На старт

Для начала подготовим рабочую среду. Для этого необходимо открыть браузер Google Chrome и консоль в нем(клавиша F12, сочетание Ctrl+Shift+I, правый клик на пустом месте новой вкладки - “Просмотреть код”. Также будем использовать блокнот, для сохранение частей кода. Именуйте файлы согласно текущего задания, на английском, с использованием kebab-case, например “first-program.js”
Первая задача: написать программу, которая считает, сколько времени вы провели в колледже за последние два дня:

- первый день — 3 часа,
- второй день — 4.5 часа.

Каждая программа — это набор команд. Чтобы решить эту задачу, достаточно одной команды:

```
3 + 4.5
```

! Обратите внимание, для указания десятичной дроби используется точка. Запятая в данном случае не годится — её JavaScript интерпретирует как перечисление.

Задание

Напишите программу для расчёта времени в соцсетях:

1. введите в консоль команду 3 + 4.5 и посмотрите на результат

Арифметика в JS

Вторая задача: написать программу для расчёта времени в часах и минутах.
Эту программу с помощью одной команды не написать, понадобятся две команды — для часов и для минут. В JS команды разделяются точкой с запятой ;.
Для расчёта времени в часах достаточно сложить два числа — мы сделали это в прошлом задании. А чтобы получить из часов минуты, надо умножить время в часах на 60:

```
(3 + 4.5) * 60;
```

Арифметические операции в JavaScript выполняются так же, как в математике: сначала умножение, потом сложение. Изменить порядок операций можно с помощью круглых скобок. Снова как в математике: выражение в скобках посчитается в первую очередь. Напишем программу для расчёта времени в часах и минутах. И, конечно, не забудем про точки с запятой.

Задание

Напишите программу, которая подсчитывает время в часах и в минутах.

1. Напишите в редакторе команду для подсчёта времени в часах: 3 + 4.5;.
2. На следующей строке напишите команду для подсчёта времени в минутах: (3 + 4.5) * 60; — и посмотрите на результат в консоли.
   Обратите внимание, в консоль выводится только одно значение.

Вывод в консоль, комментарии

Давайте дорабатывать наш код. Но сначала разберёмся, как работает текущая версия. JavaScript выполняет программу последовательно, команда за командой, и выводит в консоль результат выполнения последней команды. В текущей версии программы последняя команда возвращает результат 450. Именно его мы и видим в консоли. Чтобы вывести дополнительную информацию в консоль, воспользуемся командой console.log:

```
console.log(данные для вывода в консоль);
```

Эту команду можно использовать в любом месте программы и выводить в консоль результаты выполнения операций и текстовые подсказки:

```
console.log(2 * 2);
console.log(Hello, world!');
```

Обратите внимание, текстовые подсказки, в отличие от результатов операций, нужно заключать в кавычки. Есть и другой способ пояснить код — использовать комментарии. Они не выводятся в консоль и не влияют на работу программы, но видны разработчику. Код внутри комментариев не выполняется. Обычно в них пишут поясняющие тексты для себя, или для других, или для себя в будущем. Комментарии бывают двух типов: однострочные и многострочные:

```
// Эта строка кода не выполнится. Однострочный комментарий.

/*
Все эти строки кода не выполнятся.
Так как это многострочный комментарий.
*/
```

Нам необходимо, чтобы подсказки выводились в консоль. Используем команду console.log, чтобы улучшить программу.

Задание

Улучшите программу для расчёта времени.

1. Напишите команду console.log('Время в часах:');
2. На следующей строке напишите команду вывода суммы часов, проведенных в колледже;
3. На 3 строке напишите команду console.log('Время в минутах:');
4. Далее команду вывода времени в минутах;

//Обратите внимание на undefined в консоли на последней строке

Типы данных

Итак, программа работает как необходимо: показывает все результаты и подсказки. Проясним детали, вспомнив результат работы в консоли:

```
Время в часах:
7.5
Время в минутах:
450
undefined
```

Простые строки вывода в браузере Google Chrome не имеют значков в начале строки. Значок “стрелка влево” показывает значение, которое **возвращает** программа после выполнения. Что значит «возвращает»? Программа может выполнить код и вернуть, то есть отдать результат своей работы для дальнейшего использования.

