



# Лекция 2

Циклы и программная логика

## Цели на сегодня

- Поговорить об алгоритмах;
- Узнать о базовых способах взаимодействия с пользователем;
- Глянуть на циклы, понятие инкремента / декремента;
- Научиться использовать программную логику.

## **Алгоритм –**

набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата.

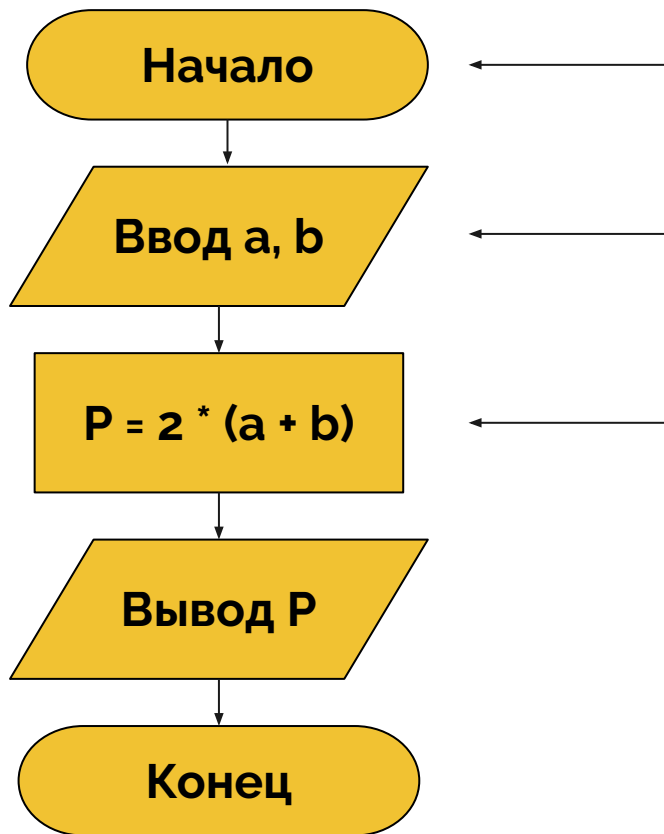
## **Примеры алгоритмов:**

- синтез вещества;
- готовка по рецепту;
- сортировка предметов;
- перемещение по навигатору.

## Основные виды описания алгоритмов:

- **блок-схемы** — представление алгоритма в виде геометрических фигур;
- **псевдокод** — использование основных конструкций JS для описания алгоритма, однако без подробностей.

