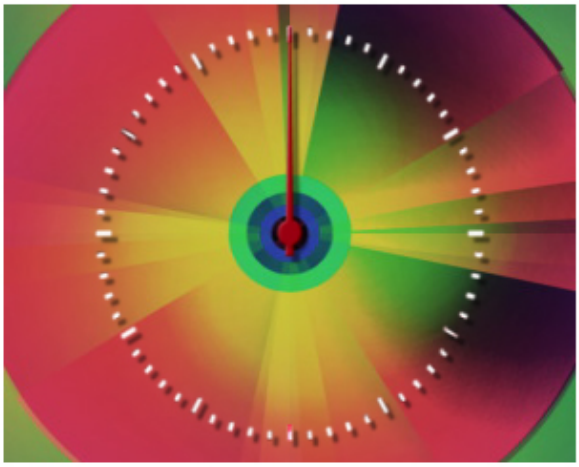
Создание выражений
в Adobe After Effects



Created by «Frosty» Alexander Lavrov

2003

В качестве вступления

Если позволите, несколько слов от автора. Это руководство не претендует на право называться всеобъемлющим и совершенным. Оно представляет собой попытку объединить множество материалов, посвящённых созданию выражений, в одном источнике, базирующемся на официальном руководстве от **Adobe**.

По этой причине, хочу привести основные источники материалов для этого руководства и поблагодарить их авторов за труд:

1. Adobe After Effects user guide
2. www.jjgifford.com
3. Энтони Боланте «Adobe After Effects 5.5 для windows и mac»
4. Adobe After Effects: Classroom in a book
5. creativecow.net
6. Справочник по языку JavaScript
7. aefreemart.com
8. adobe.com
9. Вся литература от CyberMotion

Отдельное спасибо **Digital Lion** его работу. Именно прочтение его статей по выражениям толкнуло на дальнейшее их изучение.

Перед тем, как вы начнёте читать этот импровизированный manual, хочу порекомендовать, чаще обращаться к главе 11 **«*Справочник по языку выражений Adobe After Effects*».** Поверьте, многие вопросы отпадут сами собой. Я очень постарался перевести его наиболее полно. А так же дополнил информацией по **JavaScript,** которая может вам пригодится.

Да и ещё, руководство писалось для пользователей самого разного уровня. Так что, если разбираетесь в выражениях, и первые главы вам кажутся совершенно ненужными и слишком простыми, то взгляните на главу с примерами выражений. Там есть достаточно интересные примеры, созданные профессионалами (см. creativecow.net и CyberMotion).

1. Использование выражений

Хотелось бы начать с небольшого введения в тему. На многих форумах, посвящённых **Adobe After Effects**, часто можно увидеть вопросы по выражениям. А для чего собственно нужны выражения? Ответ достаточно прост и банален: использование их часто упрощает работу пользователя **Adobe After Effects** и значительно сокращают время, затрачиваемое на создание сложных взаимодействий между слоями, эффектами посредством создание связи между их свойствами.

Можно привести такой пример, когда слой вращается и к нему применён эффект **Drop Shadow**. Мы можем связать значение свойства **rotation** со значением свойства **direction** отбрасываемой тени. Таким образом, нам удастся достичь того, что слой будет вращаться, и соответственно ему будет вращаться и отбрасываемая им тень. Мы получили этот эффект быстро и без применения ключевых кадров.

В версию **Production Bundle** входит инструмент **«Motion Math»**. Использование его готовых скриптов иногда бывает гораздо более простым и удобным, чем использование выражений. Например, для того чтобы слой масштабировался в зависимости от каких-либо параметров звукового файла, мы можем использовать скрипт **layeraud.mm**.

Язык выражений базируется на стандартном языке **JavaScript,** но, к счастью пользователь **AAE** не нуждается в совершенном знании **JavaScript** для написания выражений. Вместо этого мы можем создавать выражения, используя инструмент **«Pick wipe»**, создавая простые выражения и затем модифицируя их соответственно нашим дальнейшим потребностям. Для понимания значений операторов языка рекомендую

заглянуть в главу ***«Справочник по языку выражений Adobe After Effects».*** Если вы имеете базовое представление языка **JavaScript,** то вы можете создавать сложные связи между свойствами слоёв. Авторы официального руководства по AAE рекомендуют для лучшего понимания **JavaScript** почитать справочное руководство ***«JavaScript: The Definitive Guide»***, написанное Дэвидом Фланаганом. Но собственно это не критично, так как **AE** использует только ядро **JavaScript**, а не расширенную версию для web-браузеров. Вместо расширений для web, выражения **AE** содержат набор собственных встроенных объектов, таких как **Layer, Comp, Footage, и Camera** и т.д.

2. Создание выражений

Вся ваша работа по добавлению, редактированию и написанию выражений происходит в окне **Timeline**. Когда вы добавляете выражение к свойству слоя, то оно появляется по умолчанию в текстовом поле под свойством. Используйте это поле для того, что бы ввести новое выражение или редактировать уже написанное.

Если вы ввели в выражение название слоя; затем изменили название слоя на **Timeline**, не изменив в выражении старое название слоя на новое, то выражение не будет работать.