Например, вы пытаетесь вспомнить, сколько пар будет завтра. Вы решили позвонить другу:
— Сколько завтра пар? — спрашиваете Вы.
— Четыре, — быстро отвечает вам друг.
— Значит, надо подготовиться к 4м парам! — решаете вы.
Своим ответом друг «вернул» вам значение «четыре». Вы получили ответ на свой вопрос и смогли что-то сделать с этим ответом. А вот если бы вы, во время телефонного разговора,спросили друга о парах и попросили написать результат на листочке, вы бы не услышали в ответ «четыре». Друг бы записал ответ на бумаге, но вы бы ничего не смогли сделать с этим результатом.
Точно так же работает console.log. Вы просите команду вывести произведение чисел console.log(3+4.5), она выводит в консоль 7.5 и всё. Как ваш друг, который просто записывает ответ, ничего вам не говоря. Получается, что на самом деле console.log ничего не возвращает, то есть возвращает «ничего». Такое «отсутствие значения» в JavaScript обозначается ключевым словом undefined («не определено», англ.). Его мы и видим в последней строчке.
Команда console.log выводит в консоль и другую информацию. Например, она показывает тип выводимых данных. С разными типами данных можно производить разные действия, поэтому программисту важно знать, с чем он работает. В нашей консоли чтобы получить тип данных необходимо использовать встроенную функцию `typeof`, чтобы получить тип переменной (например (String) или (Number)).
Выведем в консоль разные данные и посмотрим на их тип

Задание

С помощью команды console.log (typeof ()) выведите в консоль перечисленные ниже значения и посмотрите, какой у них будет тип. Для каждого значения записывайте отдельный вывод в консоль с новой строки:

1. 3
2. 2.5
3. ‘Hello, world!’

4. true

Переходим к сложным типам

Мы познакомились с тремя типами данных:

- `number` — числа: целые и с точкой;
- `string` — строки;
- `boolean` — логические значения: true — «истина» и false — «ложь».

На самом деле мы познакомились с четырьмя типами, так как undefined — это особый тип данных, включающий одно значение — undefined.
Кстати, в некоторых языках программирования целые и дробные числа относятся к разным типам данных. Но в JavaScript это один тип — `number`.
Все вышеперечисленные типы данных — простые, или примитивы. Но в JavaScript есть и более сложные, или составные, типы данных: массив и объект.
Составные типы содержат не одно, а несколько значений. Массив, `array`, хранит последовательность значений, и порядок этих значений важен. Объект, `object`, состоит из множества пар «ключ-значение», порядок этих пар не важен. Записываются эти типы данных так:

```
// Массив
[1, 2, 3, 4, 5]

// Объект
{month: 'june', day: 15}

// Вывод массива в консоль
console.log([1, 2, 3, 4, 5]);

// Вывод объекта в консоль
console.log({month: 'june', day: 15});
```

Задание

С помощью команды console.log выведите в консоль перечисленные ниже значения и посмотрите, какой у них будет тип. Для каждого значения записывайте команду с новой строки:

1. [50, 320, 480]
2. {media: 'Twitter', time: 5}

Неизвестные данные

Новая задача - написать программу для расчёта времени на парах. На ввод будем получать, сколько часов ты провел сегодня в колледже, и видеть это время в минутах. Ранее мы писали программы, которые работали с известными данными. И эти данные находились прямо в коде программы. Но намного чаще вам придётся работать с данными, которые на момент запуска программы неизвестны. Например, собирать пользовательский ввод и обрабатывать его.
Для начала вспомним как получать данные от пользователя. Для этого нам пригодится команда `prompt()`:

```
prompt('сколько часов ты провел в колледже?');
```

Обратите внимание, что текст вопроса — это строка, поэтому он заключается в кавычки. Кавычки могут быть одинарными, 'строка', или двойными, "строка".
Потренируемся использовать команду `prompt()`

Задание

1. Напишите в редакторе команду prompt() с вопросом ‘сколько часов ты провел в колледже?’.

Обратимся к переменным

Программа получила данные, и теперь ей нужно где-то сохранить их для дальнейшего использования. Для этого обработаем данные. Все они сохраняются в памяти компьютера. А память устроена сложно, она использует сложные адреса. Раньше с памятью работали примерно так:

```
put 0xEC002...0xEC003 1 // Сохранили число 1 в ячейку памяти
get 0xEC002...0xEC003   // Достали число 1 из ячейки памяти
```

С такими адресами работать неудобно: очень сложно запомнить, что и зачем туда сохранено. Поэтому было придумано простое решение — переменные.

```
put my_number 1 // Сохранили число 1 в переменную my_number
get my_number   // Достали число 1 из переменной my_number
```

Переменная — просто название для данных, которое можно делать понятным для людей. Переменные упрощают работу с памятью: они «приклеиваются» к ячейкам памяти, как наклейка с названием приклеивается к папке с документами.
В JavaScript переменные можно создавать командой `let`, за которой следует имя переменной:

```
let имяПеременной;
```

