Линейность/нелинейность программ

В первой лабораторной мы использовали линейные программы - они выполняли одни и те же команды в указанном нами порядке. Нелинейные программы (программы с ветвлением) — в зависимости от разных условий выполняют разные команды.
Для этого используют условный оператор `if`:

```
if (условие) {
  команды, тело условного ветвления;
}
```

Здесь «условие» — это выражение, возвращающее true или false, а «команды» внутри фигурных скобок — это команды, которые выполняются, если условие удовлетворено, то есть если оно вернуло true.
Значения true («истина») или false («ложь») называют логическими, или булевыми. Это отдельный тип данных, который включает только эти два значения.
Начнём работу над программой. В программе используются следующие данные:

- переменная `optimalTime` — сколько минут рекомендуется гулять;
- переменная `walkTime` — сколько минут ты уже гуляешь.

Напишем условие: если `walkTime` больше, чем `optimalTime`, выведем сообщение, что прогулка была достаточно длительной. Чтобы сравнить переменные, используем оператор «больше» `>`.

Задание

1. Объявите переменную `optimalTime`, присвойте ей значение 80, подразумеваем “минут для идеальной прогулки”
2. То же проделайте для `walkTime`, значение 60
3. После объявления переменных добавьте условный оператор: `if (true) { }`
4. Внутри фигурных скобок условного оператора выведите сообщение: `console.log('Прогулка достаточно длинная');`
5. Теперь замените `true` в условии на проверку `walkTime > optimalTime`
6. Теперь замените значение переменной `walkTime` на `90`JS

else

Программа выдаёт сообщение, если Вы гуляете достаточно долго, но молчит, если прогулка слишком короткая. Исправим это создав ветку, которая будет срабатывать, если условие не выполнено. Для этого используем конструкцию 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.
В нашем случае условие такое: если время прогулки больше дневной нормы, то можно остановиться, иначе нужно ещё погулять.
Добавим вторую ветку в программу «Сколько гулять?»

Задание

Добавьте ветку else в программу:

1. После фигурных скобок `if` добавьте `else { }`
2. Внутри фигурных скобок `else` выведите в консоль сообщение «`Нужно ещё погулять`».
3. Замените значение переменной `walkTime` на 60
   Обратите внимание, теперь в консоль выводится другое сообщение.
4. Замените значение переменной `walkTime` на 80 Сообщение в консоли останется прежним

Сравнения, допускающие равенство

Таймер идеальной прогулки уже работает, но недостаточно точно - программа заставляет Вас гулять дольше.
Сейчас программа говорит, что прогулка достаточно длинная, только если время прогулки больше рекомендуемой нормы. Однако по условиям задачи программа должна считать прогулку достаточно длительной, даже если переменные `walkTime` и `optimalTime` равны.
Чтобы исправить программу, используем оператор «больше или равно» `>=`. Он возвращает `true`, если первое число больше или равно второму.

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

Аналогично, если первое число может быть меньше или равно второму, используют оператор «меньше или равно» `<=`.

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

Заменим оператор «больше» в программе «Сколько гулять?» на «больше или равно» и убедимся, что теперь программа не заставляет гулять дольше, чем нужно

Задание

Исправьте программу «Сколько гулять?»:

1. Замените оператор «больше» в условии на «больше или равно».
   Обратите внимание, теперь в консоль выводится другое сообщение

Приведение типов

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
```

Если сравниваются две строки, то к числовому типу они не приводятся, даже если обе строки состоят только из цифр:
console.log('2' > '11'); // Выведет: true
В таблице Unicode цифра 2 идёт после цифры 1, поэтому при посимвольном сравнении строка '2' окажется больше строки '11'

Задание

С помощью команды console.log выведите в консоль перечисленные ниже сравнения и посмотрите на результат. Для каждого сравнения записывайте отдельный вывод в консоль с новой строки:

1. `true <= 0`
2. `'2' > '1'`
3. `'прогулка'< 'ПРОГУЛКА'`

Равенство/неравенство

Так в JavaScript можно проверить значения на равенство и неравенство.

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.

Рассмотрим приведение типов данныхна примерах.