# Линейный алгоритм

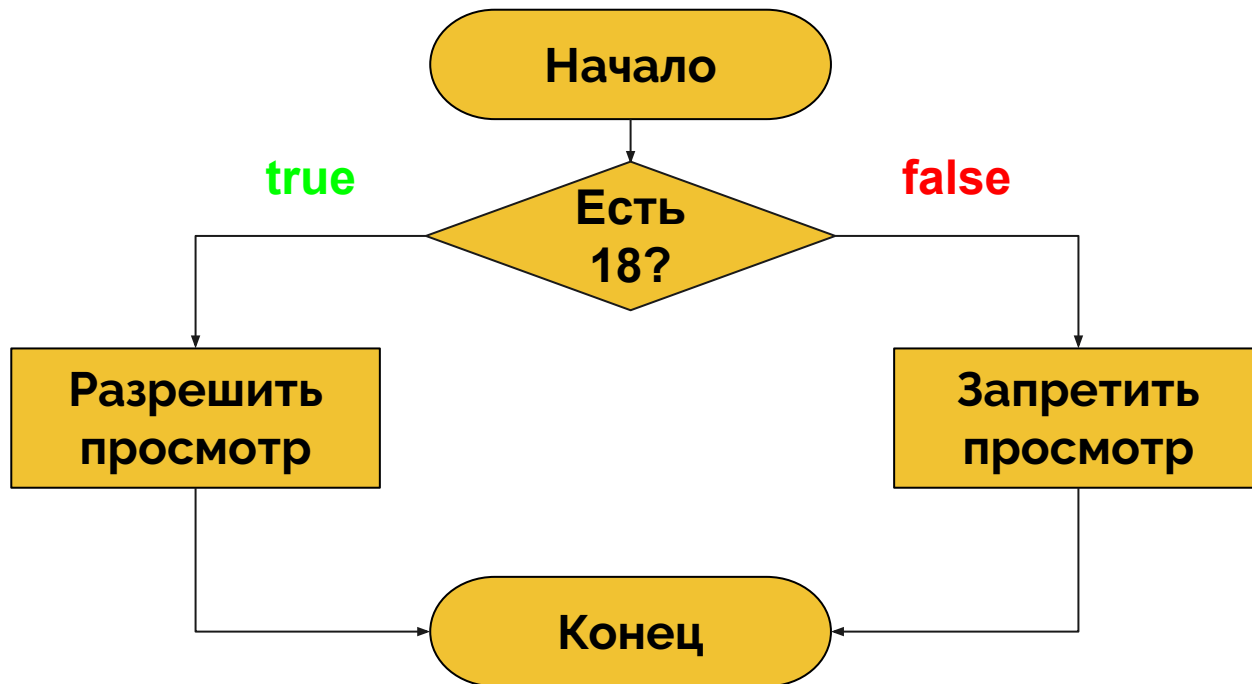


← Обычно принято указывать  
начало и конец программы

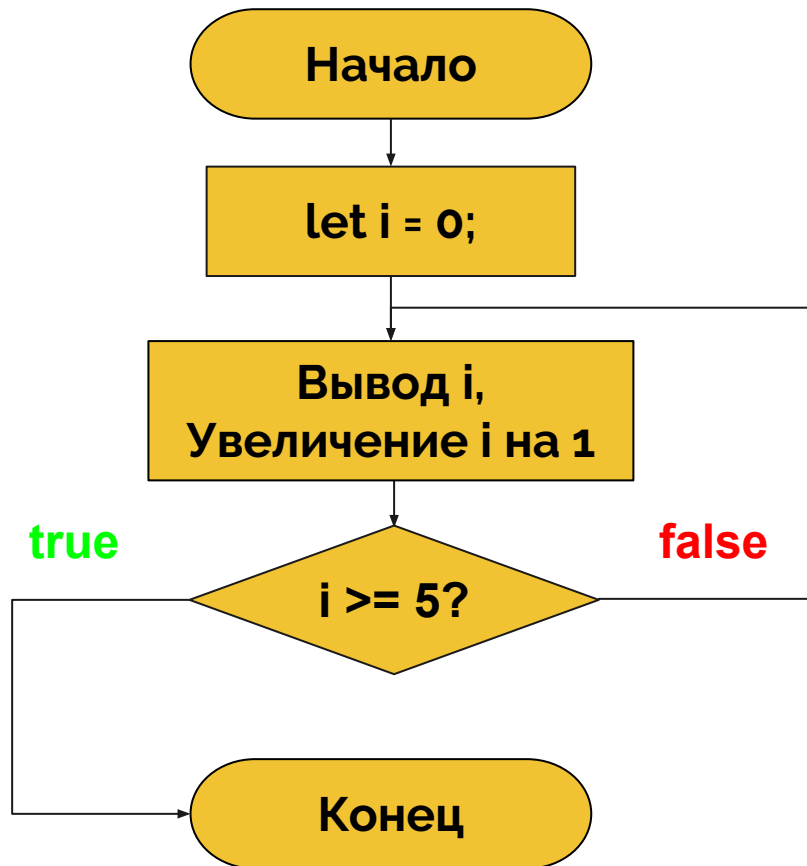
← Параллелограмм указывает  
на ввод и вывод данных