Пошаговое создание выражений

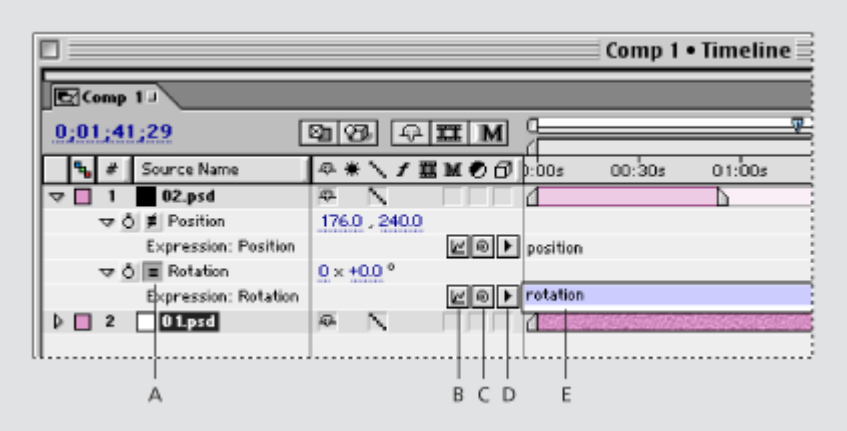
1. Выделите какое-то свойство слоя в окне **Timeline** и сделайте одну из следующих вещей (все они абсолютно равноценны и приведут к одному и тому же результату):
   - в главном меню **Choose Animation > Add Expression**
   - **Alt-click** (Windows) или **Option-click** (Mac OS) на **stopwatch** () свойства слоя
   - Для любителей горячих клавишей **Shift+Alt+Ins**

AE автоматически заполняет поле выражения, по умолчанию выражением которое копирует постоянное значение свойства или значение ключевых кадров, если они имеются.

2. Создайте выражение, делая одну из следующих вещей:
   - Перетащите **«Pick wipe»** () данного свойства к другому свойству этого или другого слоя в окне **Timeline**. Если вам понадобиться вы сможете редактировать полученный результат. Подробнее об этом в главе «Создание выражений при помощи инструмента *«Pick wipe»».*
   - Вы можете написать своё собственное выражение. См. главу ***«Написание собственных выражений».*** Сильно облегчает жизнь в этом плане меню написания выражений. Оно содержит большое количество элементов, которые вы можете ввести в ваше выражение, не забывая при этом соблюдать синтаксис языка. См. главу ***«Использование меню языка выражений».***
3. Кликните где-нибудь снаружи поля выражения или же нажмите **enter** дополнительной клавиатуре для того, что бы активизировать ваше выражение.

Если ваше выражение не может быть обработано, то AE выдаёт сообщение, описывающее ошибку и затем, автоматически отключает выражение. При этом на выражении появляется жёлтая иконка предупреждения (). Щёлкните по ней, если хотите видеть сообщение об ошибке снова.

Далее привожу небольшой путеводитель по основным кнопкам, имеющим отношение к выражениям.



A. Включение\выключение выражения.
B. Графическое отображение выражения.
C. Pick wipe
D. Меню элементов языка
E. Поле выражения

Переключение выражений (on/off)

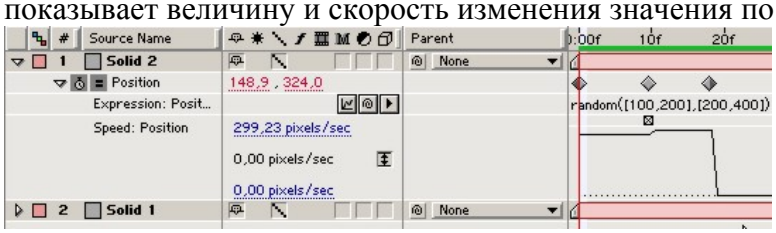
Вы можете переключать эти режимы в любое время. Когда выражение включено, то переключатель имеет вид (). В выключенном состоянии ().

Изменение размера поля выражения

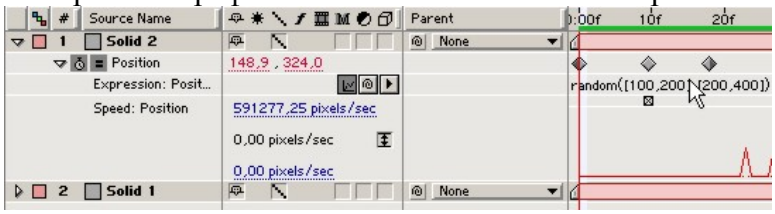
Ваши выражения могут содержать большое количество строк. Если необходимо, вы можете увеличивать или уменьшать размер поля выражений. Для этого вы должны поместить курсор на нижний край поля. Он примет вид двухсторонней вертикальной стрелки (). Теперь тащите курсор вверх или вниз.

Просмотр графического отображения выражения

Для того, что бы увидеть, как изменяется значение или скорость в выражении в виде графика, кликните на иконке графического отображения выражения (). Чёрный график показывает величину и скорость изменения значения по ключевым кадрам.



Красный график изменение величины и скорости в выражении.



Просмотр всех выражений в композиции

Для просмотра всех выражений выделенных слоёв нажмите ***EE*** на клавиатуре.

