Тема 7. Функции.

хекслет колледж

Цель занятия:

Сформировать у студентов понимание назначения функций в JavaScript и научить создавать и использовать функции для повторного применения кода.

Учебные вопросы:

1. Понятие функции
2. Объявление и вызов функции
3. Параметры и аргументы
4. Возврат значения (return)
5. Область видимости переменных

1. Понятие функции

Функция — это именованный блок кода, который выполняет определённую задачу и может быть вызван многократно.
Функции позволяют разделять программу на логические части и не повторять одинаковый код.

Зачем нужны функции:

- Повторное использование кода. Например, один и тот же алгоритм можно использовать в нескольких местах программы без копирования кода.
- Упрощение программы. Функции делают код более читаемым и структурированным.
- Инкапсуляция логики. Детали выполнения спрятаны внутри функции, вызывающий код видит только имя функции и её параметры.

Пример функции

```
function greet() {
    console.log("Привет, студент!");
}
```

function — ключевое слово для объявления функции
greet — имя функции
() — скобки для параметров (пока пустые)
{ ... } — тело функции, выполняемое при вызове

Вызов функции

```
greet(); // вызов функции
```

Результат в консоли:

```
Привет, студент!
```

Пример повторного использования

```
function sayHello(name) {
    console.log("Привет, " + name + "!");
}

sayHello("Анна");
sayHello("Иван");
```

Результат:

```
Привет, Анна!
Привет, Иван!
```

Вывод:

- Функция — это способ структурировать код и избежать дублирования.
- Имя функции отражает её назначение.
- Для выполнения функции нужно её вызвать.
- Функции могут принимать параметры, о чём будет рассказано далее.

2. Объявление и вызов функции

Функция объявляется с помощью ключевого слова function, имени и скобок:

```
function имяФункции() {
    // тело функции
}
```

function — объявление функции
имяФункции — идентификатор, по которому вызывается функция
() — скобки для параметров (пока пустые)
{ ... } — блок кода, который выполняется при вызове

Примеры объявлений

```
function greet() {
    console.log("Привет!");
}

function sayHello(name) {
    console.log("Привет, " + name + "!");
}
```

Вызов функции

Чтобы выполнить код внутри функции, её нужно вызвать:

```
greet();              // вывод: Привет!
sayHello("Иван"); // вывод: Привет, Иван!
sayHello("Анна"); // вывод: Привет, Анна!
```

Имя функции + скобки () — вызов функции
Можно вызывать несколько раз, не переписывая код

Вывод:

- Объявление функции — это создание именованного блока кода.
- Вызов функции выполняет код внутри неё.
- Функцию можно вызывать несколько раз с разными аргументами.
- Функции повышают читаемость и удобство работы с повторяющимися действиями.

3. Параметры и аргументы функции

Параметры — это переменные, которые объявляются в скобках функции и служат для передачи данных в функцию.

```
function greet(name) {
    console.log("Привет, " + name + "!");
}
```

name — параметр функции, используемый внутри её тела

Аргументы функции

Аргументы — это реальные значения, которые передаются функции при вызове.

```
greet("Иван"); // "Иван" — аргумент
greet("Анна"); // "Анна" — аргумент
```

Аргументы сопоставляются с параметрами по порядку.

Пример с несколькими параметрами

```
function sum(a, b) {
    console.log(a + b);
}

sum(5, 3);   // 8
sum(10, 20); // 30
```

a и b — параметры
5, 3 или 10, 20 — аргументы при вызове

Значения по умолчанию

Можно задавать значения по умолчанию, если аргумент не передан:

```
function greet(name = "Гость") {
    console.log("Привет, " + name + "!");
}

greet();          // Привет, Гость!
greet("Ирина"); // Привет, Ирина!
```

Значения по умолчанию

Ещё пример:

```
function multiply(a, b = 1) {
    console.log(a * b);
}

multiply(5);   // 5 * 1 = 5
multiply(5, 3); // 15
```

Практические моменты

- Количество аргументов может отличаться от количества параметров:
- Больше аргументов — лишние игнорируются
- Меньше аргументов — недостающие принимают значение undefined или значение по умолчанию

Пример:

```
function greet(name) {
    console.log("Привет, " + name + "!");
}

greet();                    // Привет, undefined!
greet("Ирина", "Анна"); // Привет, Ирина!
```

Вывод:

- Параметры — переменные в объявлении функции.
- Аргументы — конкретные значения, передаваемые при вызове.
- Значения по умолчанию делают функции более гибкими.
- Использование параметров позволяет функции работать с разными данными без переписывания кода.
- Количество аргументов может отличаться от количества параметров

4. Возврат значения (return)

Что такое **return**?

- **return** — это оператор, который позволяет функции вернуть значение после её выполнения.
- Возвращённое значение можно использовать в дальнейшем в программе.
- Без **return** функция выполняет действия, но не даёт результат для использования вне себя.
- **return** завершает выполнение функции сразу после его выполнения.

