

1. Двоичная логика
2. Логические операторы
3. Операторы if, else, else if
4. Тернарный оператор
5. Оператор switch

1 - Двоичная логика

Любая программа может быть создана на основе 3-х принципов.

Последовательность выполнения — это то же, что и поток выполнения в HTML, следование записи кода сверху вниз. Но не всегда программа должна выполняться строго последовательно. Для изменения последовательности используются циклы и ветвления.

Циклам посвящен следующий модуль. В это модуле мы рассмотрим ветвления.

Ветвление - возможность выполнить ту или иную последовательность кода в зависимости от условия. Условие может быть любым, но результат его проверки всегда имеет два значения - `true` или `false`.

Компьютер использует бинарный (от латинского *bis* - дважды) код. Т.е. всего два значения: `0` и `1` используются для создания любых программ. Это значит, что условия также задаются в виде `0` (нет, `false`) и `1` (да, `true`).

В математике существует раздел булевой логики (в честь [George Boole](#)), в котором условия бинарны, т.е. могут быть представлены в виде `0` и `1`, а также в виде слов `true` и `false`. Именно эта логика используется для реализации ветвления.

1.1 - Приведение типов

Для того чтобы работала булевая логика необходимо на входе иметь два значения. Поэтому в JavaScript осуществляется приведение типов. Компилятор JS приводит к `true` или `false` любые примитивные типы языка, а также приводит к `true` любой не примитивный тип.

Truthy и **Falsy** – термины, которые используются для тех значений, которые, в логической операции, приводятся к `true` или `false`, хотя изначально не были булями.

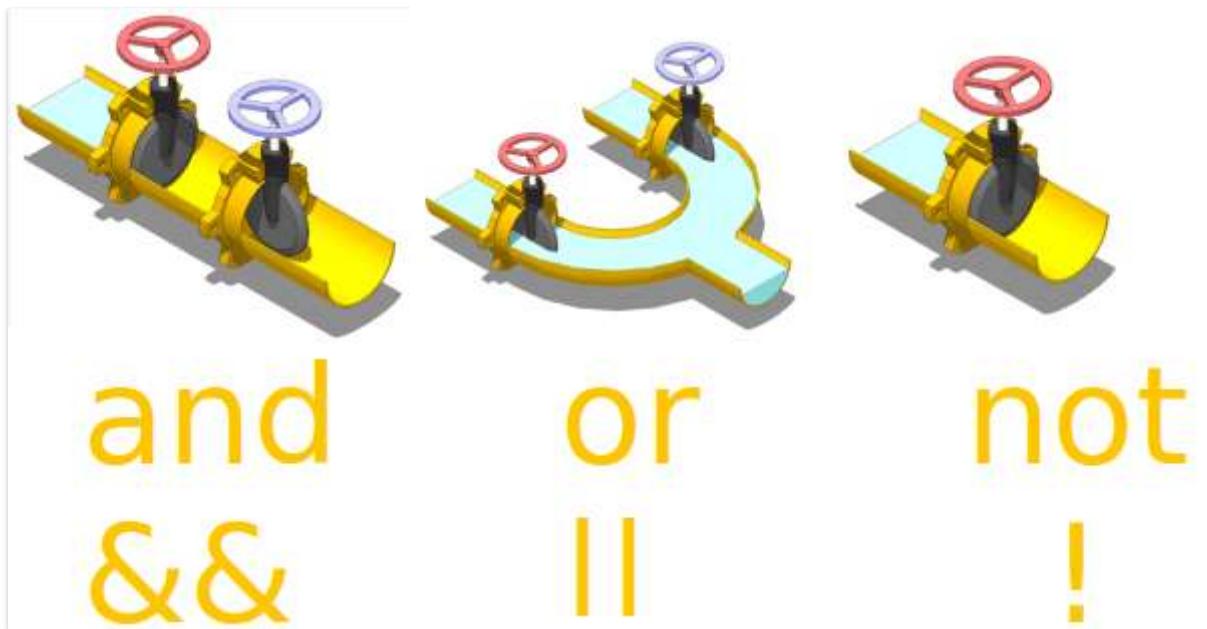
Запомните 7 ложных (**falsy**) значений, приводящихся к `false` :

`0, -0, NaN, null, undefined, пустая строка: "" или '', false`

Ложные значения приводятся к `false`. Все остальное приводится к `true`. Это истинные (**truthy**) значения . Приведение происходит в том случае, если компилятор обнаружит логический оператор.