Конвертирование выражений в ключевые кадры

В некоторых ситуациях может быть полезно конвертирование выражений в ключевые кадры. Например, если вы хотите заморозить значение в выражении, вы можете конвертировать выражение в ключевые кадры и затем корректировать ключевые кадры соответственно своему желанию. Конвертирование так же может быть полезным,

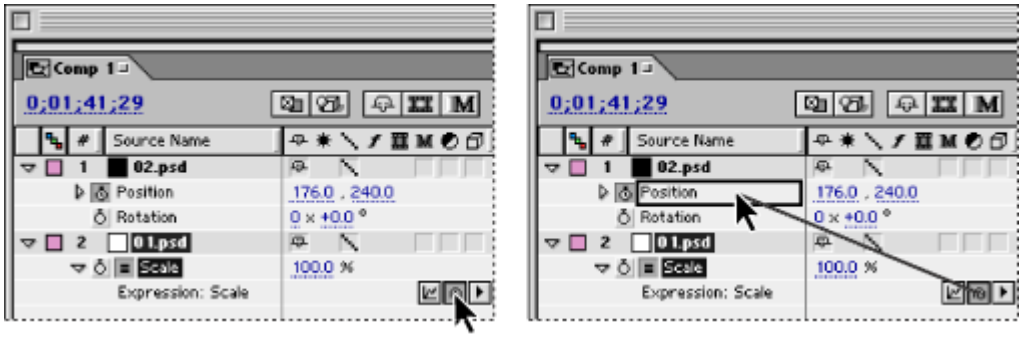
когда вы написали сложное для рендеринга выражение. Конвертируйте его в ключевые кадры и время рендеринга сократиться. Когда выражение конвертируется в ключевые кадры, AE создаёт ключевые кадры каждый кадр и затем отключает выражение. Для преобразования выражений в ключевые кадры необходимо проделать следующую последовательность действий: в главном меню **Animation > Keyframe Assistant > Convert Expression to Keyframes**.

3. Создание выражений при помощи инструмента *«Pick wipe»*

Инструмент **«Pick wipe»** используется при создании выражений для связывания одного свойства или эффекта с другим. Например, если мы свяжем значение свойства **rotation** слоя 1 со свойством **rotation** слоя 2, то значение **rotation** слоя 1 будет равно значению **rotation** слоя 2, в любой момент времени. Или же такой пример – нам нужно сделать так, что бы камера следила за каким-то слоем. Для этого мы свяжем свойство **Point of Interest** камеры со свойством **position** слоя, за которым должна следить камера.

Выбор свойств и значений при помощи «Pick wipe»

Для того, что бы использовать **«Pick wipe»**, кликните на иконке (@) и перетащите её на другое свойство или же на одно из значений свойства. Когда вы выбираете значение или свойство, то AE автоматически вставляет соответствующее выражение в поле выражения. Если текст в поле уже выделен, то он заменяется текстом нового выражения. Если курсор не находиться в поле выражения, то всё выражение заменяется новым.



Значения, которые используются в выражениях

Различные свойства в AE имеют различное число измерений (**1D, 2D, 3D,** или **4D**). Это зависит от того, какое количество значений имеет свойство. В языке выражений, значение свойств может быть простым числом и массивом. Массивы, такие, как, например, значения свойства **position**, всегда заключены в квадратные скобки и разделены запятыми. Простые значения, например opacity, в скобки не заключаются.

Примеры значений, в зависимости от измерений:

1D значения: **Rotation**(°) и **Opacity**(%)

2D значения: **Scale**[x=width, y=height], **Position**[x, y], и **Anchor Point**[x, y]

3D значения: **Scale**[width, height, depth], **Position**[x, y, z], и **Anchor Point**[x, y, z]

4D значения: **Color**[red, green, blue, alpha]

Пример использования «Pick wipe»

Следующий пример показывает, как использовать **«Pick wipe»** для имитации светового пятна, проявляющегося во время вращения подлежащего прямоугольного слоя.

Для того, чтобы понять значение текста в данном выражении, необходимо ознакомиться с главой *«Понимание языка выражений».*

Откройте файл **Dial.aep,** что бы понять как будет выглядеть конечная композиция. Теперь закройте его.

Сделаем следующие шаги:

1. Создайте композицию, содержащую два слоя:
   - создайте верхний слой круглым и назовите его *«Light»*
   - нижний слой пусть будет прямоугольным, дайте ему название *«Dial»*
2. Создайте ключевые кадры для анимации вращения слоя *«Dial».* Пусть он совершит вращение на 360^0.
3. Выберете свойство opacity слоя *«Light».* Далее в главном меню **Animation > Add Expression**. По умолчанию выражение появиться под свойством.
4. Кликнете иконку **«Pick wipe»** в свойстве **opacity** слоя *«Light»* и перетащите её на свойство **rotation** слоя *«Dial».* After Effects автоматически заполнит поле выражений следующим текстом **this_comp.layer("Dial").rotation**. Нажмите **Enter** на дополнительной клавиатуре или кликнете снаружи поля выражений для активизации вашего выражения.
5. Просмотрите полученную анимацию. Обратите внимание, что свойство **opacity** анимировано, но не имеет ключевых кадров.