Синтаксис

```
function имяФункции() {
    ...
    return значение;
}
```

значение может быть числом, строкой, логическим значением или любым выражением.

Пример функции с return

```
function sum(a, b) {
    return a + b;
}

let result = sum(5, 3);
console.log(result); // 8
```

- **sum(5, 3)** возвращает 8
- Значение сохраняется в переменной **result**

Функция без return

Если return не указан, функция возвращает undefined:

```
function greet(name) {
    console.log("Привет, " + name);
}

let value = greet("Ирина"); // вывод: Привет, Ирина
console.log(value);         // undefined
```

Функция выполнила действие (console.log) — это побочный эффект.
Но она не дала значение для использования в других вычислениях.

Побочные эффекты функции

Побочный эффект — это любое действие функции, которое влияет на внешний мир, кроме возврата значения.

Примеры побочных эффектов:

- Вывод в консоль:
- Изменение глобальной переменной:
- Взаимодействие с DOM (будет изучаться позже)

Можно использовать ранний return для выхода из функции при проверках:

```
function checkAge(age) {
    if (age < 0) return "Ошибка: возраст не может быть отрицательным";
    return age >= 18 ? "Совершеннолетний" : "Несовершеннолетний";
}

console.log(checkAge(-5)); // Ошибка: возраст не может быть отрицательным
```

Вывод:

- **return** позволяет функции возвращать результат для дальнейшего использования.
- Без **return** функция выполняет только побочные действия (например, **console.log**).
- Использование **return** делает функции гибкими и позволяет строить сложные выражения.
- Функции без **return** полезны для действий (побочных эффектов).
- Побочные эффекты — это изменения состояния программы или внешней среды.

5. Область видимости переменных

Область видимости (scope) — это часть программы, в которой переменная доступна для использования.
В JavaScript есть две основные области видимости:

Глобальная — переменные доступны во всей программе.
Локальная (функциональная) — переменные доступны только внутри функции, в которой они объявлены.

Глобальная область видимости

Переменные, объявленные вне функций, являются глобальными:

```
let globalVar = "Я глобальная";

function showGlobal() {
    console.log(globalVar); // доступна внутри функции
}

showGlobal();           // Вывод: Я глобальная
console.log(globalVar); // Вывод: Я глобальная
```

- Доступны везде, включая функции
- Использование большого числа глобальных переменных может усложнять код и приводить к ошибкам

Локальная область видимости

Переменные, объявленные внутри функции с let, const или var, доступны только внутри этой функции:

```
function greet() {
    let localVar = "Я локальная";
    console.log(localVar); // доступна
}

greet();                // Я локальная
console.log(localVar); // Ошибка: localVar is not defined
```

Локальные переменные изолированы от внешнего кода
Это предотвращает конфликты имён

Вложенные функции и область видимости

Локальная переменная функции доступна внутри вложенных функций:

```
function outer() {
    let outerVar = "Я внешняя";
    function inner() {
        console.log(outerVar); // доступна вложенной функции
    }
    inner();
}
outer(); // Вывод: Я внешняя
```

Переменные внешней функции видны вложенной, но не наоборот

```
function outer() {
    console.log(innerVar)  // outerVar is not defined
    function inner() {
        let innerVar = "Я внутренняя";
        console.log(innerVar);
    }
    inner();
}
```

Вывод:

- Scope определяет, где переменная может быть использована.
- Глобальные переменные доступны везде, локальные — только внутри функции.
- Понимание области видимости помогает избегать ошибок и конфликтов имён.
- Следует минимизировать использование глобальных переменных

Итоги лекции:

- Функция — это именованный блок кода, выполняющий определённую задачу.
- Функции позволяют повторно использовать код и упрощают программу.
- Функции объявляются с помощью function имя() { ... } и вызываются по имени с круглыми скобками.
- Параметры — это переменные функции, аргументы — значения, передаваемые при вызове.
- Значения по умолчанию позволяют функции работать корректно даже при отсутствии аргумента.
- return используется для возврата значения функции; без него функция возвращает undefined.
- Функции могут иметь побочные эффекты, например вывод в консоль или изменение переменных.
- Область видимости (scope) определяет, где переменные доступны:
  - глобальная — переменные доступны везде;
  - локальная — переменные доступны только внутри функции.
- Понимание функций и scope помогает структурировать код и избегать ошибок.

Контрольные вопросы к лекции:

- Что такое функция и зачем она нужна?
- Как объявляется функция в JavaScript?
- Как вызвать функцию?
- В чём разница между параметрами и аргументами функции?
- Как задать значение параметра по умолчанию?
- Для чего используется оператор return?
- Чем функция с return отличается от функции с побочными эффектами?
- Что такое побочные эффекты функции? Приведите пример.
- Что такое область видимости переменных (scope)?
- Чем глобальные переменные отличаются от локальных?
- Как область видимости влияет на доступ к переменным внутри функции?

Домашнее задание:

https://ru.hexlet.io/courses/js-basics

хекслет колледж

@HEXLY.KZ