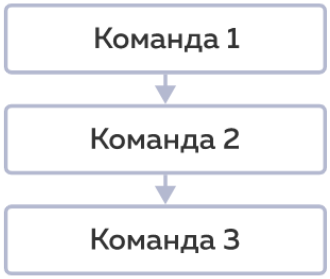
Лабораторная работа 12. Основные алгоритмические конструкции в JavaScript.

Цель работы. Познакомиться конструкциями ветвления и циклами в JavaScript, с их помощью изменить вёрстку на странице.

Результатом работы должны быть выполненные задания (12.1–12.6) /или (для студентов с полным доступом к тренажерам) пройденные тренажеры на htmlacademy.ru (https://htmlacademy.ru/courses/207, https://htmlacademy.ru/courses/209 , https://htmlacademy.ru/courses/211)

Линейные и нелинейные программы

Линейные программы всегда выполняют одни и те же команды.



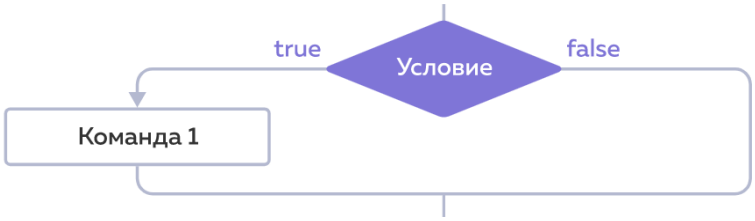
Нелинейная программа выполняет разные команды в зависимости от разных условий. Нелинейные программы ещё называют программами с ветвлением, а команды, которые выполняются в зависимости от условий, — ветками.

if и else

Чтобы программа проверяла условия и принимала решения на основе результатов проверок, используют оператор if:

```
if (условие) {
  действия;
}
```

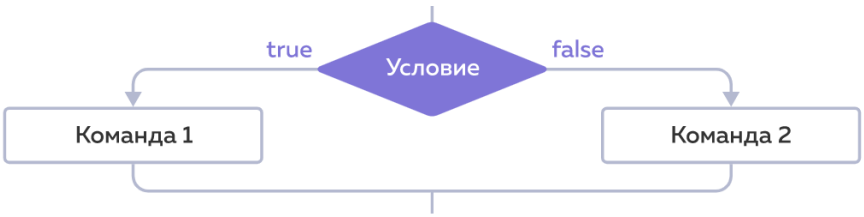
Здесь «условие» — это выражение, возвращающее `true` или `false`, а «действия» внутри фигурных скобок — это команды, которые выполняются, если условие удовлетворено. Удовлетворённым считается условие, которое возвращает `true`.



Чтобы создать ветку, которая будет срабатывать, если условие не выполнено, используем else:

```
if (условие) {
  действия;
} else {
  другие действия;
}
```

Ветка «действия» срабатывает, если условие выполнено. Ветка «другие действия» срабатывает, если условие не выполнено. Такие конструкции можно читать так: *если условие выполняется, сделай действие, иначе сделай другие действия.*