Редактирование выражений, созданных при помощи «Pick wipe»

Я думаю, все обратили внимание на то, что анимация получилась не совсем такой, как нам хотелось. Это получилось потому, что различные свойства имеют различные единицы измерения и как следствие, различную размерность. Свойство **opacity** варьирует от 0% до 100%, а свойство **rotation** от 0^0 до 360^0. То есть, у нас получилась такая ситуация, что на 1 единицу opacity приходится 3,6 единиц rotation. И как только свойство **rotation** достигает значения 100, то **opacity** возрастать уже некуда, и мы получаем, что прямоугольник продолжает вращаться, а круг уже имеет полную непрозрачность. Для борьбы с этим, мы должны сравнять единицы измерения свойств. Несколько редактировав наше уже имеющееся выражение, мы получим нужный нам результат. В выражениях мы можем использовать простейшие математические операторы. Такие как: + (сложение), - (вычитание), * (умножение), / (деление), *-1 (смена значения на противоположное). Добавим в наше выражение следующее: деление на 360 и умножение на 100. Выражение примет вид **this_comp.layer("Dial").rotation/360*100.**

Если хотите посмотреть, как это получилось у меня, откройте файл **DialEdit.AEP**

Как видите, теперь **opacity** достигает своего максимума одновременно с **rotation**. Таким приёмом вам придётся пользоваться очень часто, причём ваши выражения будут гораздо сложнее этого простейшего примера.

Примеры редактирования выражений, созданных при помощи «Pick wipe»

А теперь ещё несколько примеров редактирования выражений. Эти выражения легки в создании, а в результате мы получаем сложно выглядящий эффект, для создания которого потребовалось бы создание большого количества ключевых кадров.

Пример 1 . Анимация часов

Для начала откройте файл **CLOCK.AEP** и посмотрите, что вы должны получить.

Ну а теперь приступим.

1. Создайте 3 слоя
   - минутная стрелка *(«MinuteHand»)*
   - часовая стрелка *(«HourHand»)*
   - циферблат *(«ClockFace»)*
2. Установите **anchor points** стрелок в центре часов.

3. Создайте анимацию для часовой стрелки. Затем свяжите при помощи **«Pick wipe»: rotation** минутной стрелки с полученной анимацией **rotation** часовой стрелки. Под свойством **rotation** минутной стрелки появится следующее выражение **this_comp.layer("HourHand").rotation**.

4. Мы получили минутную стрелку следующую за часовой.

5. Сделаем теперь так, что бы скорость минутной стрелки была в 12 раз больше, чем часовой. Для этого умножим свойство rotation минутной стрелки на 12. Выражение примет следующий вид **this_comp.layer("HourHand").rotation*12**.

Пример 2. Управление размытием слоя, путём связывания его со значением позиции по оси x.

Откройте файл **BLUR.AEP.** В дальнейшем, во всех примерах, я буду указывать в начале упражнений название файла с конечным результатом без лишних слов.

1. Создайте слой, имеющий меньший размер, чем композиция. Назовите его *«BlurSolid».*

2. Добавьте слою эффект **Fast Blur**.

3. Задайте анимацию **Position** по **оси x**. Пусть слой двигается из правого края композиции в левый.

4. Теперь свяжите свойство **Blurriness** эффекта **Fast Blur** со свойством **x-position.** Получите простейшее выражение следующего вида **position[0]**.

5. Размытие получилось слишком сильным. Для того, что бы его уменьшить, разделим значение **position** в выражении на 10. Получим **position[0]/10**.

Пример 3. Слои увеличивающиеся друг за другом.

Откройте файл **ScaleBoxes.aep**

1. Создайте 3 разноцветных, не перекрывающихся слоя, размером меньше композиции. Назовите их *«Red», «Green», «Blue».*

2. Создайте анимацию для слоя *«Red»*. Пусть он увеличивается от 0 до 100% с 0 до 2 секунды.

3. Выберите свойство **scale** слоя *«Green»*, и при помощи **«Pick wipe»** свяжите его с анимированным **scale** слоя *«Red»*. Появиться выражение **this_comp.layer("Red").scale**. Для того, что бы добиться задержки появления следующего слоя на 1 секунду вам придётся редактировать выражение. Добавьте в него **value_at_time(time-1).** В меню языка выражений: **property > value_at_time(t).** Вместо **t** подставьте **time-1.** Это будет означать задержку начала увеличения слоя *«Green»* на 1 секунду. У вас получиться выражение следующего вида **this_comp.layer("Red").scale.value_at_time(time-1).** Подробно о значении операторов и методов смотрите в главе *«Справочник по языку выражений в Adobe After Effects».*

4. Свяжите точно так же scale слоя *«Blue»* со scale слоя *«Red»*.

Но сделайте задержку в 2 секунды.