Имя переменной можно записать по-разному. Два самых популярных способа: camelCase (верблюжья нотация) и snake_case (змеиная нотация). В первом случае все слова пишутся слитно и каждое слово, за исключением первого, начинается с большой буквы (myNumber, userName). Во втором случае все слова разделяются нижним подчеркиванием (my_number, my_name). Чаще в JS используется camelCase.
Имена переменных в JavaScript чувствительны к регистру: myname и myName — две разные переменные. Имя переменной может содержать буквы, цифры и знак подчеркивания, но оно не должно начинаться с цифры. Кроме того, в качестве имени переменной нельзя использовать ключевые слова, такие как let или if. Полный список ключевых слов можно найти в библиотеке https://developer.mozilla.org/ по запросу Зарезервированные ключевые слова в ECMAScript 2015.
Имя переменной должно отражать то, что в ней хранится.
После создания переменной ее можно использовать в других командах, например, выводить в консоль:

```
// Обратите внимание, что кавычек нет!
console.log(имяПеременной);
```

Создадим переменную timeInHours («время в часах», англ.) и выведем её в консоль

Задание

1. Создайте переменную с именем timeInHours

2. Затем выведите эту переменную в консоль

Объявление и присваивание переменных

Мы вывели переменную в консоль и получили undefined. Почему именно его? Когда мы создаём переменную, программа просто запоминает имя новой переменной, но никакие данные в неё не записывает. Если обратиться к пустой переменной, то получим undefined — «не определено».
Можно создать, или объявить, переменную и не сохранять в неё никакие данные. Иногда это делают, чтобы «застолбить» имя переменной на будущее.
Но намного чаще мы будем создавать не пустые переменные. Чтобы записать в переменную данные, ей их нужно присвоить. Для операции присваивания используется знак равенства:

```
let timeInHours;              // Объявляем переменную
console.log(timeInHours);     // Выведет: undefined

timeInHours = 2;              // Присваиваем одно значение
console.log(timeInHours);     // Выведет: 2

timeInHours = 'три часа';     // Присваиваем совершенно другое
значение
console.log(timeInHours);     // Выведет: три часа
```

Обратите внимание на две особенности.

1. команда `let` для создания каждой переменной используется всего один раз. Дальше мы обращаемся к переменной по её имени, без let.
2. если повторно задать значение переменной, то значение этой переменной изменится. Предыдущее значение при этом исчезнет. Это называется переопределением переменной.

Чаще всего одновременно объявляется переменную и присваивается значение. Причём это значение может быть получено из какой-нибудь команды. Например:

```
let minutesPerHour = 60;
let timeInHours = prompt('сколько часов ты провел в колледже?');
let studyInTwoDays = 3 + 4.5;
```

Получим данные с помощью команды prompt, сохраним их в переменную и выведем в консоль

Задание

1. Объявите переменную timeInHours
2. На следующей строке присвойте этой переменной значение команды prompt с вопросом «сколько часов ты провел в колледже?».
3. На следующей строке выведите в консоль текст-подсказку «Проведено в колледже часов:».
4. И на последней строке выведите переменную в консоль

Операции, операторы и операнды

Чтобы писать универсальные программы, мы должны были научиться:

1. получать неизвестные данные;
2. сохранять эти данные в переменные для дальнейшего использования;
3. обрабатывать данные, то есть проводить над ними операции.

С первыми двумя пунктами мы разобрались. Теперь перейдём к операциям.
Посмотрите на такую команду:
5 + 10;
Это операция. Она состоит из оператора **+**, и двух операндов **5** и **10**.
Оператор указывает, что произойдёт с операндами. В команде выше мы используем плюс, а значит, это операция сложения, и операнды (слагаемые) сложатся, и команда вернёт результат сложения (сумму).
Операция сложения — бинарная, так как в ней два операнда. Бинарные операции самые распространённые. Но существуют и унарные операции, с одним операндом, и тернарные операции, с тремя операндами.
В качестве операндов мы можем использовать переменные:

```
firstDay + secondDay;
timeInHours * 60;
5 + studyInTwoDays;
```

Над разными типами операндов можно производить разные операции, поэтому важно понимать, данные какого типа хранятся в переменных. Узнать, к какому типу относятся данные, можно с помощью команды `typeof`.
Попрактикуемся в операциях над числами. Вот некоторые из них:

Сложение	+
Вычитание	-
Умножение	*
Деление	/

Задание

С помощью команды console.log выведите в консоль результаты операций с переменными. Значения самих переменных не меняйте.