```
console.log('123' == 123); // Выведет true
console.log('123' != 123); // Выведет false
```

Строка и число окажутся равны друг другу. Так происходит, потому что при сравнении разных типов с помощью == значения приводятся к единому типу. Например, строка '123' превращается в число 123. А число 123 равно числу 123, поэтому сравнение возвращает true. По этой же причине неравенство != возвращает для этих значений false — их можно привести к одному типу, и тогда они будут равны друг другу.

```
console.log('123' === 123); // Выведет false
console.log('123' !== 123); // Выведет true
```

В этом случае результат получается противоположным. Строгое равенство === не приводит значения к одному типу, а сравнивает как есть: строку и число. А строка, какое бы значение она ни содержала, не равна числу, поэтому сравнение на равенство возвращает false. Зато строгое неравенство для таких значений возвращает true. Потренируемся использовать операторы равенства.

Операторы «больше или равно» и «меньше или равно» приводят данные к одному типу. В этом случае равенство получается нестрогим

Задание

Используйте операторы равенства и неравенства и понаблюдайте, какой результат выводится в консоль.

1. Объявите переменную `string` со значением ‘123’ и переменную `number` с значением 123
1. Ниже объявления переменных выведите с помощью команды `console.log` результат операции `string == number`
2. Замените нестрогое равенство в сравнении на нестрогое неравенство.
3. А теперь замените нестрогое неравенство на строгое равенство.
4. Наконец, замените строгое равенство на строгое неравенство

Команда String

Мы уже знаем, что JavaScript умеет автоматически приводить данные к определенному типу. Но как быть, если есть два значения разных типов, а на приведение по умолчанию надеяться не хочется? Использовать приведение самостоятельно. Например, можно привести числовое значение к строковому типу. Один из способов — использовать команду `String`:

```
String(число);
```

Вот как она работает:

```
let number = 1;
console.log(number);          // Выведет число: 1 (Number)
console.log(String(number)); // Выведет строку: 1 (String)
```

Чтобы превратить число в строку, также используют метод `toString`.На практике различия между toString и String - в основном в синтаксисе

Задание

Приведём второе значение к строковому типу.

1. Замените в условии == на ===. Приведение перестанет работать, и условие не выполнится.
2. Замените number в условии на `String(number)`. Теперь условие выполняется

Команда Number

Теперь преобразуем строку в число. Для этого можно использовать команду `Number`:

```
Number(строка);
```

Посмотрим, как работает эта команда:

```
let string = '1';
console.log(string);          // Выведет строку: 1 (String)
console.log(Number(string)); // Выведет число: 1 (Number)
```

Для превращения строки в число можно использовать и другие команды, например `parseInt`. Эта команда позволяет указать основание системы счисления: двоичную, десятичную и так далее.

```
parseInt('17', 10);   // Вернёт 17
parseInt('10001', 2); // Вернёт 17
parseInt('11', 16);   // Вернёт 17
```

Обратите внимание, в скобках после parseInt следует указывать два значения:

1. строку, которую мы пытаемся превратить в число;
2. основание системы счисления, в которую мы переводим число

Задание

Теперь сравним числовые значения обеих переменных.

1. Уберите команду String, оставив в сравнении просто number
2. В сравнении замените переменную string на команду Number(string)

Программа «Идеальная прогулка», v.1

Нужно усовершенствовать программу «Сколько гулять?». Входные данные те же:

- рекомендуемое время прогулки в переменной `optimalTime`;
- время, в течение которого ты гулял, в переменной `walkTime`.

Но логику нужно изменить. Программа должна показывать, сколько минут еще осталось гулять. Если ты гулял достаточно, то программа должна показывать ноль.
Сначала продумаем алгоритм решения:

1. заводим новую переменную для хранения времени, которое ещё нужно гулять;
2. проверяем, что время прогулки больше или равно рекомендуемому времени;
3. если да, то записываем в новую переменную ноль;
4. если нет, то вычитаем из рекомендуемого времени время прогулки, а результат сохраняем в новую переменную;
5. выводим красочное информативное сообщение.

Займёмся первыми тремя шагами алгоритма

Задание

1. Используйте код первого таймер “Идеальной прогулки” из упражнения выше
2. Сразу после объявления переменной `walkTime` объявите переменную `timeLeft`
3. Вместо команды `console.log('Прогулка достаточно длинная');` присвойте переменной `timeLeft` нулевое значение

Программа «Идеальная прогулка», v.2

Мы создали переменную и записываем в неё ноль, если гуляли достаточно долго. Что дальше? Вспомним алгоритм:

1. заводим новую переменную для хранения времени, которое ещё нужно гулять;
2. проверяем, что время прогулки больше или равно рекомендуемому времени;
3. если да, то записываем в новую переменную ноль;
4. если нет, то вычитаем из рекомендуемого времени время прогулки, а результат сохраняем в новую переменную;
5. выводим красивое информативное сообщение.

Посчитаем, сколько минут нужно гулять, если рекомендуемая норма не достигнута. После этого выведем в консоль красиво отформатированное сообщение. Для этого используем конкатенацию

Задание

Завершим программу.

1. Замените вывод сообщения `console.log('Прогулка слишком короткая!');` на `timeLeft = optimalTime - walkTime;`
2. После блока else выведите в консоль сообщение, склеенное из трёх фрагментов: строки `'Осталось гулять ещё '`, переменной `timeLeft` и строки `' минут.'`
3. Измените значение переменной `walkTime` на 20
4. Программа готова!

Приведение к логическому типу данных

Рассмотрим новую программу - она будет анализировать несколько условий и выдаст нам экспертное заключение. Вначале она проверяет, находится ли кто-то из разработчиков в отпуске. Информация об этом хранится в переменной `onVacation`:

```
if (onVacation) {
  console.log('Проект нельзя выполнить');
} else {
  console.log('Проект можно выполнить');
}
```

Сложность в том, что информацию об отпусках мы получаем от менеджеров в абсолютно разных форматах. Кто-то сразу скажет, что разработчики в отпуске, то есть передаст true, кто-то не поймёт вопроса и скажет, сколько разработчиков сейчас в отпуске, то есть отдаст нам какое-то число. А кто-то вообще промолчит, если никого в отпуске нет, то есть мы получим пустую строку.
Здесь сравнение значений нам не поможет, потому что данные могут прийти в любом виде. Но есть и хорошая новость — 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.
Главное — понимать, как эти значения приводятся к логическому типу.

Все числа, кроме нуля, — true, при этом 0 — false.

Все строки, кроме пустой строки, — true, пустая строка '' — false.

Все значения, которые “в себе не содержат ничего” - 0, пустая строка '', undefined, null приводятся к false, а все остальные приводятся к true.

Теперь мы можем заняться новой программой. Нужно убедиться, что она работает с разными типами данных. Для этого будем менять значение переменной `onVacation`.
Строка '0' не считается пустой, поэтому приводится к true

Задание

Исправим программу и поэкспериментируем со значением переменной onVacation.

1. Стартовые условия
   ```
   if (onVacation) {
        console.log('Проект нельзя выполнить');
   } else {
        console.log('Проект можно выполнить');
   }
   ```
2. Замените условие в `if` на `onVacation`
3. Теперь измените значение переменной `onVacation` на `true`
4. А теперь — на 2. Число не равно нулю и приводится к true.
5. Замените значение переменной на 0. Сообщение должно измениться.

6. Теперь измените значение на ''
7. И, наконец, удалите значение из переменной, оставив просто `let onVacation;`

Вложенные условия

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. Усложним нашу программу. Теперь реализовать проект можно, если выполняются два условия:

1. есть достаточное количество разработчиков — переменная `enoughDevelopers`;
2. и разработчики владеют требуемыми технологиями — переменная `techAvailable`.

Если разработчиков достаточно и требуемая технология освоена, выводим сообщение «`Проект можно выполнить`», иначе выводим сообщение «`Проект нельзя выполнить`»

Задание

Опишем составное условие с помощью вложенных if.

1. Создайте переменную `enoughDevelopers` со значением `true`
2. Создайте переменную `techAvailable` со значением `false`
3. Изначально нам лучше считать задачу невыполнимой, поэтому создадим переменную `message` со значением `'Проект нельзя выполнить'` и выведем ее в консоль
4. Между объявлением переменной message и выводом в консоль добавьте `if` с условием `enoughDevelopers`
5. Внутри фигурных скобок `if` добавьте ещё один `if` с условием `techAvailable`
6. Внутри фигурных скобок второго `if` присвойте переменной `message` значение `'Проект можно выполнить'`
7. Измените значение переменной `techAvailable` на `true`. Значение `message` должно измениться.

Логические операторы: И, ИЛИ

Добавим в логику принятия решений еще несколько изменений:

- Проект можно выполнить, если разработчиков достаточно и они владеют необходимыми технологиями.
- Проект нельзя выполнить, если кто-то из разработчиков в отпуске или на больничном.

Первую часть логики, ту, которая содержит `и`, мы уже умеем реализовывать с помощью вложенных условий. Но как быть со второй частью, которая содержит `или`?
Мы можем 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
}
```