← В прямоугольниках  
записывают операции

## Разветвленный (условный) алгоритм



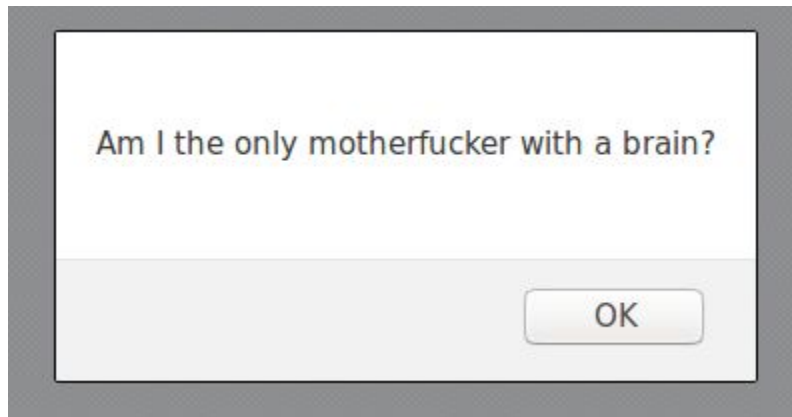
# Циклический алгоритм



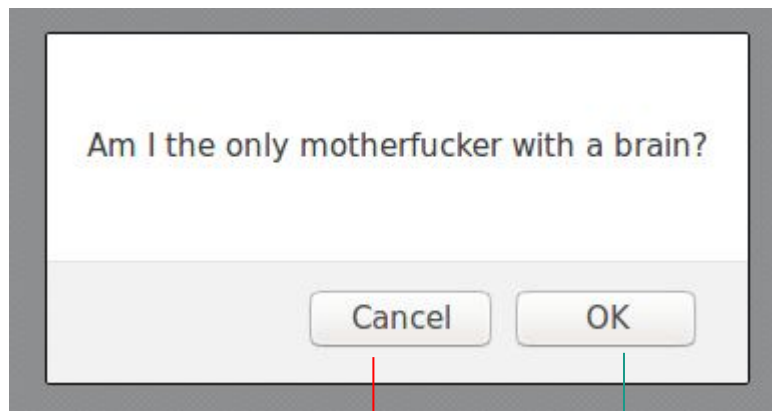


**Взаимодействие с пользователем**

```
alert('Am I the only motherfucker with a brain?');
```

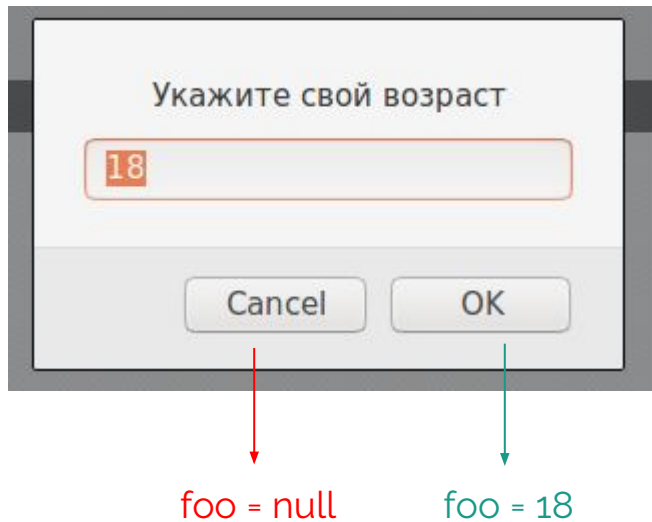


```
const question = confirm('Am I the only motherfucker  
with a brain?');
```



question = false    question = true

```
const foo = prompt('Укажите свой возраст', '18');
```



# Циклы (Loops)

## Зачем нужны циклы?

При написании скриптов зачастую встает задача сделать однотипное действие много раз.

Например: вывести товары из списка один за другим, вывести записи блога. Или просто перебрать все числа от 1 до 10 и для каждого выполнить одинаковый код.

Для многократного повторения одного участка кода предусмотрены циклы.