1. Создайте переменную firstDay со значением 3;
2. Создайте переменную secondDay со значением 4.5;
3. Сложите две переменные, чтобы получить общее время, проведенное в колледже.
4. Вычтите из secondDay firstDay чтобы узнать, на сколько часов больше ты пробыл в колледже во второй день.

5. Разделите firstDay на 24 а затем умножьте на 100 чтобы узнать, сколько процентов времени ты провёл в колледже в первый день.

Приоритет операций

Мы уже знаем, что программа состоит из команд, а команды состоят из операций. В одной команде операций может быть несколько. В прошлом задании мы написали такой код:

```
console.log(firstDay / 24 * 100);
```

Команда, результат которой выводится в консоль, состоит из двух операций: деления и умножения. JavaScript выполнит эти операции в том порядке, в котором они записаны, слева направо. Но так происходит не всегда.
Порядок выполнения операций зависит от их приоритета. Если у операций одинаковый приоритет, они выполняются слева направо. Но что если приоритет разный?
Рассмотрим такой код:

```
let timeInMinutes = (3 + 4.5) * 60;
```

В этой команде две арифметические операции и операция присваивания. В каком порядке они выполнятся?
У выражения в круглых скобках самый высокий приоритет, поэтому сначала JavaScript сложит числа 3 и 4.5. Следующее по приоритету — умножение: JavaScript умножит сумму — 7.5 — на 60. У операции присваивания низкий приоритет, она выполнится в последнюю очередь. В итоге в переменную запишется результат умножения — число 450. Приоритет различных операторов можно посмотреть на https://developer.mozilla.org/, статья “Приоритет операторов”.
Сложные команды можно упростить, разбив их на несколько шагов. Например, мы могли бы сохранить сумму в промежуточную переменную и использовать её на следующем шаге:

```
// Этот код даст тот же результат
let studyInTwoDays = 3 + 4.5;
console.log(mediaInTwoDays * 60);
// Что и этот
console.log((3 + 4.5) * 60);
```

Разработчик сам решает, использовать сложные команды из нескольких операций или разбивать их на несколько шагов, состоящих из простых операций. Но есть золотое правило — чем проще код, тем лучше.
Попрактикуемся разбивать и использовать сложные команды

Задание

1. Посчитайте время в минутах. Для этого сохраните в переменную timeInMinutes результат выражения studyInTwoDays * 60
2. а затем выведите переменную timeInMinutes в консоль.

3. Узнайте, сколько минут в день можно провести в колледже в оставшиеся 4 дня, если недельная норма 1800 минут. Для этого сохраните в переменную minutesLeftPerDay результат выражения (1800 - timeInMinutes) / 4
4. Выведите переменную minutesLeftPerDay в консоль

Конкатенация

В прошлых заданиях мы использовали числа и арифметические операции, но у каждого типа данных свои операции. Например, арифметические операции можно производить с числами, но не со строками.
Самая частая строковая операция — это «склеивание» строк, или конкатенация:

```
let name = 'Иван';
// Обратите внимание на пробелы
'Студент' + 'Иван';  // Результат: 'СтудентИван'
'Студент ' + 'Иван'; // Результат: 'Студент Иван'
'Студент ' + name;   // Результат: 'Студент Иван'
```

Для склеивания строк используется тот же знак, что и для сложения чисел, — плюс.
Как JavaScript понимает, какую операцию применить, сложение или конкатенацию? Он смотрит на тип операндов: если это строки, то они склеиваются, если это числа, то они складываются.
Но что будет, если операнды разного типа? Например:

```
'Время, мин: ' + 50; // Результат: 'Время, мин: 50'
'2' * 50;            // Результат: 100
```

В этом случае JavaScript попытается привести операнды к одному типу и выполнить операцию. Подходящий тип будет выбираться в зависимости от операции.
Плюс может быть знаком сложения или конкатенации, но так как один из операндов — строка, то сложение не подходит. Поэтому число 50 приводится к строке '50' и склеивается со строкой 'Время, мин: '.
Звёздочка — это знак умножения, со строками она не используется. Поэтому JavaScript пытается превратить строку '2' в число, и ему это удаётся. Затем числа 2 и 50 перемножаются, и получается 100.
Из-за того, что JavaScript умеет изменять тип операндов на лету, он называется языком со слабой типизацией. Есть много тонкостей и проблем с приведением типов..
Сейчас условимся, что в наших первых программах проблем с приведением типов не будет, так как программы простые, а пользователи вводят данные аккуратно.
Вернёмся к конкатенации. Зачем её используют? Конкатенация позволяет делать сообщения программ более информативными и «человечными». Убедимся в этом на практике

Задание

Сделайте вывод в консоль более информативным, склеив переменные со строками-подсказками. Для этого:

1. Внутри первой команды console.log замените timeInMinutes на 'Ты провел в колледже ' + timeInMinutes + ' минут.'
2. Внутри второй команды console.log замените minutesLeftPerDay на 'На этой неделе осталось примерно по ' + minutesLeftPerDay + ' минут в день.'

