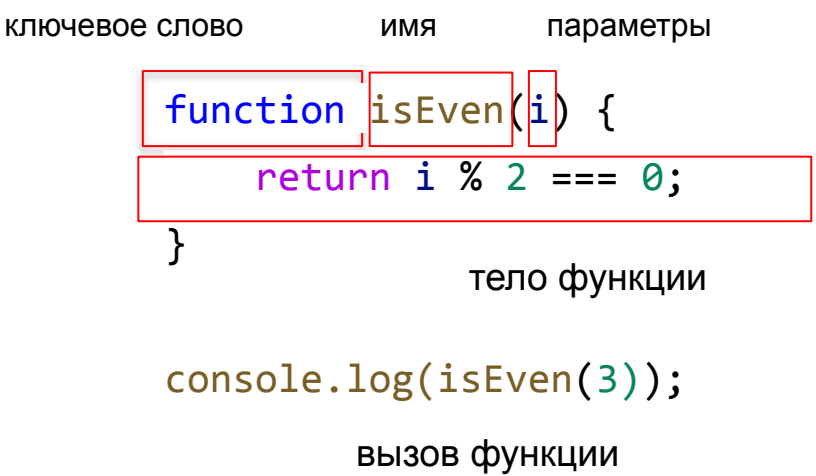
Алгоритмы обработки данных на JavaScript

Процедурный подход программирования. Определение функции. Передача параметров в функции. Локальные, глобальные переменные. Рекурсия

Функция - блок программного кода, который определяется один раз и может вызываться многократно

Определение функции. Function declaration

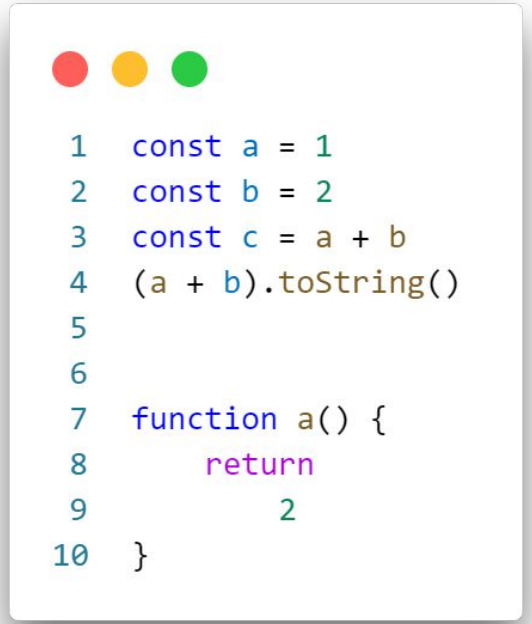


Инструкция return прекращает выполнение функции и возвращает значение указанного ей выражения.

Если в функции отсутствует инструкция return, то она возвращает undefined

return и ;

Google Javascript Style Guide



```
1  const a = 1
2  const b = 2
3  const c = a + b
4  (a + b).toString()
5
6
7  function a() {
8      return
9          2
10 }
```

Параметры функции

```
function printNameAndSurname(name, surname) {
    console.log(name);
    console.log(surname);
}

printNameAndSurname('name');
// name
// undefined

printNameAndSurname('name', 'surname', 'a', 'b', 'c')
// name
// surname
```

Функции в JS могут вызываться с произвольным числом аргументов

В случае если функция вызвана с меньшим числом параметров, чем указано в сигнатуре, то недостающие аргументы получают значение undefined

Если функция вызвана с большим числом параметров, чем указано в сигнатуре, то лишние аргументы будут проигнорированы

Параметры по умолчанию

значение по умолчанию

```
function printFullName(name, surname = '?') {
    console.log(`${name} ${surname}`);
}

printFullName('Vasya');
// Vasya ?
```

Если мы хотим сделать какой-то параметр функции необязательным, то мы можем указать ему значение по умолчанию

Локальные переменные

```
1  function inc(x) {
2      const y = x + 1;
3
4      return y;
5  }
6
7  inc(x);
8
9  console.log(y); // ReferenceError: y is not defined
```

Переменные, объявленные внутри функции, не видны за её пределами

Глобальные переменные

```
let counter = 0;

function count() {
    counter++;
}

console.log(counter); // 0
count();
console.log(counter); // 1
```