Теперь мы знаем, как запрограммировать вторую часть логики: скомбинируем условия «сотрудники в отпуске» и «сотрудники на больничном» через «логическое ИЛИ».
Первую часть логики тоже запрограммируем без вложенных if: объединим условия «достаточно разработчиков» и «технология освоена» через «логическое И». Мы отказались от вкладывания условий, так как это может сделать код сложным и

запутанным. Если вложенность большая, то понять, почему выполняется то или иное действие, становится трудно

Задание

Опишем новую логику программы:

1. Добавим в исходные данные:
   ```
   let onVacation = false;
   let onSickLeave = false;
   ```
2. Добавьте первое условие, объединяющее `enoughDevelopers` и `techAvailable` через «логическое И».
3. Внутри условия выведите в консоль сообщение «`Проект можно выполнить`».
4. Добавьте ниже второе условие, объединяющее `onVacation` и `onSickLeave` через «логическое ИЛИ».
5. Внутри второго условия выведите сообщение «`Проект нельзя выполнить`».
6. Измените значение переменной `techAvailable` на false а значение переменной `onSickLeave` на true

Ловушки логики

В текущую программу закралась ошибка. При каких-то условиях программа выдаёт сразу два сообщения. Чтобы поймать ошибку, её сначала надо воспроизвести. Поэтому давайте менять значения входных данных.
При некоторых значениях переменных программа не выведет ни одного сообщения. Это тоже неправильно, и эту ошибку мы также исправим

Задание

По очереди меняйте значения переменных.

1. Укажите, что разработчиков хватает.
2. Укажите, что разработчики владеют технологиями.
3. Укажите, что кто-то из разработчиков в отпуске

Логическое отрицание

Наша ошибка в том, что у программы есть две отдельные проверки. И эти проверки могут сработать одновременно. Когда это случается, появляются два сообщения. Чтобы исправить ошибку, сначала введем новые проверки с отрицаниями:

- нет разработчиков в отпуске;
- нет разработчиков на больничном.

В этих условиях есть ключевое слово «нет», а значит, они должны выполняться в тех случаях, когда значение переменной `false`, и не выполняться, если значение — `true`. Чтобы создать проверки с отрицанием, используют унарный (с одним операндом) логический оператор !. Например:

```
let condition = false;
if (!condition) {
  // код выполнится
}
```

Задание

Работаем с отрицаниями.

1. Измените проверку про отпуск на проверку с отрицанием.
2. То же самое сделайте с проверкой про больничный.
3. Укажите, что кто-то из разработчиков в отпуске.
4. Укажите, что кто-то из разработчиков на больничном

Комбинируем логические операторы

Теперь, когда вы научились использовать отрицания, мы можем описать новую логику программы, которая будет работать без ошибок.
Нам нужно объединить две отдельных проверки в одну общую. Проект можно начать, если:

- разработчиков достаточно
- и они владеют технологиями
- и нет разработчиков в отпуске
- и нет разработчиков на больничном.

Реализовать в коде эту логику достаточно легко, ведь отрицания можно комбинировать с другими логическими операторами. Пример:

```
let conditionOne = true;
let conditionTwo = true;
let conditionThree = false;
if (conditionOne && conditionTwo && !conditionThree) {
  // код выполнится
}
```

Задание

Завершим программу и проверим её поведение.