Релиз Учебного таймера v0.1, часть 1

В прошлом задании мы познакомились с конкатенацией. Теперь мы готовы полностью решить третью задачу. И не просто решить, а сделать программу удобной и понятной.
Вспомним задание:
Третья задача: «Напиши программу для расчёта времени в колледже. Хочу вводить, сколько часов я провёл в соцсетях, и видеть это время в минутах».
Разложим ход решения по шагам:

1. получаем данные о времени;
2. сохраняем эти данные в переменную;
3. выводим сообщение о том, что данные получены;
4. обрабатываем данные с помощью математических операций, в которых используем первую переменную;
5. результат обработки сохраняем во вторую переменную;
6. выводим сообщение с результатом работы программы.

Сейчас разберёмся с первыми тремя шагами, а в следующем задании — с оставшимися

Задание

1. Объявите переменную `timeInHours`
2. сохраните в неё данные из команды `prompt()` с вопросом ‘сколько часов ты провел в колледже?’.
3. Выведите в консоль сообщение, склеенное из трёх фрагментов: строки `'Данные получены. Проведено '`, переменной `timeInHours` и строки `' часов.'`

Релиз Учебного таймера v0.1, часть 2

Мы получили данные, сохранили их в переменную, а после вывели сообщение об этом в консоль. Осталось обработать данные и вывести результат.
Чтобы перевести часы в минуты, надо умножить их на 60. Итоговая формула: времяВЧасах * 60
Результат выражения сохраним во вторую переменную и выведем в консоль информативное сообщение. Задача решена, осталось только написать код!
Теперь вы готовы к написанию программы с нуля без посторонней помощи.

Задание

1. Объявите переменную `timeInMinutes`,
2. Сохраните в неё результат, рассчитанный по формуле времяВЧасах * 60.
3. Выведите в консоль сообщение, склеенное из трёх фрагментов: строки `'На учебе проведено '`, переменной `timeInMinutes` и строки `' минут!'`
4. Сохраните последний рабочий фрагмент кода в текстовый файл на своем личном диске, создав новую папку “Практика Javascript”

Конспект «Основы программирования на JavaScript»

Программа — это набор команд. JavaScript выполняет программу последовательно, команда за командой. Команды разделяются точкой с запятой ;.

Консоль

Чтобы вывести информацию в консоль, используем команду console.log:
console.log(данные для вывода в консоль);
Эту команду можно использовать в любом месте программы и выводить в консоль результаты выполнения операций и текстовые подсказки. Текстовые подсказки, в отличие от результатов операций, нужно заключать в кавычки.

Комментарии

Комментарии не выводятся в консоль и не влияют на работу программы, но видны разработчику. Код внутри комментариев не выполняется. Обычно в них пишут поясняющие тексты для себя, или для других, или для себя в будущем.
Комментарии бывают двух типов: однострочные и многострочные:

```
// Эта строка кода не выполнится. Однострочный комментарий.

/*
Все эти строки кода не выполнятся.
Так как это многострочный комментарий.
*/
```

Типы данных

С разными типами данных можно производить разные действия, поэтому программисту важно знать, с чем он работает. В нашей консоли тип данных выводится в скобках, например (String) или (Number).
Существуют простые и сложные типы данных. Простые:

- number — числа: целые и с точкой;
- string — строки;
- boolean — логические, или булевы, значения: true — «истина» и false — «ложь»;
- undefined — «не определено», англ.

Строки нужно оборачивать в кавычки: одинарные или двойные.
Сложные, или составные, типы содержат не одно, а несколько значений. Массив, array, хранит последовательность значений, и порядок этих значений важен. Объект, object, состоит из множества пар «ключ-значение», порядок этих пар не важен.

```
// Массив
[1, 2, 3, 4, 5]

// Объект
{month: 'june', day: 15}
```