// Инкремент используется для увеличения числа на 1

// (0) Постфиксная форма

```
let i = 0;  
console.log(i++); // 0  
console.log(i); // 1
```

// (1) Префиксная форма

```
let j = 5;  
console.log(++j); // 6
```



// Декремент используется для уменьшения числа на 1

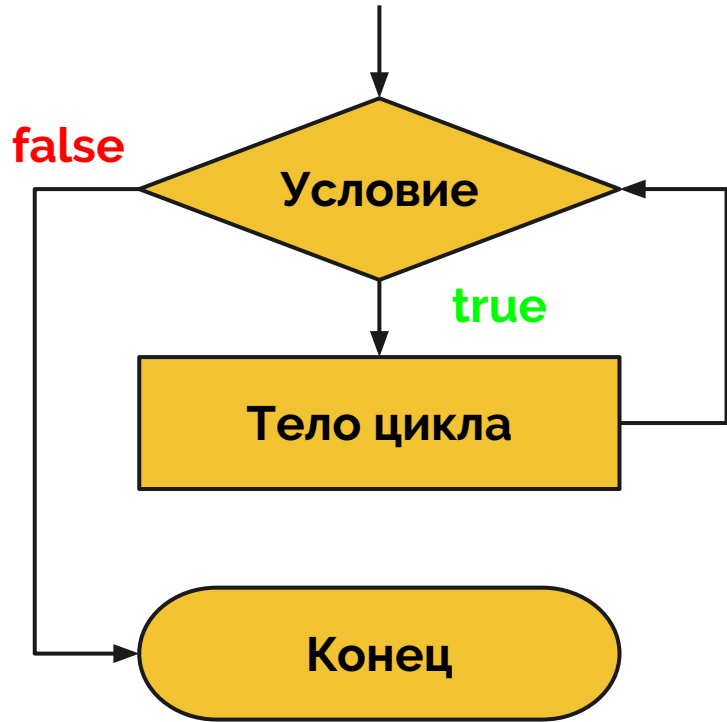
// (0) Постфиксная форма

```
let i = 10;  
console.log(i--); // 10  
console.log(i); // 9
```

// (1) Префиксная форма

```
let j = 10;  
console.log(--j); // 9
```





Первый рассматриваемый цикл – **while**. Он проверяет условие, и если оно приравнивается к **true**, то тело цикла выполняется.

Повторение цикла называется «итерация». Цикл в примерах ниже совершает 10 итераций.



// Цикл while выглядит таким образом

```
while (условие) {  
    // код, тело цикла  
}
```

```
let counter = 10;
```

```
while (counter >= 10) {  
    console.log(`The number is ${counter}`);  
    counter++;  
}
```

The number is 0

The number is 1

The number is 2

The number is 3

The number is 4

The number is 5

The number is 6

The number is 7

The number is 8

The number is 9

The number is 10

← 10



```
/*  
    Значение counter можно как увеличивать,  
    так и уменьшать  
*/  
  
let counter = 10;  
  
while (counter >= 0) {  
    console.log(`The number is ${counter}`);  
    counter--;  
}
```

```
The number is 10  
The number is 9  
The number is 8  
The number is 7  
The number is 6  
The number is 5  
The number is 4  
The number is 3  
The number is 2  
The number is 1  
The number is 0
```