Получите выражение **this_comp.layer("Red").scale.value_at_time(time-2).**

4. Написание ваших собственных выражений

After Effects использует **JavaScript 1.2** для создания выражений. **JavaScript** поддерживает все необходимые операторы и функции для создания выражений. Для написания ваших собственных выражений, вам, несомненно, может помочь знание синтаксиса и функций **JavaScript.** Но вы сможете писать достаточно сложные выражения и без знания **JavaScript.**

Написание выражений происходит в поле выражений или в любом текстовом редакторе, с последующим копированием в поле выражения. Для быстрого построения

выражения из отдельных его элементов, с присущим им синтаксисом, используйте меню языка выражений AE. Смотрите главу ***«Использование меню языка выражений».***

В **JavaScript** какое-то определённое значение объекта называется *«свойством».* В AE термин *«property»,* принятый в **JavaScript**, является методом (служат для задания данных) или атрибутом (являются поименованными значениями объекта).

Когда пишите выражения, соблюдайте следующие руководящие принципы:

- **JavaScript** учитывает регистр, то есть, например *«Red»* и *«red»* это разные слова

- Для разделения частей выражения или строк необходимо ставить точку с запятой

- Пробелы между словами, табуляции, разрывы строк игнорируются, кроме нескольких случаев:

1. Когда пробел является частью строкового выражения (например, объект **comp(“final comp”).**
2. Случайная вставка пробела в значение числа или строки будет являться ошибкой синтаксиса

Сохранение выражений

Однажды, написав выражение, вы можете сохранить его для дальнейшего использования. Для этого скопируйте его из поля выражений и вставьте в любой текстовый редактор. Например, такой как **Notepad** или **Simple Text**.

Однако, когда вы захотите использовать ваше сохранённое выражение в другом проекте, то столкнётесь с тем, что вам придётся менять названия слоёв в вашем выражении. Это связано с тем, что выражения привязаны к конкретным объектам, названия которых прописаны в их тексте.

Если вы захотите сохранить выражение для последующего использования его в другом проекте, то вы можете добавить к нему комментарий, чтобы потом было проще разобраться в выражении.

Для добавления комментариев к выражению используйте любой из приведённых ниже способов:

- Напечатайте // в начале комментария. Любой текст после // игнорируется аппаратом выражений AE и является всего лишь комментарием.

Пример:

// This is a comment.

- Напечатайте **/*** в начале и ***/** конце комментария. Любой текст между **/*** и ***/** игнорируется аппаратом выражений AE и является всего лишь комментарием.

Пример:

/* This is a multiline comment */

5. Понимание языка выражений

Как уже было сказано ранее, After Effects использует язык **JavaScript 1.2** дополненный его собственным набором объектов. Все выражения начинаются с глобальных объектов. Глобальный объект для любого выражения – это слой, на котором оно написано. Например, вы добавили выражение к свойству **scale** слоя под названием *«Solid1»* и вы хотите связать **scale** с **position** этого же слоя. Вы можете использовать любое из приведённых ниже выражений, так как они эквивалентны друг другу:

this_comp.layer("Solid1").position

this_layer.position

position

Для извлечения значения из объекта, когда слой не включает выражение, приходится вставлять объект в выражение. Например, если вы написали выражение на слое под названием *«Solid1»,* и вы хотите извлечь значение **position** из слоя *«Solid2»*, используйте следующее выражение:

this_comp.layer("Solid2").position

Доступ к элементам в выражениях

Для доступа к значению используйте цепь элементов, разделённых точкой. Для понимания порядка, в котором вы должны добавлять элементы к выражению, прочитайте главу *«Справочник по языку выражений Adobe After Effects».* Рассмотрим следующую процедуру, как пример конструирования выражений с помощью меню языка выражений.

Конструирование простого выражения:

1. Создайте два слоя

2. Выделите свойство **position** первого слоя композиции и затем **Animation > Add Expression.** Следующее выражение появиться по умолчанию: **position**.

3. Напечатайте вместо слова **position**:

this_comp

4. Элемент **this_comp** – это глобальный атрибут, который имеет значение объекта, отображающего данную композицию.

5. Для понимания, что может следовать за **this_comp** в вашем выражении, смотрите тему *«Глобальные атрибуты и методы».*

6. Заметьте, **this_comp** ведёт в данную композицию. Чтобы выбрать слой, который мы будем использовать, воспользуемся опцией *"layer(index)"*. В круглых скобках указывается название в кавычках или номер слоя. Для извлечения значения из второго слоя в активной композиции напечатайте следующее:

this_comp.layer(2)

7. Снова посмотрите в *«Справочник по языку выражений».* Найдите *«Атрибуты и методы слоя»,* что бы понять какие элементы вы хотите использовать. Например, если вы хотите получить доступ к значению свойства **position** второго слоя активной композиции, напечатайте следующее: **this_comp.layer(2).position**

8. И опять посмотрите в *«Справочник по языку выражений».* Найдите в нём *«Атрибуты и методы свойства»* и заметьте, что вы можете добавить временной фактор к выражению. Для добавления специфического времени, такого как *«данное время минус 2 секунды»* напечатайте следующее:

this_comp.layer(2).position.value_at_time(time-2)

9. Из *«Атрибуты и методы свойства»* отметьте, что временной фактор **value_at_time(time-2)** является числом. Когда элемент является числом, массивом или булевским выражением вы не можете добавить другие атрибуты или методы к выражению. Однако, если вы хотите, вы можете добавить простейшие математические операторы **(+,-,/,*).**

10. Кликните снаружи выражения, чтобы активировать его.