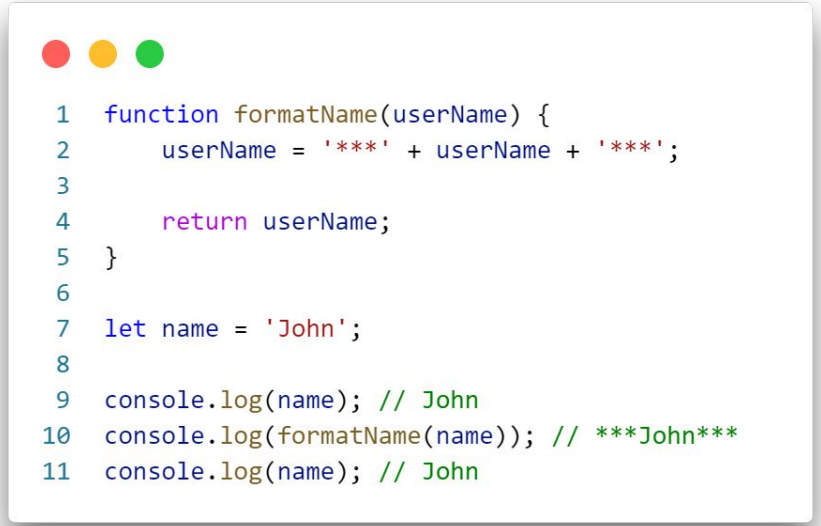
Функция обладает полным доступом к внешним переменным и способна изменять их значения

Всплытие функций

```
1  console.log(square(4)); // 16
2
3  function square(x) {
4      return x ** 2;
5  }
```

JS поднимает объявление функций в начало блока. Благодаря этому они могут вызываться раньше своего непосредственного объявления

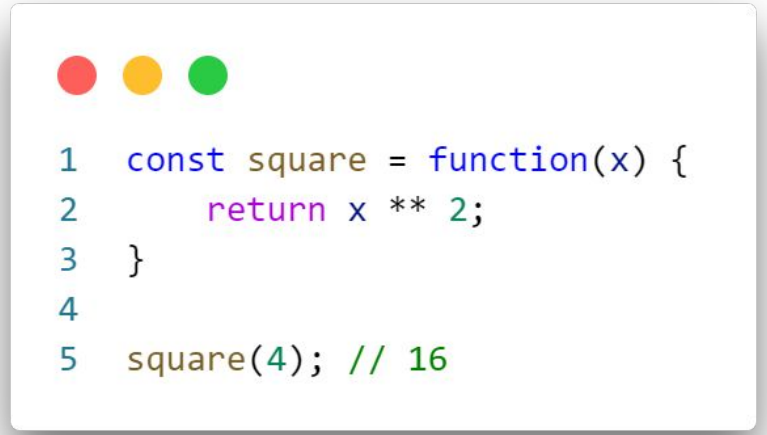
Передача параметров по значению



```
function formatName(userName) {
    userName = '***' + userName + '***';

    return userName;
}

let name = 'John';

console.log(name); // John
console.log(formatName(name)); // ***John***
console.log(name); // John
```

При передаче в функцию примитивных значений, функция получает копию этого значения

Function expression

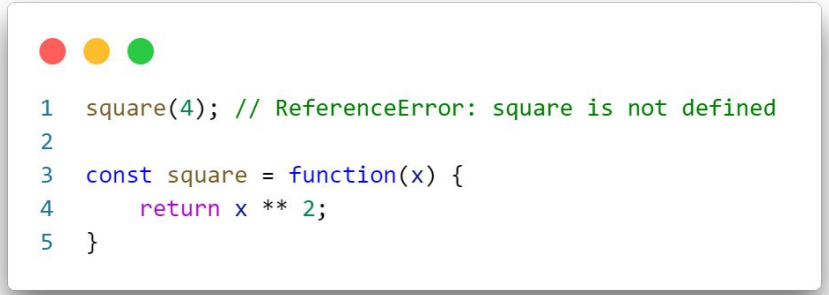


```
1  const square = function(x) {
2      return x ** 2;
3  }
4
5  square(4); // 16
```

В JS допускается способ создания функции в середине произвольного выражения с последующей записью функции в переменную

Данный способ объявления функции называется Function expression

Поднятие функциональных выражений



```
1  square(4); // ReferenceError: square is not defined
2
3  const square = function(x) {
4      return x ** 2;
5  }
```