2 - Логические операторы

Есть три логических оператора, которые используются для проверки выполнения множественных выражений.



2.1 - Логическое "И"

`&&` – приводит все операнды к булю и возвращает одно из значений (операндов). Левый operand если его можно привести к `false`, и правый в остальных случаях.

```
const num = 20;  
const result = a < 30 && a > 10;  
console.log(result); // true
```

В коде выше мы проверяем условие: переменная `a` меньше 30 И больше 10. Так как оба условия вернут `true`, то и результатом всего выражения будет `true`.

Для того чтобы оператор `&&` вернул `true`, требуется чтобы все operandы были истинными (`truthy`). Если хотябы один из operandов будет приведен к `false`, то результатом выражения будет этот operand.

```
const num = 20;  
const result = a > 30 && a > 10;  
console.log(result); // false
```

2.2 - Логическое "ИЛИ"

|| – возвращает одно из значений (операндов) - левый операнд если его можно привести к true, и правый в остальных случаях.

```
const num = 20;  
const result = a < 30 || a > 10;  
console.log(result); // true
```

Это тоже будет true так как хотябы один из операндов был приведен к true .

```
const num = 20;  
const result = a > 30 || a > 10;  
console.log(result); // true
```

2.3 - Логическое "НЕ"

! – возвращает false если операнд приводится к true, и true, если операнд приводится к false. То есть приводит операнд к булю, а затем заменяет его на противоположный.



```
console.log(!true); // false  
console.log(!false); // true
```

2.4 - Практика

Разберите код, проверьте себя, подумайте что приводится к `true`, а что к `false`, и почему именно такой результат логических операций.

```
// логическое И  
1 && 2 // 2
```

```
false && true // false
```

```
"" && "Hello" // ""
```

```
// логическое ИЛИ  
1 || 2 // 1
```

```
false || true // true
```

```
"" || "Hello" // "Hello"
```

```
// логическое НЕ  
!1 // false
```

```
!0 // true
```

При выполнении логических операций правый operand может не вычисляться:

- `false &&` (этот operand не вычисляется)

- `true ||` (этот операнд не вычисляется)