Переменные в выражениях

Иногда очень удобно использовать переменные в выражениях. Для примера, в следующих выражениях результат одинаков:

x = rotation * 10; [x, 20]

[rotation * 10, 20]

В данном случае мы ввели переменную величину **x**, которая равна **rotation*10.** Таким образом, мы получили менее загроможденное конечное выражение **[x, 20].** Такой

приём приходится довольно часто использовать, когда вы пишете большое выражение со сложной структурой.

Время в выражениях

Время в выражениях всегда исчисляется в секундах. По умолчанию время для любого выражения – это время данной композиции, в котором настоящее выражение вычисляется. Следующие выражения оба используют по умолчанию время композиции и имеют одинаковое значение:

this_comp.layer(1).position

this_comp.layer(1).position.value_at_time(time)

Для использования относительного времени, добавьте временное значение к аргументу «**(time)**». Например, для получения значения времени «на 5 секунд раньше данного времени», используйте следующее выражение:

this_comp.layer(1).position.value_at_time(time-5)

Время выражения во вложенных композициях

Во вложенных композициях, по умолчанию, не используется **remapped time**. Однако, если вы используете функцию **"*source*"**, то вы можете использовать **remapped time** во вложенных композициях.

Например, если исходник слоя в родительской композиции – это вложенная композиция, и в родительской композиции вы имеете **remapped time**, тогда вы можете получить доступ к значению **position** слоя во вложенной композиции при помощи следующего выражения:

comp("nested comp").layer(1).position

Свойство **position,** в данном выражении, будет использовать по умолчанию время композиции.

Однако если вы получите доступ к **layer 1**, используя функцию "***source***", то свойство **position** сможет использовать **remapped time.** Вы получите следующее выражение: **source.layer(1).position**

Числа и массивы

Массивы представляют многомерные значения в выражениях. Значение каждого свойства в выражении это одиночное число или булевское значения (для такого свойства, например как, **opacity)**, или же массив для двух-, трёх- и четырёхмерных значений (например, **position** или ***"bg_color"***). Большое количество значений в AE (например, такие как **position, scale,** и **color**) имеют больше, чем один параметр значения. Такие значения представляются массивами, имеющими разное количество измерений. Например, свойство **position** может быть представлено как **2d**, и как **3d** массив, потому что оно может иметь **2 [*x, y*]** или **3 [*x, y, z*]** значения.

Члены массива всегда разделяются запятыми. Для примера, выражение **[10, 20]** имеет набор свойств, состоящий из значения **X=10** и значения **Y=20**, которые разделены запятой.

Индексирование в векторах и массивах

Индексирование это путь нахождения значений в векторах и массивах. Вектор или массив начинает индексацию с нуля. Для примера:

- **Position[0]** это координата **x** позиции
- **Position[1]** это координата **y** позиции
- **Position[2]** это координата **z** позиции

AE-специфические элементы (такие как **Layer**, **Effect** и **Mask)** используют процесс индексирования, который начинается с единицы. Например, первый слой в **Timeline window** будет в выражении выглядеть так: **layer(1)**

Обычно лучше использовать имя слоя, эффекта или маски**,** чем номер. Это позволить избежать вам неразберихи и ошибок, в случае перемещения слоя, эффекта или маски. Когда вы используете имя, оно должно быть заключено в кавычки. Для примера рассмотрим нижеприведённые выражения:

effect("Colorama").param("Get Phase From")

effect(1).param(5)

На самом деле это одно и тоже выражение. Но в первом случае мы использовали имена, для обозначения эффекта и его свойства. Во втором использовали порядковые номера. Явно, что первое выражение гораздо более понятно нам. К тому же, если даже мы переставим эффекты местами в дальнейшем, это выражение будет работать как должно, так как эффект и его свойство имеют уникальные для данной композиции имена.

Доступ к значениям внутри векторов и массивов

Вы можете создавать выражения, которые связывают только одно значение внутри массива **2d** или **3d** свойства. По умолчанию, используется первое значение, пока вы не указали другое. Например, вы перетаскиваете **pick whip** из свойства **rotation** слоя 1 к свойству scale слоя 2. У вас появиться следующее выражение:

this_comp.layer(2).scale[0]

Как вы видите, по умолчанию используется первое значение свойства **scale**, то есть ширину. Если вы предпочитаете использовать высоту, то перетащите **pick whip** прямо на второе значение или измените выражение на это:

this_comp.layer(2).scale[1]

И наоборот, если вы перетащите **pick whip** со свойства **scale** слоя 2 к свойству **rotation** слоя 1, то AE автоматически свяжет оба значения свойства **scale** со значением свойства **rotation.** Получим такое выражение:

[this_comp.layer(1).rotation, this_comp.layer(1).rotation]

При желании, мы можем связать только одно значение **scale** со свойством **rotation**. Второе значение можно сделать постоянным. Следующее выражение демонстрирует это:

[this_comp.layer(1).rotation, 10]

В нём значение ширины связано со свойством **rotation,** а значение высоты постоянно и равно 10.

6. Использование меню языка выражений

Меню языка выражений содержит в себе все специфические языковые элементы, которые вы можете использовать в выражениях. Это меню очень полезно для работы с элементами и помогает придерживаться правил синтаксиса.