Переменные

Переменная — просто название для данных, которое можно делать понятным для людей. Переменные упрощают работу с памятью: они «приклеиваются» к ячейкам памяти, как наклейка с названием приклеивается к папке с документами.
В JavaScript переменные можно создавать командой let, за которой следует имя переменной:
let имяПеременной;
Имя переменной можно записать по-разному. Два самых популярных способа: camelCase (верблюжья нотация) и snake_case (змеиная нотация). В первом случае все слова пишутся слитно и каждое слово, за исключением первого, начинается с большой буквы (myNumber, userName). Во втором случае все слова разделяются нижним подчёркиванием (my_number, my_name).
Имена переменных в JavaScript чувствительны к регистру: myname и myName — две разные переменные. Имя переменной может содержать буквы, цифры и знак подчёркивания, но оно не должно начинаться с цифры. Кроме того, в качестве имени переменной нельзя использовать ключевые слова, такие как let или if. Вот полный список этих ключевых слов.
После создания переменной её можно использовать в других командах, например, выводить в консоль:

```
// Обратите внимание, что кавычек нет!
console.log(имяПеременной);
```

Если обратиться к пустой переменной, то получим undefined — «не определено».
Чтобы записать в переменную данные, ей их нужно присвоить. Для операции присваивания используется знак равенства:

```
let timeInHours;              // Объявляем переменную
console.log(timeInHours);     // Выведет: undefined

timeInHours = 2;              // Присваиваем одно значение
console.log(timeInHours);     // Выведет: 2

timeInHours = 'три часа';     // Присваиваем совершенно другое значение
console.log(timeInHours);     // Выведет: три часа
```

Команда let для создания каждой переменной используется всего один раз. Дальше мы обращаемся к переменной по её имени, без let. Если повторно задать значение переменной, то значение этой переменной изменится. Предыдущее значение при этом исчезнет. Это называется переопределением переменной.

Операции и операторы

Команды состоят из операций. 5 + 10; — это операция. Она состоит из оператора, +, и двух операндов, 5 и 10.
Оператор указывает, что произойдёт с операндами. Операции бывают унарными, бинарными и тернарными, в зависимости от количества операндов. Бинарные операции самые распространённые.
Над разными типами операндов можно производить разные операции, поэтому важно понимать, данные какого типа хранятся в переменных.

Порядок выполнения операций зависит от их приоритета. Если у операций одинаковый приоритет, они выполняются слева направо. Приоритет различных операторов можно посмотреть на https://developer.mozilla.org/ru/ статья “Приоритет операторов”.

Арифметические операции

Арифметические операции в JavaScript выполняются так же, как в математике: сначала умножение, потом сложение. Изменить порядок операций можно с помощью круглых скобок. Снова как в математике: выражение в скобках считается в первую очередь.

Сложение	+
Вычитание	-
Умножение	*
Деление	/

Конкатенация

Самая частая строковая операция — это «склеивание» строк, или конкатенация:

```
let name = 'Иван';
'Студент' + 'Иван';  // Результат: 'СтудентИван'
'Студент ' + 'Иван'; // Результат: 'Студент Иван'
'Студент ' + name;   // Результат: 'Студент Иван'
```

Конкатенация позволяет делать сообщения программ более информативными и «человечными».

Приведение типов

Что будет, если использовать операнды разного типа?
'Время, мин: ' + 50; // Результат: 'Время, мин: 50'
'2' * 50; // Результат: 100
JavaScript попытается привести операнды к одному типу и выполнить операцию. Подходящий тип будет выбираться в зависимости от операции.
Плюс может быть знаком сложения или конкатенации, но так как один из операндов — строка, то сложение не подходит. Поэтому число 50 приводится к строке '50' и склеивается со строкой 'Время, мин: '.
Звёздочка — это знак умножения, со строками она не используется. Поэтому JavaScript пытается превратить строку '2' в число, и ему это удаётся. Затем числа 2 и 50 перемножаются, и получается 100.
Из-за того, что JavaScript умеет изменять тип операндов на лету, он называется языком со слабой типизацией

Первая программа: Отработка пропущенной физкультуры

Ваша задача — написать калькулятор. Техническое задание:
За каждый пропуск физры на следующем занятии необходимо провести отработку.
На ввод приходит сколько пар было пропущено, а программа показывает сообщение о количестве необходимых отработок. Тренировка состоит из бега по кругу (400м).
Сообщение выглядит так: «`За <число> пропущенных полагается пробежать <число> км.`»
Вот алгоритм работы калькулятора:
Количество пропущенных пар хранится в переменной `classesMissed`.

За каждую пропущенную пару нужно пробежать 15 кругов.
Например: за 5 пропущенных, я должен пробежать 75 кругов.
Км храним в переменной `debt`.

Сообщение склеиваем из трёх строк: `'За ', ' пропущенных, я должен пробежать ', ' км.'` – и двух переменных: `classesMissed` и `debt`.

