ТЕМА 3. ФУНКЦИИ

Функция – это блок программного кода на языке JavaScript, который определяется один раз и может вызываться многократно. Функции могут иметь параметры, или аргументы, – локальные переменные, значения которых определяются при вызове функции.

Функции так же позволяют выполнить сценарий, тогда, когда это необходимо, а не сразу как его прочитает браузер.

Определение и вызов функций

Определения функции осуществляется с использованием инструкции function. Она состоит из ключевого слова function, за которым следуют:

- имя функции;
- заключенный в круглые скобки необязательный список аргументов, разделенных запятыми;
- JavaScript инструкции, составляющие тело функции, заключенные в фигурные скобки.

```
function pechat()
{
document.write(“Привет!”);
}
```

Добавление аргументов

Информация, которая нужна функции для выполнения своей задачи и которая не содержится в самом определении функции, должна указываться в виде аргумента. Аргумент представляет собой одно или несколько значений, указываемых в скобках в определении функции. Все это можно увидеть в следующем примере. Переменная oldWages является аргументом функции Wages ().

```
function Wages (oldWages)
{
let newWages = oldWages *1.25;
alert("Ваша новая зарплата   "   + newWages);
}
```

Представьте себе, что аргумент — это что-то вроде переменной. Для аргумента имя указывается по тем же принципам, что и имена переменных. Все, что можно совершать с переменной, можно делать и с аргументом.

А каким же образом аргументу присваивается значение? Это осуществляется тогда, когда происходит обращение к функции. Команды сценария JavaScript или теги HTML на Web-странице указывают нужное значение, когда вызывают функцию. Это действие называется передачей значения функции.

Вызов функции может осуществляться несколькими способами. Самый простой это вызов из сценария. Для вызова функции из сценария достаточно набрать название функции, как только браузер прочитает название функции он сразу начнет ее выполнять.

Вызов функции всегда должен осуществляться после ее объявления.

```
Wages ()
```

Вызов функции может быть так же реакцией на событие. Например:

```
<body onLoad=”Wages()”>
```

В данном примере вызывается функция Wages (), как реакция на событие onLoad. Событие onLoad наступает при окончательной загрузке всего документа html.

Или событие onClick, которое наступает при щелчке мыши на объект.

```
function Wages (oldWages)
{
let newWages = oldWages *1.25;
alert("Ваша новая зарплата   "   + newWages);
}
….
<p onClick=’ Wages(10000)’
```

В этом примере при щелчке мышью на параграф запускается функция Wages() и одновременно аргументу oldWages задается значение 10000.

Добавление нескольких аргументов

Для выполнения действий, можно использовать любое количество аргументов внутри функции. Каждый аргумент должен иметь уникальное имя и быть отделен от другого запятой.

Внесем изменения в предыдущий пример и сделаем так, чтобы коэфицент увеличения зарплаты стал аргументом функции. В данном случае используется два аргумента: OldSalary и coefficientWages.

```
function Wages (oldWages, coefficientWages)
{
let newWages = oldWages * coefficientWages;
alert("Ваша новая зарплата   "   + newWages);
}
….
<p onClick=’ Wages(10000, 1.5)’>
```

В данном примере, функции newWages передается значение двух аргументов. Первому аргументу (oldWages) передается первое значение (10000), второму аргументу (coefficientWages) передается второе значение (1.5).

Параметры по умолчанию

Если параметр не указан, то его значением становится undefined.

Например, вышеупомянутая функция showMessage(from, text) может быть вызвана с одним аргументом:

```
showMessage("Аня");
```

Это не приведёт к ошибке. Такой вызов выведет "Аня: undefined". В вызове не указан параметр text, поэтому предполагается, что text === undefined.

Если мы хотим задать параметру text значение по умолчанию, мы должны указать его после =:

```
function showMessage(from, text = "текст не добавлен") {
  alert( from + ": " + text );
}

showMessage("Аня"); // Аня: текст не добавлен
```

Теперь, если параметр text не указан, его значением будет "текст не добавлен"

В данном случае "текст не добавлен" это строка, но на её месте могло бы быть и более сложное выражение, которое бы вычислялось и присваивалось при отсутствии параметра. Например:

```
function showMessage(from, text = anotherFunction()) {
  // anotherFunction() выполнится только если не передан text
  // результатом будет значение text
}
```

Вычисление параметров по умолчанию

В JavaScript параметры по умолчанию вычисляются каждый раз, когда функция вызывается без соответствующего параметра.

В примере выше anotherFunction() будет вызываться каждый раз, когда showMessage() вызывается без параметра text.

Использование параметров по умолчанию в ранних версиях JavaScript

Ранние версии JavaScript не поддерживали параметры по умолчанию. Поэтому существуют альтернативные способы, которые могут встречаться в старых скриптах.

Например, явная проверка на undefined:

```
function showMessage(from, text) {
  if (text === undefined) {
    text = 'текст не добавлен';
  }
  alert( from + ": " + text );
}
```

…Или с помощью оператора ||:

```
function showMessage(from, text) {
  // Если значение text ложно, тогда присвоить параметру text значение по умолчанию
  text = text || 'текст не добавлен';
  ...
}
```

Область видимости переменных и аргументов

Переменная может быть указана внутри функции, например переменная newWages указывается внутри функции Wages(). Такая переменная называется локальной (Local), так как она указывается в определенном месте в функции. Другие части сценария не знают о существовании локальной переменной, так как она недоступна за пределами функции.