1. Уберите отдельные проверки на отпуск и больничный
2. Добавьте в конец первой проверки через «логическое И» условие `!onVacation`
3. А после него, тоже через «логическое И», — условие `!onSickLeave`
4. Теперь добавьте ветку `else` и выведите в ней сообщение `'Проект нельзя выполнить'`
5. Исправьте значения переменных так, чтобы условие выполнилось.
6. Затем укажите, что кто-то из разработчиков в отпуске, и убедитесь, что проект нельзя выполнить

Конспект «Условия»

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
}
```

Вторая программа: «Время прогулки»

Следующая программа, которую вам предстоит написать, будет рассчитывать длительность прогулки в зависимости от погодных условий и температуры воздуха. Если идёт дождь, гулять не идем. В этом случае длительность прогулки равняется 0. А вот если дождя нет, всё зависит от температуры на улице:

1. если температура от 10°C (включительно) до 15°C (не включая это значение), гулять 30 минут - прохладно.
2. если температура от 15°C (включительно) до 25°C (не включая значение), гулять 40 минут - отлично.
3. при температуре от 25°C (включительно) до 35°C (включительно), гулять 20 минут - жарко.
4. в остальных случаях остаетесь дома: либо слишком холодно, либо слишком жарко.

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`.

После того, как задание будет выполнено, проведите тесты над своей программой. В случае успеха и отсутствия ошибок - сохраните файл на личном диске

JS стартовый код

```
let temperature = 20;
let isRaining = true;
let minutes = 0;
```

Тесты

1. Идёт дождь, температура — 15. Ожидаю время прогулки 0
2. Дождя нет, температура — 40. Ожидаю время прогулки 0
3. Дождя нет, температура — 15. Ожидаю время прогулки 40
4. Дождя нет, температура — 24. Ожидаю время прогулки 40
5. Дождя нет, температура — 25. Ожидаю время прогулки 20
6. Идёт дождь, температура — 27. Ожидаю время прогулки 0
7. Дождя нет, температура — 12. Ожидаю время прогулки 30
8. Дождя нет, температура — 27. Ожидаю время прогулки 20
9. Дождя нет, температура — 35. Ожидаю время прогулки 20
10. Дождя нет, температура — 5. Ожидаю время прогулки 0

Соревнования

Ты решил принять участие в очередных спортивных соревнованиях. В анкете надо указать, к какой возрастной группе ты относишься как участник. Нужно написать программу, которая определит возрастную группу, чтобы 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 11 включительно, то возрастная группа называется ‘Дети’.
Если возраст от 11 (не включая это значение) до 15 лет включительно — 'Юниоры'.
Если возраст от 15 лет (не включая это значение) до 18 (включительно) — 'Молодёжь'.
А если возраст от 18 (не включая это значение) и больше — 'Взрослые’

JS стартовый код

```
let age = 16;
let ageGroup;
```

Тесты

- возраст 9
- возраст 11
- возраст 21

Рекомендации по питанию

Самому следить за весом утомительно, поэтому нужно запрограммировать умные весы, чтобы они подсказывали, когда нужно взять себя в руки, а когда можно позволить себе отдых от всех правил.
Запрограммируй умные весы, чтобы они давали рекомендации в зависимости от веса.
Вес записан в переменную `weight`.
Рекомендацию записывай строкой в переменную `recommendation`.
Если вес до 48 кг (не включая это значение), рекомендация — `'Пора перекусить'`.
Если вес от 48 кг включительно до 54 кг включительно — `'Вес в норме'`.
Если вес больше 54 кг — `'Пора на тренировку'`

JS стартовый код

```
let weight = 52;
let recommendation;
```

Тесты

- вес 47
- вес 48
- вес 71

FizzBuzz

«Fizz Buzz» — одна из классических программистских задач. Её часто предлагают решить на собеседованиях соискателям на вакансию разработчика. Вот техническое задание:

Программа должна анализировать числа.
Если число делится на 3, результат работы программы — строка `'Fizz'`.
Если число делится на 5 — строка `'Buzz'`.
Если число одновременно делится на 3 и на 5 — результат `'FizzBuzz'`.
В остальных случаях результат работы программы — изначальное число.
Число записано в переменную `number`.
Результат работы программы записывайте в переменную `taskResult`.
Чтобы проверить, делится ли одно число на другое без остатка, используйте оператор «остаток от деления». Он записывается в виде знака процента (%) и возвращает остаток от деления чисел. Работает это так:

```
12 % 5;  // Вернёт 2
27 % 3;  // Вернёт 0
13 % 3;  // Вернёт 1
```

Значение, которое возвращает оператор % — то же самое, что остаток от деления в арифметике. Это деление проще понять на бытовом примере. Представьте, что у вас 13 конфет, а людей в компании 4 (включая вас). Как поделить конфеты на всех, чтобы никого не обидеть? Поровну не получится, потому что 13 не делится на 4 без дробных частей. Зато поровну на 4 части делится число 12. Можно раздать по 3 конфеты каждому. Тогда вы раздадите 12 конфет, и ещё останется одна про запас, потому что изначально конфет было 13. Вот эта конфета «про запас» и есть остаток от деления

JS стартовый код