Функциональные выражения не поднимаются в отличие от объявлений функций. Это означает, что их невозможно использовать до их непосредственного объявления

Function expression

```
function getGreetingFn(language) {
    switch (language) {
        case 'ru': {
            return function (name) {
                return 'Здравствуйте, ' + name;
            }
        }
        case 'de': {
            return function (name) {
                return 'Guten tag, ' + name;
            }
        }
        case 'en':
        default: {
            return function (name) { return 'Hello ' + name };
        }
    }
}


const language = 'en';
const grettingFn = getGreetingFn(language);

alert(grettingFn('Artem'));
```

Function expression стоит использовать, например, если функция объявляется в зависимости от внешнего условия

Arrow function

```
const fn = function (x, y, z) { return 123; }; === const fn = (x, y, z) => { return 123 };
```

Телом стрелочной функции может быть блок, как показано выше. Однако, телом может быть и выражение

```
const fn = (x, y, z) => 123;
```

Если стрелочная функция принимает только один аргумент, то скобки вокруг списка параметров могут быть опущены

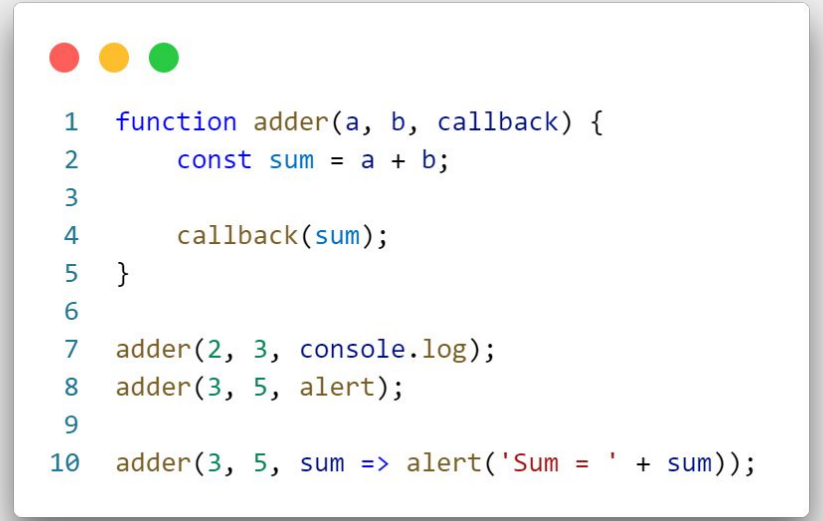
```
const id = x => x;
```

Если функция не зависит от аргументов, то круглые скобки будут пустыми, но обязательными

```
const random = () => Math.random();
```

Callback function

Колбэк-функция (или обратный вызов) - это функция, переданная в другую функцию в качестве аргумента, которая затем вызывается по завершению какого-либо действия.



```
function adder(a, b, callback) {
    const sum = a + b;

    callback(sum);
}

adder(2, 3, console.log);
adder(3, 5, alert);

adder(3, 5, sum => alert('Sum = ' + sum));
```

Рекурсия

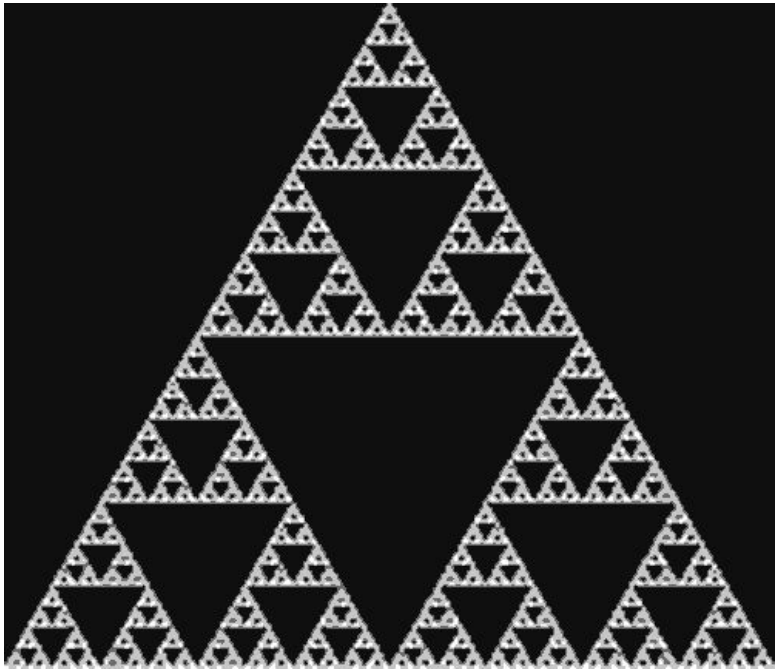
Рекурсия – есть метод определения множества объектов или процесса в терминах самого себя