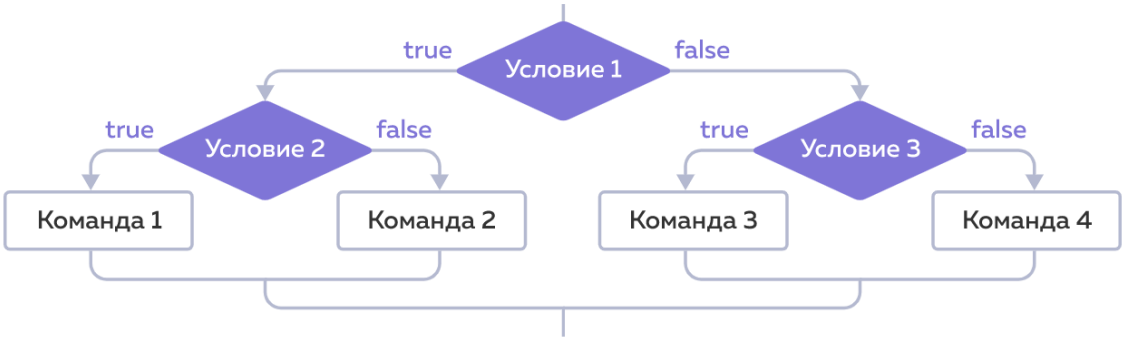


Вложенные условия

Что делать, если принятие решения зависит не от одного, а от двух и более условий? Можно использовать один if, а затем, внутри ветки, выполнить ещё одну проверку.

```
if (условие1) {
  if (условие2) {
    действия;
  }
}
```

Затем внутри вложенного условия можно добавить ещё одно и так до бесконечности.



Вложенные условия могут сделать код сложным и запутанным. Если вложенность большая, то понять, почему выполняется то или иное действие, становится трудно.

Операторы сравнения

Для сравнения значений используют операторы «больше» >, «меньше» <, «больше или равно» >= и «меньше или равно» <=.

```
console.log(1 > 1);  // Выведет: false
console.log(1 < 1);  // Выведет: false
console.log(1 >= 1); // Выведет: true
console.log(1 <= 1); // Выведет: true
```

Операторы сравнения работают не только с числами, но и с другими типами данных.

Для сравнения строк JavaScript использует таблицу кодирования Unicode. Порядок символов в ней совпадает с порядком букв в алфавите. Чем больше порядковый номер символа в таблице, тем больше символ. Обратите внимание, строчные буквы в таблице Unicode идут после заглавных, поэтому они считаются «больше»:

```
console.log('Б' > 'А'); // Выведет: true
console.log('а' > 'А'); // Выведет: true
```

Строки JavaScript сравнивает посимвольно. Если первый символ в первой строке больше первого символа во второй строке, то считается, что первая строка больше. Если первые символы совпадают, то сравниваются вторые символы и так далее. Если все символы совпадают, но одна из строк длиннее, то она и считается большей. Например:

```
console.log('Кот' > 'Код');         // Выведет: true
console.log('JavaScript' > 'Java'); // Выведет: true
```

Если сравниваются данные разных типов, то они приводятся к числу. При этом false становится нулём, а true — единицей.

```
console.log(2 > '1');    // Выведет: true
console.log(false <= 0); // Выведет: true
console.log(true >= 1);  // Выведет: true
```

Равенство и неравенство

В JavaScript можно также проверить значения на равенство и неравенство. При этом используют операторы `==`, `!=`, `===` и `!==`.

Оператор	Название	Действие
`==`	Нестрогое равенство (с приведением типов)	Сравнивает два значения, перед этим приводит одно из значений к типу другого. Если значения равны, возвращает `true`.
`===`	Строгое равенство (без приведения типов)	Сравнивает два значения. Если типы значений разные или значения не равны, возвращает `false`.

`!=`	Неравенство (с приведением типов)	Сравнивает два значения, перед этим приводит одно из значений к типу другого. Если значения не равны, возвращает `true`.
`!==`	Строгое неравенство (без приведения типов)	Сравнивает два значения. Если типы значений разные или значения не равны, возвращает `true`.

String и Number

Можно привести числовое значение к строковому типу. Один из способов — использовать команду String:

```
String(число);
```

Чтобы превратить строку в число, используют команду **`Number`**:

```
Number(строка);
```

Приведение к логическому типу

В условии все значения приводятся к логическому типу. Поэтому мы можем использовать в качестве условий любые значения: числа, строки, true и false, а также переменные, которые содержат такие данные.