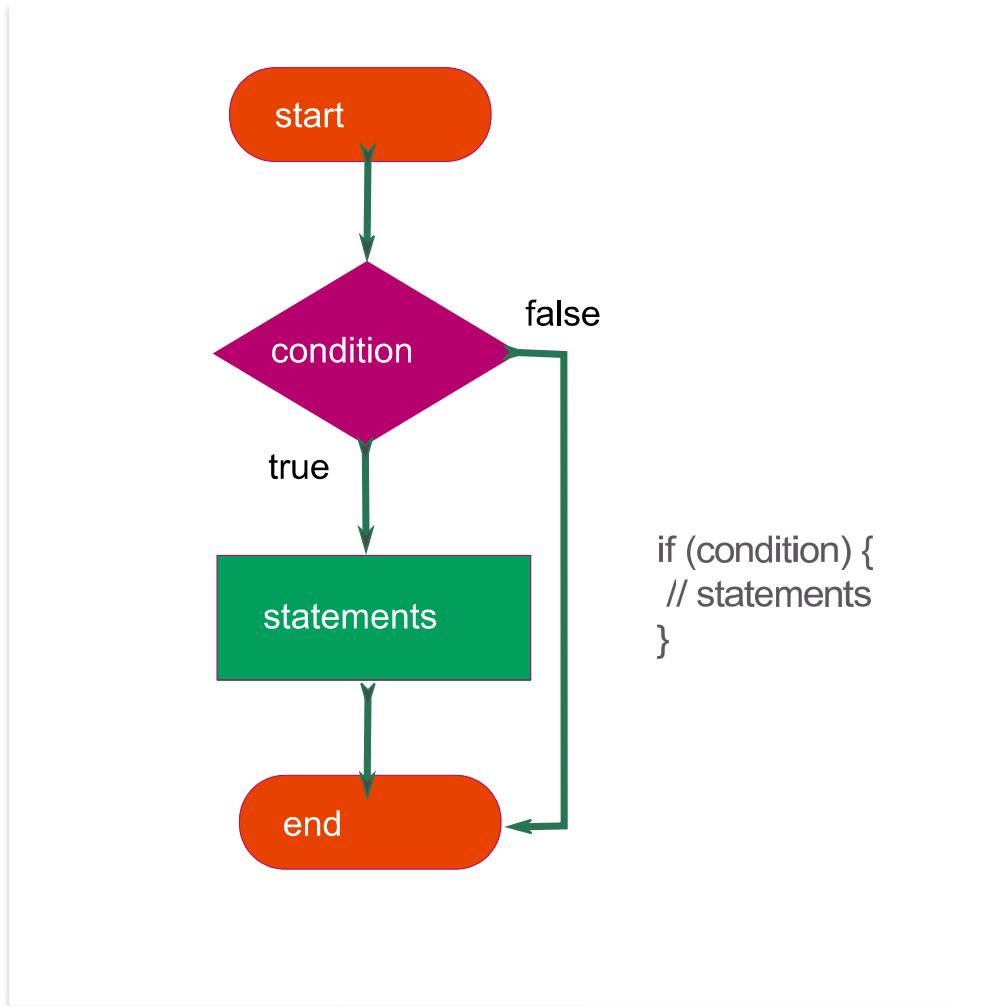


- "Купить билет на курорт **И** отдохнуть!". Если левый операнд "Купить билет на курорт" окажется `false`, то вычислять второй нет смысла.
 - "Отдохнуть **ИЛИ** сохранить деньги." Если левый операнд "Отдохнуть" окажется `true`, то вычислять второй нет смысла.
- [Видео - Logic Gates Explained](#)
 - [MDN: Логические операторы](#)

3 - Операторы if, else, else if

Логические операторы не могут самостоятельно управлять потоком выполнения программы. Для этого в JS используются **операторы ветвления**. Все они устроены по одному принципу - входные данные приводятся к булевому значению `true` или `false`, и в зависимости от результата этого значения, поток программы направляется в тот или иной программный блок.

3.1 - Синтаксис оператора if



Входные данные, которые приводятся к булевому типу называются **условием**. Условие помещают за оператором `if` в круглых скобках. Если условие приводится к `true`, то выполняется код в **фигурных скобках** (программный блок), а если условие приводится к `false`, код в фигурных скобках будет пропущен.

```

let season;
const monthName = "January";

if (monthName === "June") {
    season = "summer";
}
    
```

Синтаксис оператора `if` допускает запись без использования **фигурных скобок**. Компилятор сочтет блоком выполнения первую

найденную строку после круглых скобок. Пример подобного синтаксиса с блоком выполнения в виде строки `season = "summer";`.

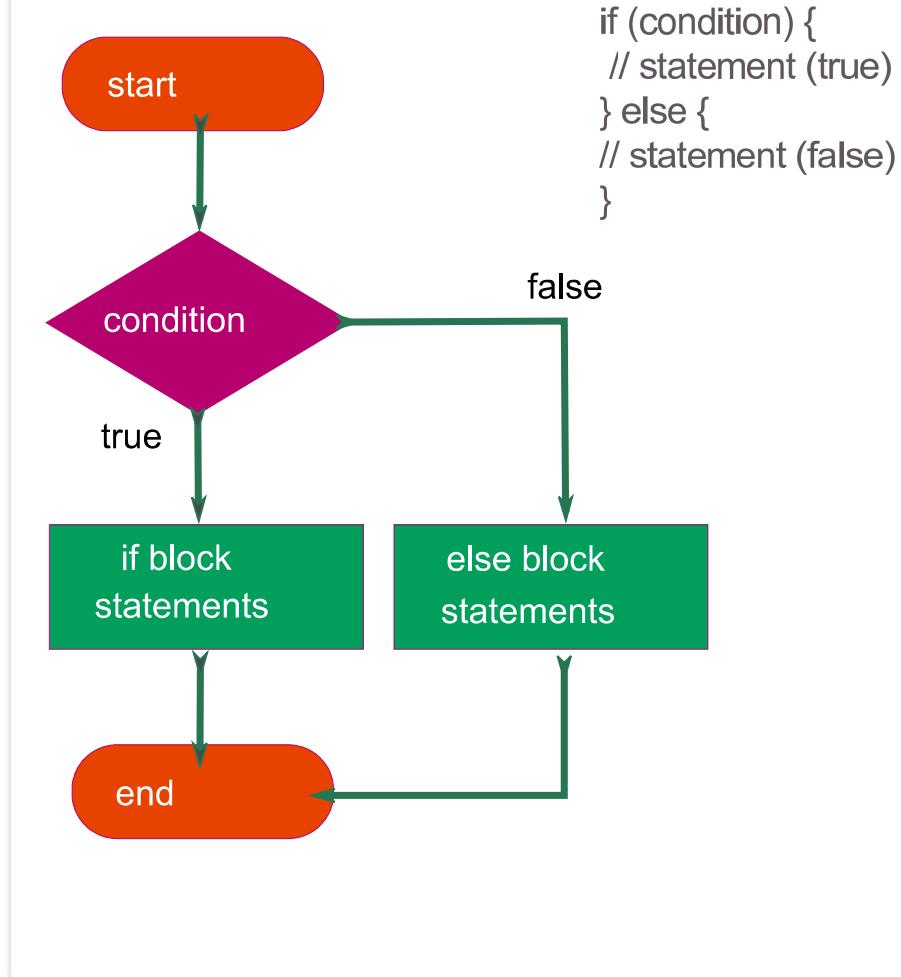
Подобная запись не всегда очевидна и ее следует избегать.

```
let season = "winter";
const monthName = "June";

if (monthName === "June")
    season = "summer";

console.log(season); // summer
```