С другой стороны, переменную можно указать за пределами функции. Подобная переменная называется **глобальной (Global)**, так как она доступна всем частям сценария JavaScript, то есть ее могут использовать операторы в любой части сценария.

Внутри тела функции локальная переменная имеет преимущество перед глобальной переменной с тем же именем. Если объявить локальную переменную или параметр функции с тем же именем, что у глобальной переменной, то фактически глобальная переменная будет скрыта. Так, следующий код выводит «локальная переменная»:

Переменные, объявленные внутри функции, видны только внутри этой функции.
Например:

```
function showMessage() {
  let message = "Привет, я JavaScript!"; // локальная переменная

  alert( message );
}

showMessage(); // Привет, я JavaScript!

alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри функции
```

Внешние переменные

У функции есть доступ к внешним переменным, например:

```
let userName = 'Вася';

function showMessage() {
  let message = 'Привет, ' + userName;
  alert(message);
}

showMessage(); // Привет, Вася
```

Функция обладает полным доступом к внешним переменным и может изменять их значение.

Например:

```
let userName = 'Вася';
function showMessage() {
userName = "Петя"; // (1) изменяем значение внешней переменной
let message = 'Привет, ' + userName;
alert(message);
}

alert( userName ); // Вася перед вызовом функции
showMessage();
alert( userName ); // Петя, значение внешней переменной было изменено функцией
```

Внешняя переменная используется, только если внутри функции нет такой локальной.

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю. Например, в коде ниже функция использует локальную переменную userName. Внешняя будет проигнорирована:

```
let userName = 'Вася';

function showMessage() {
  let userName = "Петя"; // объявляем локальную переменную

  let message = 'Привет, ' + userName; // Петя
  alert(message);
}

// функция создаст и будет использовать свою собственную локальную переменную userName
showMessage();
```

```
alert( userName ); // Вася, не изменилась, функция не трогала внешнюю переменную
```

Глобальные переменные

Переменные, объявленные снаружи всех функций, такие как внешняя переменная userName в вышеприведённом коде – называются глобальными.

Глобальные переменные видимы для любой функции (если только их не перекрывают одноимённые локальные переменные).

Желательно сводить использование глобальных переменных к минимуму. В современном коде обычно мало или совсем нет глобальных переменных. Хотя они иногда полезны для хранения важнейших «общепроектовых» данных.

Вызов функции

Вызов (Call) функции может происходить в любой момент, когда нужно, чтобы браузер выполнил команды, содержащиеся в кодовом блоке функции. Вызов функции происходит путем указания ее имени вместе со следующими за ним скобками. Если у функции есть аргументы, то значение каждого аргумента помещается в эти скобки. Нужно указывать значения в том же порядке, что и аргументы внутри определения функции. Каждое значение отделяется запятой. Вот как происходит вызов функции Wages():

```
Wages(10000, 1.5);
```

Обратите внимание на то, что указанное значение (10000) — это прежняя зарплата, а второе значение (1.5) — это коэффицент увеличения зарплаты. Это соответствует порядку аргументов в определении функции Wages () из предыдущего примера, а именно oldWages и coefficientWages.

А что же произойдет, если перепутать значения местами?

```
Wages(6, 500000);
```

Значение 1.5 будет указано для аргумента coefficientWages, а число 10000 – для oldWages.

Если аргументы были строками, например, именем пользователя или паролем, то нужно заключить их в кавычки. Пример:

```
validageLogon('ИвановИван', 'пароль');
```

Вызов функции из HTML

Вызов функции можно произвести через теги HTML на Web-странице. Обычно вызов функции происходит в ответ на какое-либо событие, например при загрузке или выгрузке Web-страницы.

Вызов функции из тегов HTML происходит таким же образом, что и обращение к

ней в сценарии JavaScript, с тем исключением, что вызов функции записывается в виде значения атрибута тега HTML. Допустим, что нужно вызвать функцию в тот момент, когда браузер загружает Web-страницу. Вот что нужно ввести в тег <body> на Web-странице:

```
<body onload = "WelcomeMessage();">
```

А вот что нужно указать для вызова функции, когда пользователь переходит на другую Web-страницу:

```
<body onunload = "GoodbyeMessage();">
```

Функции, вызывающие другие функции

Разработчики на языке JavaScript обычно распределяют сценарий между несколькими функциями, каждая из которых отвечает за определенную часть этого сценария. Вызов одной функции может происходить из другой.

Допустим, что пользователь создал функцию авторизации, которая отвечает за все действия, необходимые для авторизации пользователя и последующего запуска сценария. В состав функции входят диалоговые окна, которые открываются для того, чтобы пользователь ввел имя и пароль. Допустим, что была определена еще одна функция, которая только занимается проверкой имени пользователя и пароля и сообщает, правильно ли они указаны. Функция авторизации передает функции проверки правильности имени пользователя и пароля соответствующие значения, а затем ждет от нее информации о том, правильно ли они указаны. Затем функция авторизации сообщает пользователю, правильно он ввел имя пользователя и пароль или нет.

Возвращаемое значение функции

Функция может совершать действия, а затем возвращать некоторое значение, информирующее об окончании совершения действия и его результате.

Функция «отвечает» команде, вызывающей ее, путем возврата значения с помощью ключевого слова return, за которым следует возвращаемое значение. Вот пример этого действия:

```
function name()
{
return value;
}
```