Сообщение записываем в переменную `message`.
Значения переменных вывести в консоль. Файл сохранить на личном диске

JS стартовый код

```
let classesMissed = prompt('Сколько пар ты пропустил?');
```

Конвертер валют

Представляем, что у тебя появилась возможность отдохнуть за рубежом. Ты изучил цены и понял, что для комфортной поездки нужно 500 евро (на всякий случай) и 2500 баксов. Но у тебя есть только рубли.
Напиши программу, которая посчитает сколько всего рублей мне понадобится на поездку.
Курсы валют указаны в переменных `euroRate` и `dollarRate`.
Переменные `euroAmount` и `dollarAmount` — необходимые суммы на поездку.
Создай переменную rublesAmount и записывай в неё результат вычислений.

JS стартовый код

```
let euroRate = 74;
let dollarRate = 63;

let euroAmount = 500;
let dollarAmount = 2500;
```

Твоя первая кредитная история

Следующим этапом пришло осознание, что денег на отдых не хватает. А тебя уже не первую неделю донимает *выбрать_цвет* банк с щедрым предложением - менеджер банка мечтает дать денег на неограниченный срок, но вернуть надо будет в два раза больше денег.
Напиши программу, которая посчитает, сколько в итоге денег ты должен отдать после поездки.

В переменной `travelCost` хранится сумма необходимая на поездку.
В переменной `balance` находится сумма, которая есть у меня сейчас.
Узнай, сколько ты должен отдать банку и запиши результат в переменную `debtAmount`.

JS стартовый код

```
let travelCost = 150000;
let balance = 100000;
```

Авиасейлз на минималках

Ты готов вылетать, но осталась задача выбрать, каким рейсом. Чем быстрее - тем лучше. Посчитай сколько времени займет мой перелёт.
В переменную `flightDistance` записано расстояние полёта в километрах.
В переменной `averageSpeed` находится средняя скорость самолёта (километры в час).
Найди время полета (в часах) и запиши его в переменную `flightTime`.
Округляй результат вычислений с помощью команды `Math.round()`.
Команда округления `Math.round()` округляет дробное число к ближайшему целому числу. Записывается это так:

```
Math.round(40.15); // Вернёт 40
Math.round(12.75); // Вернёт 13
// Можно использовать переменные
var number = 23.055555;
Math.round(number); // Вернёт 23
```

Всё как в математике. Если после точки число меньше 5, то дробная часть отбрасывается и ностается еизменная целая часть. А если число после точки больше или равно 5, то дробная часть убирается, а целое число увеличивается на 1.

JS стартовый код

```
let flightDistance = 7260;
let averageSpeed = 600;
```

Блаблакар на минималках

Из-за неблагоприятных погодных условий твой самолет приземляется не в городе назначения, а в соседнем. Необходимо добраться в твой город-курорт, учитывая что в этой стране все измеряется в милях.
Средняя скорость транспорта находится в переменной `averageSpeed` и записана в километрах в час.
Расстояние записано в переменную `routeDistance` и указано в милях. Поэтому сначала нужно перевести его в километры.
В одной миле содержится 1.6 километров.
Посчитай сколько времени займёт поездка и запиши результат в переменную `routeTime`.
Округляй результат с помощью `Math.round`

JS стартовый код

```
let routeDistance = 78;
let averageSpeed = 80;
```

Погода

В целом твой выбор пал на страну с имперской системой мер, поэтому твои мучения продолжаются. Ты хочешь узнать температуру - но на всех градусниках температура указана по Фаренгейту, а не привычная тебе по Цельсию.
Напиши программу, которая будет переводить градусы Фаренгейта в температуру по Цельсию.
В переменной `fahrenheitTemperature` указаны градусы по Фаренгейту.
Формула расчёта выглядит так: Tc = 5/9 * (Tf - 32).
Результат вычислений запиши в переменную `celsiusTemperature`

JS стартовый код

```
let fahrenheitTemperature = 77;
```

Тренируйся с умом!