← 0

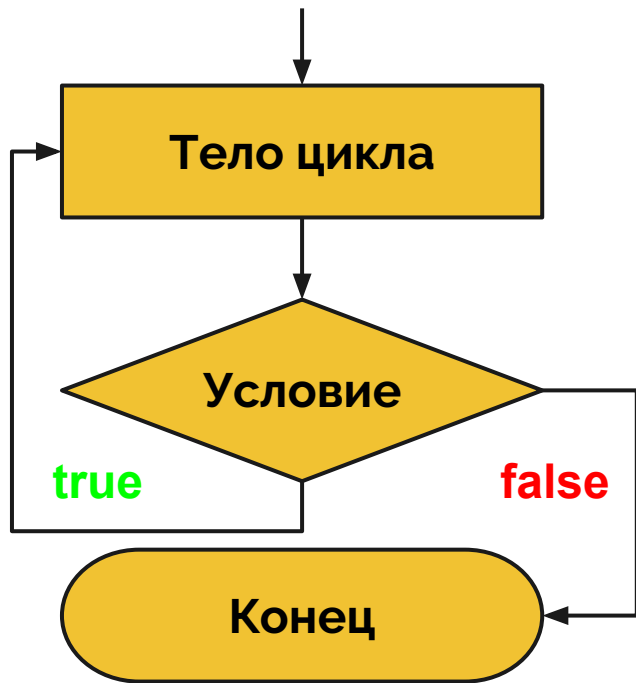


```
/*  
    Можно упростить условие.  
    Чем проще условие, тем быстрее будет выполняться  
    цикл.  
*/  
  
let counter = 10;  
  
while (counter--) {  
    console.log(`The number is ${counter}`);  
}
```

```
The number is 9  
The number is 8  
The number is 7  
The number is 6  
The number is 5  
The number is 4  
The number is 3  
The number is 2  
The number is 1  
The number is 0
```

```
← undefined
```

Если можно двигаться к нулю, то лучше  
двигаться к нулю.



В цикле **do...while** сначала выполняется тело, а уже потом проверяется соответствие условию.

Циклы **do...while** используются достаточно редко, поскольку считается, что они сложнее воспринимаются визуально.



```
// Следующим типом циклов является do...while
```

```
// Синтаксис:
```

```
do {  
    выражение  
} while (условие);
```

```
// Пример:
```

```
let num = 0;
```

```
do {  
    console.log(`There are ${num} students here`);  
    num++;  
} while (num <= 7);
```

```
There are 0 students here  
There are 1 students here  
There are 2 students here  
There are 3 students here  
There are 4 students here  
There are 5 students here  
There are 6 students here  
There are 7 students here
```

```
← 7
```

● ● ●

```
// Чаще всего используется цикл for
```

```
// Синтаксис:
```

```
for (начало; условие; шаг) {  
    // ... тело цикла ...  
}
```

```
// Пример:
```

```
for (let i = 0; i <= 10; ++i) {  
    console.log(i);  
}
```

0

1

2

3

4

5

6

7


8

9

10

← undefined





```
// В цикле for можно пропускать все части цикла:
```

```
// (1)
```

```
let i = 0;
```

```
for (;i < 3; i++) {  
    console.log(i); // 0, 1, 2  
}
```

```
// (2)
```

```
let j = 5;
```

```
for (;j > 0;) {  
    j--;  
    console.log(j); // 4, 3, 2, 1, 0  
}
```

**Бесконечный цикл (Infinite loop)**



// Бесконечный цикл, который что-то делает

```
let i = 0;
```

```
while (true) {  
    console.log(i);  
    i++;  
}
```

// Бесконечный цикл, который ничего не делает

```
for(;;);
```



```
let sum = 0;

while (true) {
    let value = +prompt('Введите число', '');

    if (!value) break;

    sum += value;
}

alert(`Сумма: ${sum}.`);
```

## Самостоятельная работа

Реализовать программу, которая возвращает в консоль таблицу умножения на 2 (с использованием цикла **while**) и на 3 (с использованием цикла **for**) в следующем формате:

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
← undefined
```

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
← undefined
```