Все числа, кроме нуля, — true, при этом 0 — false. Все строки, кроме пустой строки, — true, пустая строка '' — false. Можно сказать, что значения, которые как бы ничего в себе не содержат (как 0 или пустая строка ''), приводятся к false, а все остальные приводятся к true.

```
if ('какая-то строка') {
// Непустая строка приводится к true
// Условие выполнится
};

if ('') {
// Пустая строка приводится к false
// Условие не выполнится
};

if (123) {
// Число приводится к true
// Условие выполнится
};

if (0) {
// 0 приводится к false
// Условие не выполнится
};
```

Логические операторы

Отрицание

Чтобы создать проверки с отрицанием, используют унарный (с одним операндом) логический оператор !:

```
let condition = false;

if (!condition) {
  // код выполнится
}
```

И и ИЛИ

Можно комбинировать условия внутри if с помощью логических операторов: «логического И», &&, и «логического ИЛИ», `||`.

Оператор «Логическое И», возвращает true только в том случае, если оба условия, слева и справа от него, возвращают true.

```
true && true;   // Результат: true
true && false;  // Результат: false
false && true;  // Результат: false
false && false; // Результат: false
```

Оператор «логическое ИЛИ», возвращает true если любое из условий слева или справа от него, возвращают true.

```
true || true;   // Результат: true
true || false;  // Результат: true
false || true;  // Результат: true
false || false; // Результат: false
```

Например:

```
let conditionOne = true;
let conditionTwo = true;
let conditionThree = false;
let conditionFour = true;

if (conditionOne && conditionTwo) {
  // код выполнится
}
if (conditionThree || conditionFour) {
  // код тоже выполнится
}
```

Логические операторы можно комбинировать:

```
let conditionOne = true;
let conditionTwo = true;
let conditionThree = false;

if (conditionOne && conditionTwo && !conditionThree) {
  // код выполнится
}
```

«Циклы»

Цикл for

Синтаксис

```
for (let i = 0; i < 10; i++) {
  // Повторяющиеся команды
}
```

В круглых скобках записывается код управления циклом. Он состоит из трёх частей, разделённых ;.

1. Первая часть — подготовительная. Команды отсюда запускаются один раз перед началом работы цикла. Обычно здесь задаётся исходное значение для переменной-счётчика. Обратите внимание, что в цикле мы создаём переменную-счётчик с помощью let, как в случае с любой другой переменной.

```
for (let i = 0; i < 5; i = i + 1) { }
```

2. Вторая часть — проверочная. Она содержит условие и запускается перед каждым новым витком цикла. Если условие возвращает true, цикл делает ещё один виток, иначе цикл завершает свою работу.

```
for (let i = 0; i < 5; i = i + 1) { }
```

3. Третья часть — дополняющая, или «закон изменения». Код третьей части запускается после каждого витка цикла. Обычно там изменяется переменная-счётчик.

```
for (let i = 0; i < 5; i = i + 1) { }
```

Накопление значений в цикле:

Внутри циклов можно использовать обычные математические операции. Например, сложение:

```
let sum = 0;

for (let i = 1; i <= 5; i++) {
  sum += 2;
  console.log(sum);
}
```

Программа выведет:

```
LOG: 2 (number)
LOG: 4 (number)
LOG: 6 (number)
LOG: 8 (number)
LOG: 10 (number)
```

Проверки в теле цикла

Если добавить условие внутрь цикла, то оно будет проверяться на каждой итерации.

```
let sum = 0;

for (let i = 1; i <= 5; i++) {
  if (i > 2) {
    sum += 1;
  }
}
```

Поиск чётного числа

Оператор %, или «остаток от деления», возвращает остаток от деления.

```
10 % 5;  // Вернёт 0
12 % 5;  // Вернёт 2
7 % 3;   // Вернёт 1
5.5 % 2; // Вернёт 1.5
```

Если остаток от деления числа на 2 равен 0 — число чётное, иначе нечётное.

Сокращённые операторы

В JavaScript есть несколько удобных операторов, которые позволяют сократить код:

Название	Пример	Аналог
Инкремент (увеличение на единицу)	`i++`	`i = i + 1`
Декремент (уменьшение на единицу)	`i--`	`i = i - 1`
К-к-комбо!	`i += 2`	`i = i + 2`

Комбинировать можно не только сложение, но и остальные математические операции: вычитание -=, умножение *=, деление /= и нахождение остатка %=.