3.2 - Синтаксис оператора if else



```

if (condition) {
  // statement (true)
} else {
  // statement (false)
}
  
```

Расширяет синтаксис оператора `if` тем, что в случае если условие приводится к `false`, то выполняется код в **фигурных скобках** после оператора `else`. При `true` оператор `else` и связанный с ним программный блок игнорируются.

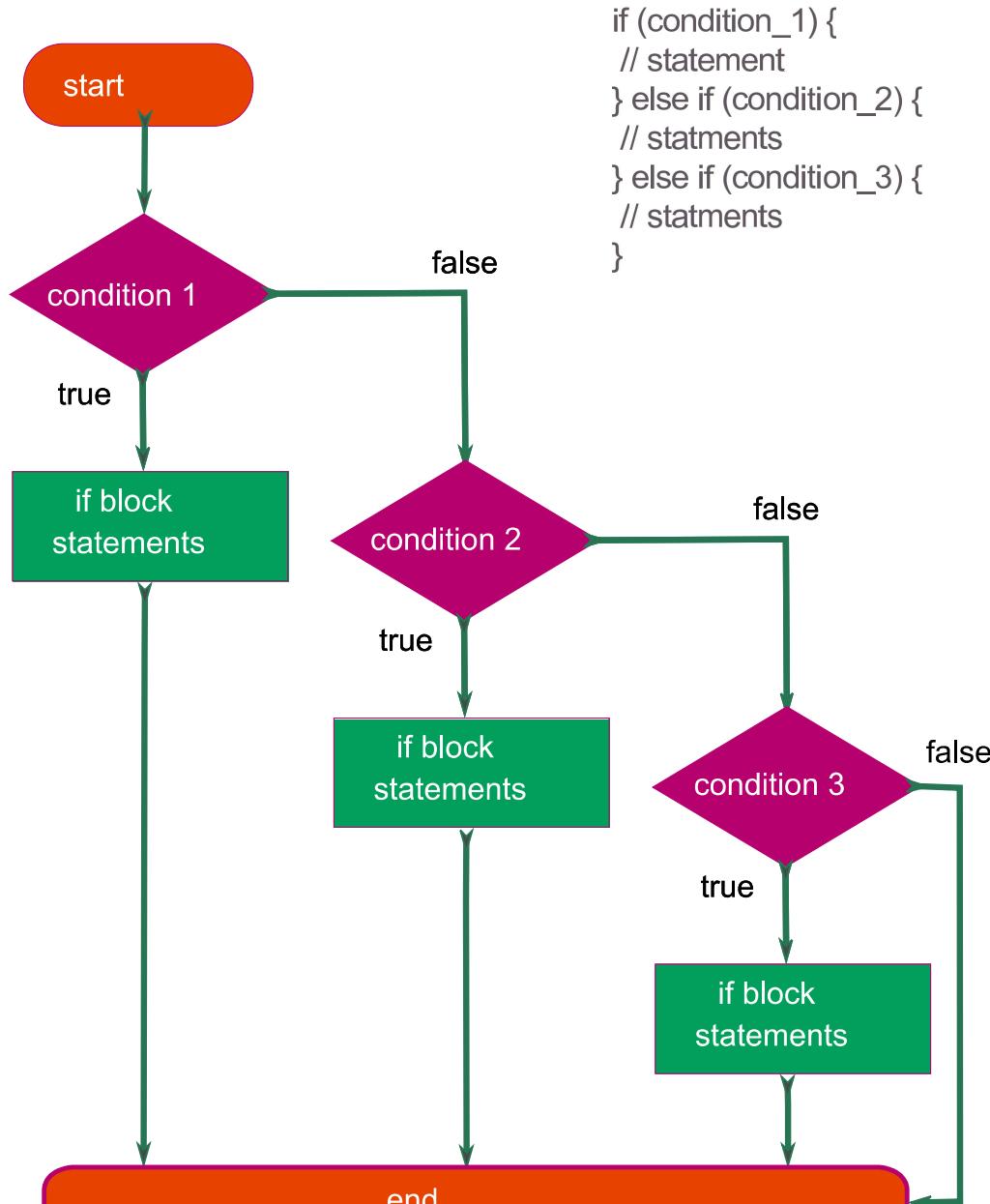
```

let season;
const monthName = "January";

if (monthName === "June") {
  season = "summer";
} else {
  season = "winter";
}
  