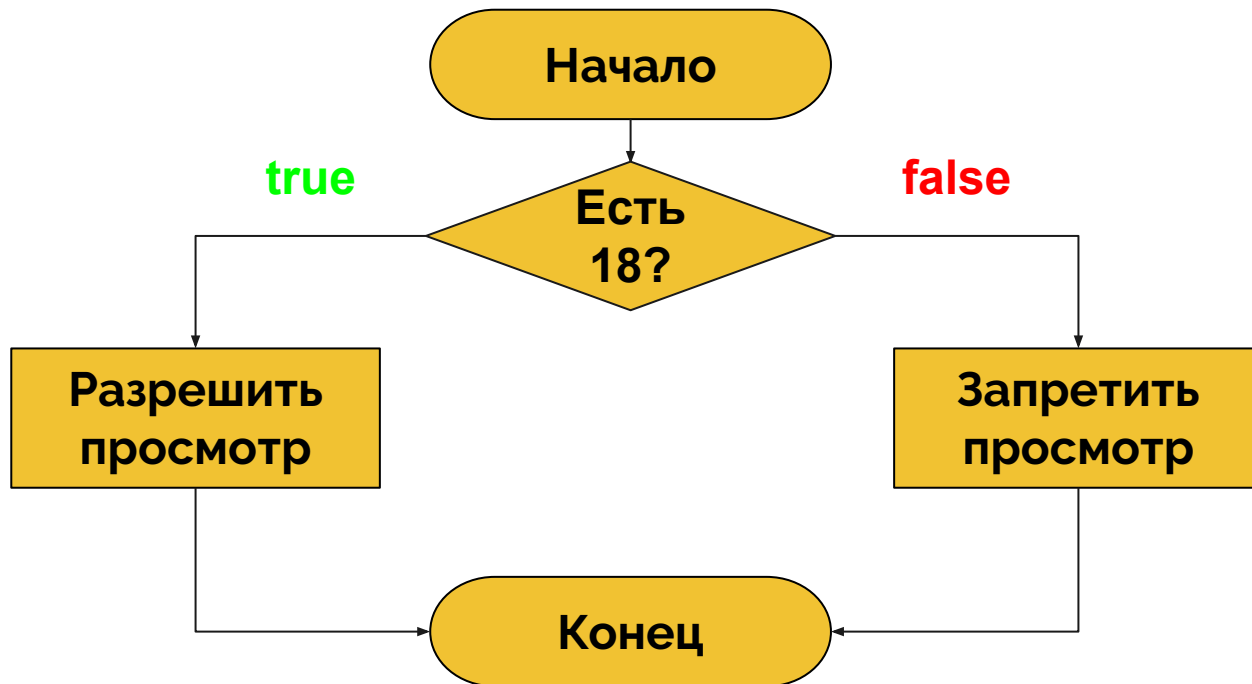
# Условная логика

## Зачем нужны условные операторы?

Если в зависимости от условия, необходимо выполнить различные действия, то используют условные операторы.

| Преобразование типов для примитивов |                   |
|-------------------------------------|-------------------|
| Приводится к false                  | Приводится к true |
| Число 0, NaN                        | Все остальное     |
| <code>null</code>                   |                   |
| <code>undefined</code>              |                   |
| Пустая строка ("")                  |                   |

## Разветвленный (условный) алгоритм







```
const age = +prompt( 'Сколько вам лет', '16' );
```

```
if ( !age ) {  
    alert( 'Данные не получены' );  
} else if ( age < 18 ) {  
    alert( 'Доступ запрещен' );  
} else if ( age >= 18 ) {  
    alert( 'Доступ разрешен' );  
} else {  
    alert( 'Системная ошибка' );  
}
```



// Синтаксис

условие ? выполнится\_если\_true : выполнится\_если\_false;

// Пример

```
const age = +prompt('Сколько вам лет', '16');  
(age < 18) ? alert('Доступ запрещен') : alert('Доступ разрешен');
```



```
/*
```

Конструкция switch используется для перебора значений и представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

```
*/
```

```
const city = 'Львов';
```

```
switch (city) {
```

```
  case 'Киев':
```

```
    console.log('Вы находитесь в Киеве');
```

```
    break;
```

```
  case 'Днепр':
```

```
    console.log('Вы находитесь в Днепре');
```

```
    break;
```

```
  default:
```

```
    console.log('Не удалось определить ваше местоположение');
```

```
}
```

## Самостоятельная работа

Реализовать программу, которая принимает название цвета и возвращает его CSS HEX код с использованием 1) if...else; 2) switch.

**Например:** мы ввели в окно **prompt** слово "Красный", а в новом окне **alert** появилось "#ff0000". Указать можно произвольное количество цветов, но **не меньше трех**.