Цикл while

Синтаксис

```
while (условие) {
  действия
}
```

Действия будут выполняться снова и снова, пока условие не вернёт false.

Чтобы цикл остановился, условие когда-нибудь должно стать ложным. Если условие выхода из цикла не срабатывает, то цикл не может остановиться. Это бесконечный цикл, одна из любимых ошибок программистов.

break и continue

Оператор break прерывает выполнение цикла.

Аналогично оператору прерывания цикла break существует оператор для быстрого перехода к следующей итерации цикла continue, но используют его крайне редко, так как он усложняет чтение кода и понимание работы цикла в целом. Использование continue без необходимости обычно является дурным тоном.

- Внутри `while` команда `continue` «перематывает» программу сразу к началу *следующей* итерации.
- Внутри `for` команда `continue` «перематывает» программу к дополнительной части *текущей* итерации, после выполнения которой начинается *следующая* итерация цикла.

Накопление значений в цикле

```
let sum = 0;
let i = 0;

while (i <= 5) {
  sum += 1;
  i++;
  console.log(i);
}
```

Программа выведет:

```
LOG: 1 (number)
LOG: 2 (number)
LOG: 3 (number)
LOG: 4 (number)
LOG: 5 (number)
LOG: 6 (number) // Код из тела цикла не выполнится, условие вернёт false
```

Поиск процента от числа

Самый простой способ найти процент от числа — разделить число на 100 и умножить на процент.

```
// Найдём 2 процента от 1000
1000 / 100 * 2 = 20;

// Найдём 7 процентов от 1200
1200 / 100 * 7 = 84;
```

Задание. Выполнить упражнения 1–6.

1 Неприличный вопрос

Макс решил принять участие в очередных спортивных соревнованиях. В анкете надо указать, к какой возрастной группе относится участник. Макса этот вопрос поставил в тупик, в душе-то ему всегда два. Нужно написать программу, которая определит возрастную группу Макса — он хочет заранее оценить свои шансы на победу в группе.

Напиши программу, которая будет определять возрастную группу по возрасту.

Возраст записан в переменную `age`.

Проверяй возраст и записывай возрастную группу в виде строки в переменную `ageGroup`.

Если возраст до года включительно, то возрастная группа называется `'Щенки'`.

Если возраст от года (не включая это значение) до трёх лет включительно — `'Молодые собаки'`.

Если возраст от трёх лет (не включая это значение) до семи (включительно) — `'Собаки средних лет'`.

А если возраст от семи лет (не включая это значение) и больше — `'Почтенные таксоны'`.

2. Длительность прогулки

2.1 Длительность прогулки зависит от нескольких условий.

Если идёт дождь, гулять Макс не ходит. В этом случае длительность прогулки равняется 0. А вот если дождя нет, всё зависит от температуры на улице:

Во-первых, если температура от 10°C (включительно) до 15°C (не включая это значение), он гуляет 30 минут. Холодновато, но шерсть спасает.

Во-вторых, если температура от 15°C (включительно) до 25°C (не включая значение), он гуляет 40 минут — погода идеальна.

В-третьих, при температуре от 25°C (включительно) до 35°C (включительно), он гуляет 20 минут — уж очень лапы потеют в такую жару.

В остальных случаях Макс никуда не выходит: либо очень холодно, либо очень жарко. Ему и дома на коврике неплохо.

Переменная isRaining указывает, идёт ли дождь. Если значение переменной true — на улице дождь, если false — дождя нет. Переменная temperature показывает температуру на улице. Результат программы — время прогулки. Его необходимо записать в переменную minutes. Вывести результат в консоль.

2.2 Теперь условия поменялись, и надо написать другой алгоритм.

Длительность прогулки зависит от нескольких условий.