```



3.2.1 - Синтаксис оператора if else if



```

if (condition_1) {
  // statement
} else if (condition_2) {
  // statements
} else if (condition_3) {
  // statements
}
  
```

Расширяет синтаксис оператора `if else` тем, что после `else` снова добавляется оператор `if`. На первый взгляд код из множества подобных вложений кажется сложным. На самом деле все ответвления это результат `false` на все предыдущие вопросы. А при первом же `true` проверки прекратятся и выполнится только один сценарий, соответствующий этому `true`. Поэтому подобную запись следует читать как "ищу первое совпадение условия, игнорирую все оставшееся".

```
let season;
const monthName = "January";

if (monthName === "June") {
    season = "summer";
} else if (monthName === "May") {
    season = "spring";
} else if (monthName === "September") {
    season = "autumn";
} else {
    season = "winter";
}
```

4 - Тернарный оператор

Записывать конструкцию `if else` достаточно трудоёмко. В JS существует схожая конструкция с упрощенным синтаксисом называемая **тернарный** (ternary eng. - тройной) оператор.

Для выделения условия, блока `if` и блока `else` в тернарном операторе используют символы `?` и `:`, а символ `;` как и всегда, объявляет об окончании инструкции.

```
const monthName = "January";
let result;

if (monthName === "June") {
    let result = "summer";
} else {
    let result = "winter";
}
```

```
// Конструкция выше записанная тернарным оператором  
const season = monthName === "June" ? "summer" : "winter";
```

Синтаксис тернарного оператора позволяет осуществлять запись аналогичную синтаксису `if else if`. Посмотрите тернарный аналог записи синтаксиса `if else if` в примере ниже. Такой синтаксис читается с трудом, поэтому использовать тернарный оператор для таких задач крайне не рекомендуется.

```
const monthName = "January";  
  
const season = monthName === "June"  
    ? "summer" : monthName === "May"  
    ? "spring" : monthName === "September"  
    ? "autumn" : "winter";  
  
season; // "winter"
```

Результат тернарного оператора должен либо присваиваться через оператор `=` либо возвращаться через `return`. Использовать тернарный оператор без присваивания или возврата результата не является лучшей практикой.