Рекурсивное определение состоит из:

- базисная часть - задает определение для некоторой фиксированной группы объектов
- рекурсивная часть - приводит утверждение из рекурсивной к базисной части

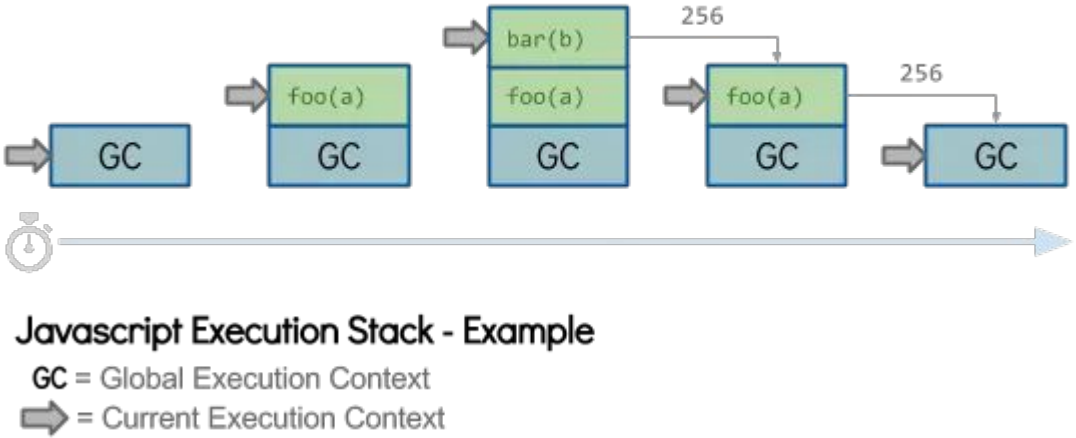
Рекурсия - способ решения задач с помощью подзадач, постановка которых аналогична исходной задаче.

Треугольник Серпинского

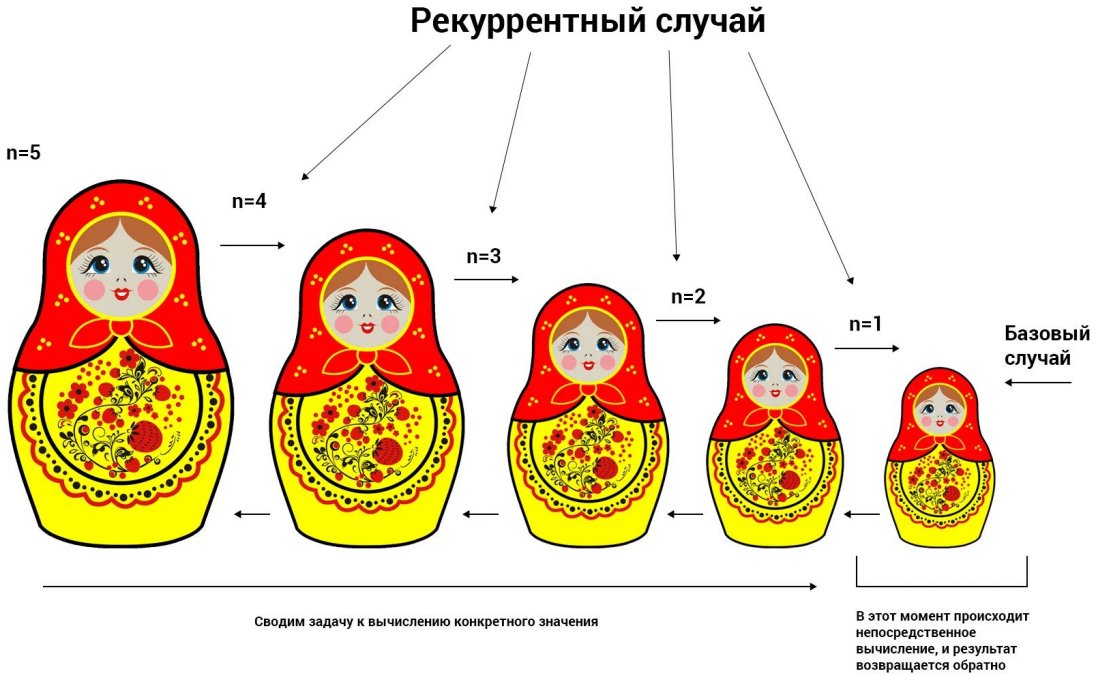


Стек выполнения функций

```
const a = 4;

function foo(x) {
  const b = a * 4;

  function bar(y) {
    const c = y * b;

    return c;
  }

  return bar(b);
}

console.log(foo(a));
```