Во-первых, если идёт дождь, прогулка не может состояться. В этом случае длительность прогулки должна равняться 0.

Во-вторых, если температура слишком низкая (ниже 0°C) или слишком высокая (выше 35°C), прогулка тоже не состоится.

В-третьих, идеальная температура для прогулки — 20°C. В этом случае прогулка длится 20 минут.

В остальных случаях длительность прогулки уменьшается на минуту с каждым градусом отклонения от идеальной температуры: при 19°C или 21°C длительность составит 19 минут, при 18°C или 22°C — 18 минут и так далее.

Переменная itsRaining указывает, идёт ли дождь, а temperature — температуру на улице.

Результат необходимо записать в переменную minutes. Вывести результат в alert-функцию

3. Путь к чипсам

Макс любит чипсы, но, чтобы их купить, надо следить за временем — магазин, рынок и фабрика работают по-разному. Напишите программу, которая по текущему времени будет советовать Максу, куда сейчас сходить за чипсами. Каждая торговая точка находится на разном расстоянии от дома Макса, поэтому нужно давать правильные рекомендации. Макс не любит тратить время зря и предпочитает ходить в места поблизости.

Напиши программу, которая определит ближайшее работающее место с чипсами.

Время записано в часах в переменную time. Переменную ввести из окна prompt().

Фабрика чипсов находится ближе всех. Она начинает работать в 8, а закрывается в 19. Перерыв на обед с 13 до 14.

Дальше находится магазин. Он работает с 9 до 17. Перерыв на обед с 14 до 15.

Дальше всех находится рынок. Он работает с 7 до 20 без перерывов.

В остальное время все места закрыты и можно никуда не ходить.

Вычисли, куда надо пойти за чипсами, и запиши значение true в одну из переменных: goToDairy (фабрика), goToStore (магазин), goToMarket (рынок). Вывести результат в alert-функцию.

4. Произведение чётных

Кроме суммы чисел от 1 до n можно ещё найти их произведение. Но в этот раз задача усложнилась — нужно найти произведение не всех чисел из последовательности, а только чётных.

Напишите универсальную программу, которая находит произведение всех чётных чисел из последовательности от 1 до n.

Число, до которого идёт последовательность (включительно), записано в переменную lastNumber

Найдите произведение всех чисел и сохраните результат в переменную multiplicationResult. Вывести результат в alert-функцию.

5. Сколько цифр?

Напиши программу, которая определяет сколько цифр в одном целом числе.

Само число записано в переменную number.

Найди количество цифр в этом числе и запиши результат в переменную quantity.

Подсказка: Чтобы решить эту задачу, можно пойти разными путями. Например, математическим: делить число на 10 и округлять его вниз на каждой итерации.

6. Палиндром

Палиндромы — это слова или фразы, которые одинаково читаются слева направо и справа налево. Среди чисел тоже есть палиндромы. Например, 3223 или 1001.

В этом задании вам нужно написать программу, которая будет определять является ли число палиндромом.

Алгоритм такой: нужно записать изначальное число задом наперёд и сравнить этот вариант с изначальным. Если оба числа равны — перед нами палиндром.

Напиши программу, которая проверяет, является ли число палиндромом.

Число записано в переменную poly.

Переменная, куда нужно записать «перевёрнутую» версию числа, называется ylop.

Выясни, является ли число из переменной poly палиндромом. Если да, значение флага isPalindrome должно быть true, если число не палиндром, то false. Вывести результат в alert-функцию

Все исходные данные вводятся при помощи модального вызова функции prompt(). Результат выводится в alert – окне. В качестве отчета предоставить архив с файлами .js (или директорию на гуглдиске или гитхабе)

Полезные ссылки и материалы

1. https://www.w3schools.com/js/default.asp
2. Основы JavaScript
3. Операторы сравнения (javascript.ru)
4. Условное ветвление: if, '?' (javascript.ru)
5. Циклы while и for (javascript.ru)
6. Конструкция "switch" (javascript.ru)