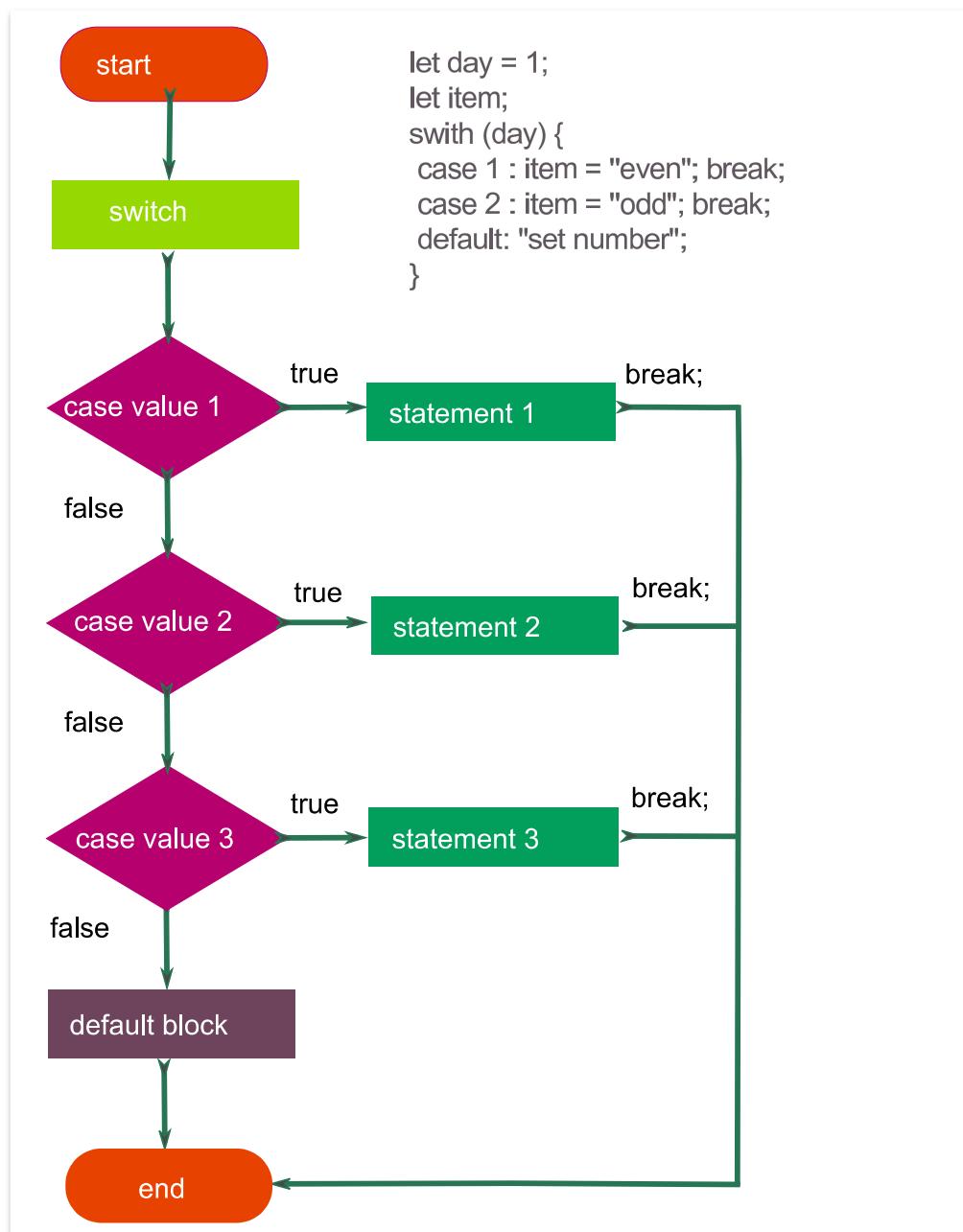
5 - Оператор switch

В некоторых случаях сложность чтения логических конструкций избежать, используя оператор ветвления `switch`. Синтаксис этого оператора разбивает условие на общую часть `switch` и множество

отдельных частей `case`. Т.е. применимость этого оператора ограничена только задачами с одним общим вопросом и множеством вариантов ответов.

Блок выполнения следует сразу за `:` и заканчивается как обычно `;`.

Оператор `break` в завершении каждого блока `case` ставят чтобы прервать дальнейшие проверки и сразу перейти к коду за оператором `switch`.



```

const monthName = "January";
let season;
  
```

```
switch (monthName) {  
    case "June":  
        season = "summer";  
        break;  
    case "May":  
        season = "spring";  
        break;  
    case "September":  
        season = "autumn";  
        break;  
    default:  
        season = "winter";  
}  
  
console.log(season); //winter
```

Если оператор **break** будет отсутствовать, то после того, как выполнится условие **case** все последующие блоки кода будут выполняться один за другим.

```
const monthName = "May";  
let season;  
  
switch (monthName) {  
    case "June":  
        season += "Summer";  
    case "May":  
        season += "Spring";  
    case "September":  
        season += "Autumn";  
    default:  
        season += "Winter";  
}
```

```
console.log(season); // "undefinedSpringAutumnWinter"
```