В той замечательной вселенной, в который ты уехал отдыхать в какую-то из стран с имперской системой мер, ты, также, являешься спортсменом, яро соблюдающим режим тренировок. Чтобы не нарушать режим, к которому ты привык, нужно держать под контролем индекс массы тела (ИМТ) и процент жира в организме. ИМТ легко найти по формуле, зная рост и вес. Эти показатели тебе известны. А вот с процентом жира чуть сложнее — массу жира можно узнать с помощью весов. Посчитаем ИМТ и процент жира в организме.
Запишем рост в переменную `length`, в метрах.
Вес указан в килограммах и хранится в переменной `weight`.
Рассчитывай ИМТ по формуле: i = вес / длина ^ 2 (вес делить на длину в квадрате).
Результат округляй с помощью `Math.round` и записывай в переменную `bodyMassIndex`. Масса жира записана в переменную `fatMass` и указана в килограммах.
Чтобы найти процент жира, дели массу жировой ткани на вес. Результат умножай на 100, чтобы получить процент, округляй с помощью `Math.round` и записывай в переменную `fatPercent`.
В формуле i = вес / длина ^ 2 символ `^` обозначает возведение в степень. Такое обозначение используется в некоторых языках программирования и компьютерной алгебре. Этот символ не используется в JavaScript. Вы можете заменить его обычным умножением, как в математике.

```
6 ^ 2 = 6 * 6 = 36;
// 6 во второй степени
2 ^ 3 = 2 * 2 * 2 = 8;
// 2 в третьей степени
```

JS стартовый код

```
let height= 1.83;
let weight = 75;
let fatMass = 0.8;
```

Вечерняя пробежка

Вечером, на легке, ты решаешь покорить местных в близлежащем парке своей атлетичной фигурой. Эстетики ради с собой ты не берешь никаких трекеров, только плеер. Но для эффективной тренировки надо знать длину дорожки в парке. Оказалось, что она повторяет по форме прямоугольный треугольник. Ты промерил шагами длины катетов, а вот измерять гипотенузу решил математически, с помощью программы.
Напиши программу, которая находит периметр треугольника через катеты.
Длины катетов указаны в метрах и хранятся в переменных `firstLeg` и `secondLeg`.
Найди гипотенузу по формуле a = sqrt(b^2 + c^2) и запиши результат в переменную `hypotenuse`. Для того чтобы найти квадратный корень числа, используй `Math.sqrt()`. Результат округляй с помощью `Math.round()`.
Найди периметр треугольника, сложив длины катетов и гипотенузы. Результат запиши в переменную `perimeter`.
Команда `Math.sqrt()` возвращает квадратный корень числа. Это то же самое, что знакомый вам знак квадратного корня из математики — √:

```
Math.sqrt(9);  // Вернёт 3
Math.sqrt(64); // Вернёт 8
Math.sqrt(1);  // Вернёт 1
```

JS стартовый код

```
let firstLeg = 300;
let secondLeg = 700;
```

Эффективная ЧСС

Все таки не взять трекер было ошибкой, ведь нужно контролировать свою ЧСС (частота сердечных сокращений). Формул подсчёта очень много, одна из самых точных — формула Карвонена. Тебе нужна программа, которая вычислит ЧСС для тренировок. Формула такая: ЧСС во время тренировки = (максимальная ЧСС — ЧСС в покое) * интенсивность + ЧСС в покое.
Максимальная ЧСС рассчитывается так: 220 – возраст. Возраст записан в переменную age.
ЧСС в покое хранится в переменной `pulseAtRest`, а интенсивность записана в процентах и хранится в переменной `intensity`.
Найди ЧСС для тренировки и запиши результат в переменную `pulseAtWorkout`. Результат округляй с помощью `Math.round()`. И не забудь перевести проценты интенсивности в дробное число

JS стартовый код

```
let age = 17;
let pulseAtRest = 60;
let intensity = 75;
```

Кубики куются на кухне

Программа тренировок разработана, но надо подумать и о питании. Вы же знаете, что здоровье и стройность зависят не только от тренировок? Нужно подсчитать норму КБЖУ (калории, белки, жиры, углеводы) для себя. Используем формулу Харриса-Бенедикта.
В первую очередь надо посчитать уровень метаболизма: 88.362 + (13.397 * вес в кг) + (4.799 * длина в см) — (5.677 * возраст в годах).
Вес в килограммах хранится в переменной weight, длина в сантиметрах записана в переменную length, возраст в годах находится в переменной age.
Норма калорий рассчитывается так: уровень метаболизма * коэффициент двигательной активности. Коэффициент записан в переменную `activityRate`.
Посчитай норму калорий и запиши в переменную `calorieRate`.
Калорийность складывается из белков, жиров и углеводов. В моей диете белки должны составлять 40% от всей калорийности. Найди их и запиши в переменную proteins.
Жиры запиши в переменную `fats`, они должны составлять 25% от всех калорий.
А углеводы запиши в переменную `carbohydrates`, они составляют 35%.
Каждое из значений (калории, белки, жиры и углеводы) округли с помощью `Math.round`

JS стартовый код

```
let weight = 75;
let height= 183;
let age = 25;
let activityRate = 1.725;
```