25.Необязательные параметры функций—

В программировании большое количество функций и методов имеют параметры, которые редко меняются. В таких случаях этим параметрам задают значения по умолчанию, которые можно поменять по необходимости. Этим немного сокращается количество одинакового кода. Пример:

*// Функция возведения в степень*

*// Второй параметр имеет значение по умолчанию 2*

**const** pow **=** (x, base **=** 2) **=>** {

**return** x **\*\*** base;

};

*// 3 во второй степени (двойка задана по умолчанию)*

pow(3); *// 9*

*// три в третьей степени*

pow(3, 3); *// 27*

Значение по умолчанию выглядит как обычное присваивание в определении. Оно срабатывает только в том случае, если аргумент не передан. К этому нужно привыкнуть.

Значение по умолчанию может быть даже в том случае, когда параметр один:

**const** print **=** (text **=** 'nothing') **=>** console.log(text);

print(); *// nothing*

print("Hexlet"); *// Hexlet*

Параметров со значениями по умолчанию может быть сколько угодно:

**const** f **=** (a **=** 5, b **=** 10, c **=** 100) **=>** { ... }

У значений по умолчанию есть одна особенность — они должны идти в самом конце списка параметров. Значения, передаваемые функции при вызове, всегда записываются в параметры слева направо, в том порядке, в котором они переданы. Получается, что если значения по умолчанию идут слева в списке параметров, перед обычными параметрами, то при вызове функции с какими-либо аргументами на место значений по умолчанию встанут значения аргументов. Например:

*// Вызовем эту функцию со следующими аргументами: f(1, 2, 3)*

**const** f **=** (a **=** 5, b **=** 10, c **=** 100, x) **=>** { ... }

*// параметры получат следующие значения: а = 1, b = 2, c = 3, x = undefined*

*// в x мы ничего не передали, и JS присвоил ему значение undefined*

*// Именно поэтому параметры со значением по умолчанию должны идти справа от обычных,*

*// иначе они либо будут перезаписаны, либо обычный параметр может остаться без значения*

*// Вызываем функцию, f(1, 2)*

**const** f **=** (a **=** 5, x, b **=** 10, c **=** 100) **=>** { ... }

*// a = 1, x = 2, остальные параметры получают значения по умолчанию*

*// Здесь все хорошо, никаких неожиданностей*

**const** f **=** (x, a **=** 5, b **=** 10, c **=** 100) **=>** { ... }

*// И здесь*

**const** f **=** (x, y, a **=** 5, b **=** 10, c **=** 100) **=>** { ... }

# 26. Упрощенный синтаксис функций—Основы JavaScript

По сравнению с некоторыми (в первую очередь функциональными) языками, определение функции в JavaScript выглядит довольно громоздко:

**const** square **=** (x) **=>** {

**return** x **\*\*** 2;

};

Здесь используется много дополнительных символов и слово return. С версии es6 в языке появился альтернативный, сокращенный синтаксис, который, в некоторых ситуациях, значительно упрощает восприятие и сокращает количество кода.

*// Требуется немного времени на привыкание к этой форме,*

*// но потом вы не сможете без нее жить*

**const** double **=** (x) **=>** x **\*\*** 2;

Отличия от полного определения два: пропали фигурные скобки и инструкция return. Сокращенная запись функции делает возврат автоматически. Подразумевается, что внутри такой функции ровно одно выражение, которое вычисляется, и его результат сразу возвращается наружу.

Подчеркнем, что отличия исключительно синтаксические, с точки зрения использования различий нет. Пример с двумя аргументами:

Полная версия:

**const** sum **=** (a, b) **=>** {

**return** a **+** b;

};

Сокращенная версия:

**const** sum **=** (a, b) **=>** a **+** b;

Обратите внимание на отсутствие фигурных скобок. Разработчики, которые не привыкли использовать такой синтаксис, иногда пишут подобный код const sum = (a, b) => { a + b };, а потом долго не могут понять, почему он не работает. Ответ очень простой: если стоят фигурные скобки, то это не сокращенная форма, а значит, чтобы функция вернула значение, придется поставить return.