```
let number = 15;
let taskResult;
```

Тесты

- Число 30. Ожидаю результат: FizzBuzz
- Число 18. Ожидаю результат: Fizz
- Число 10. Ожидаю результат: Buzz
- Число 32. Ожидаю результат: 32

Скидочная система

У тебя есть свой магазин и ты решил дарить скидки на товары.
Напиши программу, которая будет рассчитывать сумму покупки с учётом скидки.
Стоимость записана в переменную price.
Если стоимость покупки от 1000 (включительно) до 3000 (не включая это значение), скидка составляет 5%.
Если стоимость покупки от 3000 (включительно) до 5000 (не включая это значение), скидка 10%.
Если стоимость покупки от 5000 (включительно) и выше, скидка 15%.
В остальных случаях скидки для покупателей нет.
Вычисляй стоимость с учётом скидки и записывай результат в переменную `discountedPrice`

JS стартовый код

```
let price = 4000;
let discountedPrice;
```

Тесты

1. Стоимость 1200. Ожидаю результат: покупка со скидкой: 1140
2. Стоимость 3000. Ожидаю результат: покупка со скидкой: 2700
3. Стоимость 4500. Ожидаю результат: покупка со скидкой: 4050
4. Стоимость 5000. Ожидаю результат: покупка со скидкой: 4250
5. Стоимость 500. Ожидаю результат: покупка со скидкой: 500
6. Стоимость 1000. Ожидаю результат: покупка со скидкой: 950

Поход в магазин

Ты любишь покушать, и надо следить за временем — магазин, рынок и фабрика работают по-разному. Напишите программу, которая по текущему времени будет советовать, куда сейчас сходить за молоком. Каждая торговая точка находится на разном расстоянии от дома, поэтому нужно давать правильные рекомендации.
Напиши программу, которая определит ближайшее работающее место.
Время записано в часах в переменную `time`.
Фабрика 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. Часы округлены до целого. Вычисли, куда надо пойти за покупками, и запиши значение true в одну из переменных: `goToDairy` (фабрика), `goToStore` (магазин), `goToMarket` (рынок)

JS стартовый код

```
let time = 15;

let goToDairy = false;
let goToStore = false;
let goToMarket = false;
```

Тесты

1. Время 12. Ожидаю результат: На фабрику: true, в магазин: false, на рынок: false
2. Время 7. Ожидаю результат: На фабрику: false, в магазин: false, на рынок: true
3. Время 15. Ожидаю результат: На фабрику: true, в магазин: false, на рынок: false
4. Время 13. Ожидаю результат: На фабрику: false, в магазин: true, на рынок: false
5. Время 19. Ожидаю результат: На фабрику : false, в магазин: false, на рынок: true
6. Время 21. Ожидаю результат: На фабрику: false, в магазин: false, на рынок: false

Длительность прогулки

Вы уже писали одну программу для прогулок. Теперь условия изменились, и надо написать другой алгоритм.
Длительность прогулки зависит от нескольких условий:

1. если идёт дождь, прогулка не может состояться. В этом случае длительность прогулки должна равняться 0.
2. если температура слишком низкая (ниже 0°C) или слишком высокая (выше 35°C), прогулка тоже не состоится.
3. идеальная температура для прогулки — 20°C. В этом случае прогулка длится 20 минут.
4. 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`.

JS стартовый код

```
let temperature = 20;
let itsRaining = false;

let minutes;
```

Тесты

1. Идёт дождь, температура — 15°C. Ожидаю время прогулки 0 минут.
2. Дождя нет, температура — 40°C. Ожидаю время прогулки 0 минут.
3. Дождя нет, температура — 15°C. Ожидаю время прогулки 15 минут.
4. Дождя нет, температура — 22°C. Ожидаю время прогулки 18 минут.
5. Идёт дождь, температура — 25°C. Ожидаю время прогулки 0 минут.
6. Дождя нет, температура — 41°C. Ожидаю время прогулки 0 минут.