицирует значение, возвращаемое функцией.

- Все параметры передаются в функцию по значению; значение передаётся в функцию, но, если функция изменяет значение этого параметра, это изменение не отражается глобально или в вызывающей функции. Однако, если Вы передаёте в качестве параметра объект и функция изменяет свойства объекта, это изменение видно вне функции

Пример

- function myFunc(theObject) { theObject.make="Toyota" } mycar = {make:"Honda", model:"Accord", year:1998}; x=mycar.make; // возвращает Honda myFunc(mycar); // передаёт функции объект mycar y=mycar.make; // возвращает Toyota (prop было изменено функцией)

Функция может определяться на базе условия. Например, в данном определении функции:

```
if (num == 0) { function myFunc(theObject) {
theObject.make="Toyota" } }
```

функция **myFunc** определена только в том случае, если переменная **num** равна 0. Если **num** не равна 0, функция не определена, и любая попытка выполнить её потерпит неудачу.

Функция может быть также определена внутри выражения. Это называется функцией выражения. Обычно такая функция безымянна/anonymous; она не должна иметь имени.

Это удобно при передаче функции другой функции в качестве аргумента. В примере показана функция **map**, определяемая и вызываемая с анонимной функцией в качестве параметра:

- function map(f,a) { var result=new Array; for (var i = 0; i != a.length; i++) result[i] = f(a[i]); return result; }
- Вызов map(function(x) {return x * x * x}, [0, 1, 2, 5, 10];
- возвращает [0, 1, 8, 125, 1000].

Определение функции ещё не вызывает её выполнения. Определение функции просто именует её и специфицирует действия функции при её вызове. Вызов функции выполняет специфицированные действия с указанными параметрами. Например, если вы определяете функцию **square**, можно будет вызвать её так:

- square(5)
- Здесь функция вызывается с аргументом 5. Функция выполняет свои операторы и возвращает значение 25.

- Аргументы функции – это не только строки или числа. Вы можете также передавать в функцию целый объект. Функция **show_props** (определена в «Объектах и Свойствах») это пример функции, принимающей объект в качестве аргумента.

Функция может быть рекурсивной, то есть может вызывать сама себя. Например, функция вычисления факториала:

```
function factorial(n) { if ((n == 0) || (n == 1)) return 1 else { var result = (n * factorial(n-1) ); return result } }
```

Вы можете затем вычислять факториал от 1 до 5:

```
a=factorial(1) // возвращает 1 b=factorial(2) // возвращает 2 c=factorial(3) // возвращает 6 d=factorial(4) // возвращает 24 e=factorial(5) // возвращает 120
```

В JavaScript имеются предопределённые функции верхнего уровня:

- eval
- isFinite
- isNaN
- parseInt и parseFloat
- Number и String
- encodeURI, decodeURI, encodeURIComponent и decodeURIComponent

- **Функция eval** вычисляет строку кода **JavaScript** без ссылки на конкретный объект. Синтаксис **eval** таков:

- eval(expr)
- где **expr** это вычисляемая строка.

- Если строка представляет собой выражение, **eval** вычисляет это выражение. Если аргументом является один или более операторов **JavaScript**, **eval** выполняет эти операторы.
  Не вызывайте **eval**для вычисления арифметических выражений; **JavaScript** вычисляет арифметические выражения автоматически.

Функция isFinite вычисляет аргумент для определения конечности числа.

Синтаксис **isFinite** таков:

- isFinite(number)
- где **number** это обсчитываемое число.
- Если аргумент имеет значение **NaN**, положительная или отрицательная бесконечность, этот метод возвращает false, иначе возвращает true.

Функция isNaN вычисляет, является ли аргумент **"NaN"** (не-числом). Синтаксис **isNaN**:

- isNaN(testValue)
- где **testValue** это вычисляемое выражение.

- **isNaN** возвращает true, если передано **"NaN"** и false – в ином случае.

Две «разбирающие» функции, **parseInt** и **parseFloat**, возвращают числовое значение, имея в качестве аргумента строку.

- Синтаксис **parseFloat**: parseFloat(str)
- где **parseFloat** разбирает свой аргумент, строку **str**, и пытается возвратить число с плавающей точкой. Если она обнаруживает символ, отличный от знака (+ или -), цифры (0–9), десятичной точки или экспоненты, тогда она возвращает значение до этой позиции и игнорирует этот символ и последующие символы. Если первый символ не может быть конвертирован в число, функция возвращает**"NaN"** (не-число).

Синтаксис **parseInt**: parseInt(str [, radix])

- **parseIn**t разбирает свой первый аргумент, строку **str**, и пытается вернуть целое число со специфицированным основанием (**radix**), обозначенным вторым необязательным аргументом, **radix**. Например,**radix 10** означает конвертацию к десятеричному числу, 8 – восьмеричному, 16 – шестнадцатеричному и так далее. Для **radix** свыше 10, буквы алфавита обозначают цифры больше 9. Например, для 16-ричных чисел (база 16), используются буквы от A до F.

  Если **parseInt** вычисляет символ, который не является числом со специфицированным основанием, она игнорирует это число и все последующие символы и возвратит целое число, разобранное до этой позиции. Если первый символ не может быть конвертирован в число со специфицированным основанием, возвращается **"NaN"**. Функция **parseInt** усекает строку до целочисленного значения.

Функции **Number** и **String** позволяю т конвертировать объект в число или строку. Синтаксис этих функций таков:

- Number(objRef)
- String(objRef)
- где **objRef** это ссылка на объект.

Следующий код конвертирует объект **Date** в читабельную строку:

- D = new Date (430054663215) // возвращается следующее // "Thu Aug 18 04:37:43 GMT-0700 (Pacific Daylight Time) 1983" x = String(D)

- Функции **escape** и **unescape** позволяют вам кодировать и декодировать строки. Функция **escape** возвращает 16-ричное кодированное представление аргумента – набора символов **ISO Latin**. Функция**unescape** возвращает **ASCII**-строку для специфицированного аргумента – 16-ричного кодированного значения.

Синтаксис этих функций таков:

- escape(string)
- unescape(string)

- Эти функции используются в основном в серверном **JavaScript** для кодирования и декодирования пар имя/значение в **URL**.

  Функции **escape** и **unescape** неправильно работают с не – **ASCII** символами.