Документация

JS JavaScript

При установке синтаксиса *JavaScript* область кода выражения должна содержать скрипт *JavaScript*.

Код *Выражения* является телом функции, которая должна возвращать значение. Для возврата этого значения используется `return`. Однако, если код представлен единственным выражением *JavaScript*, то применять `return` не обязательно.

> **Примечание:** Наименования полей/переменных/выражений чувствительны к регистру.

Следующий пример демонстрирует вычисление суммы двух полей *Калькулятора*:

```
// Вычисление суммы полей
let result;
result = COL1 + COL2;

return result;
```

В теле скрипта можно использовать ссылки на другие поля или переменные *Калькулятора*, в том числе новые, созданные в списке выражений. Через промежуточное выражение с типом Переменный можно передать массив. Ко входным Переменным можно обращаться через объект `this.Var.`.

> **Примечание:** В выражении возможно использовать ссылки на ранее вычисленные выражения, т.е. находящиеся выше в списке выражений. В связи с этим неверная позиция в списке может приводить к ошибке.

В данном примере для вычисления переменной `result` используются ссылки на поля с именами `COL1` и `COL2`.

Для вывода результата в вычисляемое поле калькулятора используется команда `return`, которая возвращает значение переменной `result`, содержащей вычисленную сумму полей *Калькулятора*.

Компактный вариант кода вычисления суммы полей калькулятора:

```
// Вычисление суммы полей
return COL1 + COL2;
```

Пример вычисления степени числа:

```
// Функция вычисления степени числа
function pow(x, n) {
  if (n != 1) {
    return x * pow(x, n - 1);
  } else {
    return x;
  }
}
// Вывод результата функции в поле калькулятора
return pow(COL1, COL2);
```

В данном скрипте определена функция `pow(x, n)`, принимающая в качестве аргументов число и степень, в которую оно возводится. Для вывода результата скрипта в вычисляемое поле калькулятора используется команда `return pow(COL1, COL2)`, где `pow(COL1, COL2)` — вызов объявленной функции с передачей параметрам `x` и `n` значений из полей (или переменных) калькулятора `COL1` и `COL2` соответственно.

В коде JavaScript можно использовать встроенные функции *Калькулятора*:

```
function my_concat(x, n) {
 let s;
 // s = x.concat(n);         - используется метод JavaScript
 s = Concat(x,n);        //   - используется функция калькулятора

return s;
}

return my_concat(COL1, COL2);
```

Наименования встроенных функций *Калькулятора* и JavaScript не пересекаются. Функции *Калькулятора* всегда начинаются с большой буквы, однако, их можно переопределить в коде JS. Явного импорта функций *Калькулятора* (как это делается в компоненте *JavaScript*) делать не нужно.

> **Примечание:** в отличии от компонента *JavaScript* в *Калькуляторе* не поддерживается объект Promise.

Импорт внешних модулей

Как и в компоненте JavaScript поддерживаются внешние модули, но в *Калькуляторе* возможно использование только модульной системы CommonJS. Модульная система ES6 (ECMAScript 6) не поддерживается. С примерами и документацией использования модулей CommonJS можно

ознакомится в статье JavaScript: Импорт внешних модулей.

Особенности применения внешних модулей

Для импорта модуля CommonJS применяется функция `require`. Пример:

```
const foo = require("foo/foo.js");
```

Инициализация объектов модуля происходит при первом таком вызове, при последующих используется кэш, в котором сохраняется состояние объектов модуля предыдущего вызова. Поскольку в *Калькуляторе* для каждой из вычисляемых ячеек таблицы происходит выполнение скрипта, то содержащийся в нем вызов `require` модуля происходит многократно и используется кэш. Таким образом измененное состояние объектов модуля может передаваться от вызова к вызову. Это необходимо учитывать и при необходимости очищать кэш, доступный через объект `require.cache`:

```
let path = require.resolve("foo/foo.js");
delete require.cache[path];  // Очищается кэш модуля "foo/foo.js"
```

> **Важно**: использование кэша для передачи значений между вызовами `require` может привести к непредсказуемым результатам вычислений в виду использования многопоточности вычислений.

Панель быстрого доступа

На панели расположена кнопка вызова окна **Предпросмотр**.

Предпросмотр — позволяет оценить корректность расчетов, отображая до 100 первых строк результирующей таблицы. Горячая клавиша вызова — `F3`.

В нижней части окна *Предпросмотр* расположена **Консоль**, в которой отображаются сообщения об ошибках при вычислении выражений и результат выполнения кода `console`. Она работает аналогично консоли отладки кода в компоненте JavaScript.

Для отображения/сворачивания *Консоли* используются кнопки ^ и v.

Помимо кнопки, открывающей *Предпросмотр*, на панели быстрого доступа есть кнопки, по нажатию на которые в область кода выражения вставляется заготовка или шаблон.

Логические операции

- **&&** — Логическое "И"
- || — Логическое "ИЛИ"
- **!** — Логическое "НЕ"

Операторы сравнения

- **=** — Равно
- **!=** — Не равно
- **<** — Меньше
- **>** — Больше
- **<=** — Меньше или равно
- **>=** — Больше или равно

Шаблоны

- 9.0 — для ввода вещественного числа, будет вставлено `0.0`
- “” — для ввода строки, будет вставлено `""`
- 31 — для ввода даты, будет вставлена конструкция создания объекта, содержащего текущую дату. Пример: `new Date(2020, 1, 5)`
- 🕓 — для ввода даты/времени, будет вставлена конструкция создания объекта, содержащего текущее дата/время. Пример: `new Date(2020, 1, 5, 13, 12, 50, 100)`

Логические значения

- **false** — Ложь
- **true** — Истина