Вы можете выбрать любой элемент, атрибут или метод в меню и AE автоматически вставит его в поле выражений после курсора. После этого вы можете редактировать его и добавлять к нему то, что считаете нужным. Например, если вы хотите начать выражение с композиции, как объекта, то выберете ***"this_comp"*** в меню глобальных объектов (**Global menu**):

"this_comp"

Для продолжения выражения добавьте период (**.**) в конец. Затем выберите атрибут из **Comp menu**, такой как ***"layer"***:

this_comp.layer(

Теперь вставьте название слоя:

this_comp.layer(1)

Затем выберите атрибут или метод слоя из **Layer**, **Light**, или **Camera menu**. Например, если слой 1 имеет ключевые кадры в значении **position**, то выберите ***"position"***:

this_comp.layer(1).position

Вставка элементов языка выражений

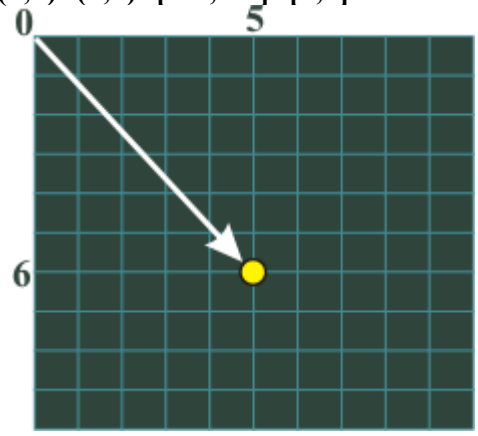
1. Выберите элемент из меню языка выражений. Элемент появится в поле выражений после курсора.
2. Редактируйте и вставляйте ещё элементы, если нужно.

7. Основы геометрии

Понятие о геометрии и векторах

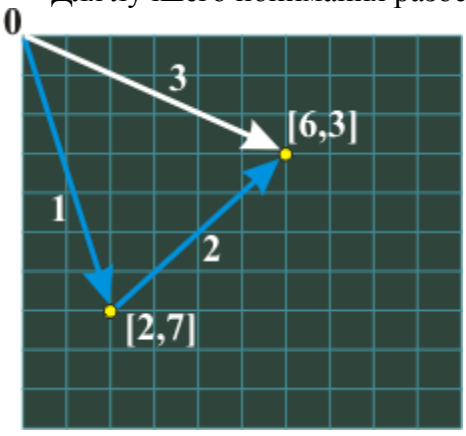
Я думаю, вы достаточно освоились с алгеброй в AE. Теперь пришло время осваивать геометрию. Она откроет перед вами ещё большие возможности по использованию выражений в ваших проектах.

Основой векторной математики является сложение и вычитание векторов. Как вы помните, мы оперировали термином *«вектор»,* когда речь шла о массивах, и тогда ни о какой геометрии мы не упоминали. Разберёмся в этой ситуации. На рисунке вы видите плоскость, на которой находиться точка с координатами **(5,6).** Из точки с координатами **(0,0)** в точку с координатами **(5,6)** направлен вектор. Его длину мы можем записать как **(5,6)- (0,0)=[5-0,6-0]=[5,6]**



Сложение векторов

Для лучшего понимания разберём эту тему на следующем примере.



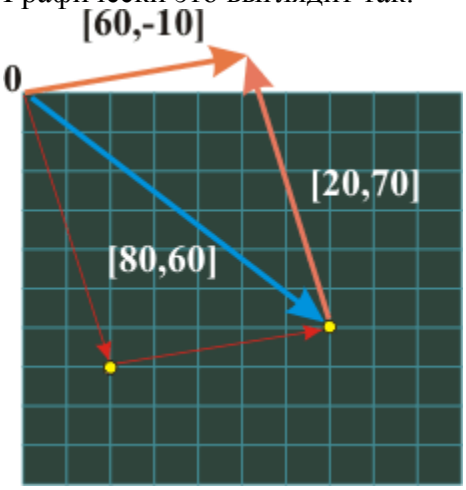
Первоначально имеется два вектора: вектор1 и вектор2 с координатами **[(0,0),(2,7)]** и **[(2,7), (6,3)]** соответственно. Для сложение *вектора1* с *вектором2* проведем линию из начала *вектора1* в конец *вектора2*.

Как видно из этого рисунка компоненты вектора образуют прямоугольный треугольник. Отсюда выход на теорему Пифагора, которую в дальнейшем мы будем использовать.

Вычитание векторов

Математически вычитание векторов есть вычитание их компонент. Например, **[80, 60] - [20, 70] = [60, -10]**

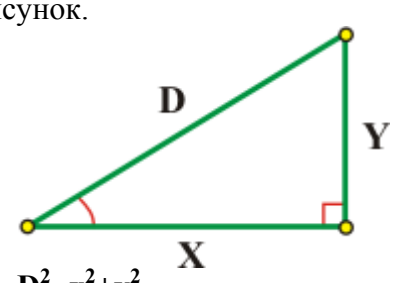
Графически это выглядит так.



Вычисление длины вектора

Теперь немного практики. Попытаемся при помощи теоремы Пифагора и функции **length()** определить длину векторов.

Если помните, квадрат гипотенузы равен сумме квадратов катетов. Взгляните на рисунок.