Матрёшки



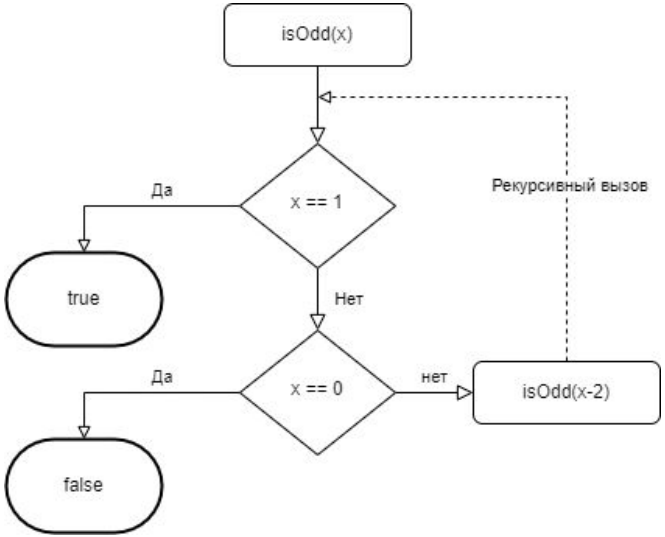

```
function makeMatryoshka(n) {
    if (n === 1) {
        return console.log('Неделимая матрёшка');
    } else {
        console.log('Верхняя половина матрёшки', n);
        makeMatryoshka(n - 1);
        console.log('Нижняя половина матрёшки', n);
    }
};

makeMatryoshka(6);
```

Алгоритм проверки чётности числа

- Базисное утверждение - число 1 является нечётным целым числом, число 0 является чётным целом числом
- Рекурсивное утверждение - если число N является нечётным целым числом, то число N-2 является нечётным целым

Является ли число 7 нечётным?

- Число 7 нечётно, если число 5 нечётно
- Число 5 нечётно, если число 3 нечётно
- Число 3 нечётно, если число 1 нечётно
- Число 1 нечётно



Алгоритм проверки чётности числа

При рекурсивном запуске функции информация о текущем вызове сохраняется в стеке контекстов выполнения (Call stack)

При рекурсивном возврате контекст выполнения функции получается из стека, и функция продолжает своё выполнение

Алгоритм проверки чётности числа

Глубина рекурсии составила 3

```
1  function isOdd(x) {
2      if (x === 0) {
3          return false;
4      }
5
6      if (x === 1) {
7          return true;
8      }
9
10     return isOdd(x - 2);
11 }
```

CALL STACK
{ x: 1, line: 1 } return true
{ x: 3, line: 10 }
{ x: 5, line: 10 }
{ x: 7, line: 10 }

Алгоритм проверки чётности числа

На практике бывает возможно найти итеративный алгоритм, решающий ту же задачу, но с меньшими затратами по памяти.

```
1  function isOdd(x) {
2      while (x > 1) {
3          x -= 2;
4      }
5
6      return x === 1;
7  }
```

```
1  function isOdd(x) {
2      return x % 2 === 1;
3  }
```

Факториал числа n

Произведение натуральных чисел от 1 до n;

5! = 5*4*3*2*1;

4! = 4*3*2*1;

3! = 3*2*1;

2! = 2*1;

1! = 1;

```
factorial(5) =
5 * factorial(4) =
5 * 4 * factorial(3) =
5 * 4 * 3 * 2 * factorial(1) =
5 * 4 * 3 * 2 * 1 = 120
```

```
let getFactorial = (num) => {

  if (num == 1)
    return 1;

  else
    return num * getFactorial(num-1);
}
```

return

```
let getFact = (num) => {

  if (num == 1)
    return 1;

  else
    return num * getFact(num-1);
}
```

return