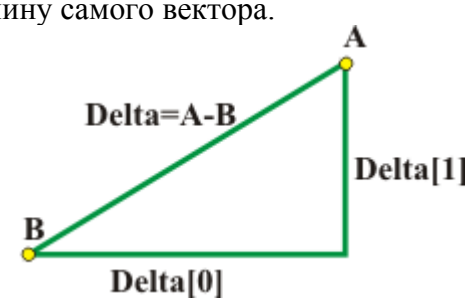


$D^2=x^2+y^2$

Если записать это же, но языком выражений, то получим **D=math.sqrt(x*x+y*y)**

Компоненты любого двухмерного вектора являются катетами, а сам вектор гипотенузой. Воспользуемся этим, чтобы найти длину вектора по его компонентам.

У нас имеется некий вектор *Delta*. Длина его компонент известна. Требуется найти длину самого вектора.



Вычисление с помощью теоремы Пифагора

1. Определение точек, расстояние между которыми будем вычислять

Для этого введём переменные *point1* и *point2*

point1=this_comp.layer("A").position;
point2=this_comp.layer("B").position;

2. Определение компонент получившегося вектора

delta=sub(point1, point2);
X=delta[0];
Y=delta[1];

3. Вычисление длины вектора

distance=Math.sqrt(X*X+Y*Y);

Этот способ достаточно простой, но попробуйте сделать те же вычисления для вектора в 3d пространстве. Получиться громоздкое выражение. Поэтому более правильно и удобно использовать метод **length(vector).**

Метод length(vector)

В **Adobe After Effects** имеется встроенный метод определения длины любого вектора: **length().** Укажите методу вектор любого размера и вы получите его длину, выраженную положительным числом.

Используя метод length(), подправим уже имеющееся выражение:

1. Определение точек, расстояние между которыми надо вычислить

point1=this_comp.layer("A").position;
point2=this_comp.layer("B").position;

2. Определение компонентов получившегося вектора

delta=sub(point1, point2);

3. Вычисление длины вектора

length(delta);

Пример на вычисление длины вектора

Пример называется *«интерактивное размытие»,* и был взят с сайта www.jjgifford.com. После небольшой модификации я привожу его ниже.

Откройте файл **VectorBlur.aep** из папки **Interactive**

Исходно имеется композиция, содержащая в себе 6 слоёв: 5 из них *(“a”, ”b”, ”c”, ”d”, ”e”)* будут подвергаться трансформациям, и один управляющий слой *(“Drag me”).*

Наша задача сделать так, чтобы при приближении управляющего слоя к любому из остальных слоёв, этот слой увеличивался и размывался.

1. Подключим выражение для свойства **scale** слоя *«a»*.
2. Зададим переменные для позиции текущего слоя и корректировочного

point1=this_layer.position;
point2=this_comp.layer("Drag Me").position;

3. Теперь нам нужно найти вектор между этими двумя точками

delta=sub(point1, point2);

4. Следующим шагом будет нахождение длины полученного вектора

distance=length(delta);

5. Транслируем значения в значения масштаба

linear(distance, 0, 80, [250,250], [75,75]);

6. Для размытия повторите это же выражение за исключением **linear(distance, 0, 80, [250,250], [75,75]);** В данном случае выражение будет выглядеть так

linear(distance, 0, 80, 10, 0);

7. Повторите все эти пункты для остальных слоёв

Тригонометрия

Как уже ранее говорилось, вектор и его компоненты представляют собой прямоугольный треугольник. Это открывает для нас возможность пользоваться тригонометрическими функциями при нашей работе с векторами, так как тригонометрические величины это соотношение сторон треугольника.

синус угла = длина противолежащего катета/длина гипотенузы
косинус угла = длина прилежащего катета/длина гипотенузы
тангенс угла = длина противолежащего катета/длина прилежащего катета

В языке **JavaScript** существуют специальные методы для вызова тригонометрических функций:

Math.sin(angle)
Math.cos(angle)
Math.tan(angle)

Эти методы возвращают значение в виде скаляра. Если это синус/косинус - возвращаемое значение будет лежать в пределах от -1 до 1. В случае с тангенсом, возвращаемое значение может быть любым.

Понятие о радианах

Все тригонометрические методы работают с радианами. Радиана это единица измерения, равная 57^0.

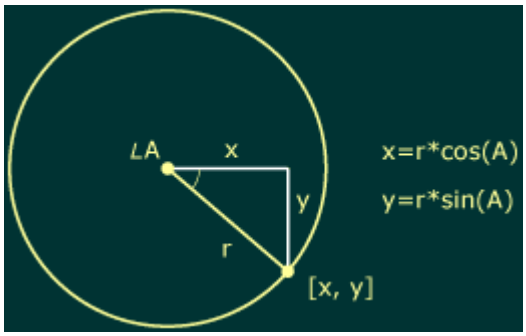
Когда вы будете работать с тригонометрическими методами, то столкнётесь с тем, что в AE всё кроме тригонометрии поддерживает градусы, а не радианы. Вы можете, конечно, конвертировать радианы в градусы и наоборот вручную, но к счастью, мудрые разработчики AE включили в пакет методы конвертации:

radians_to_degrees(radians)
degrees_to_radians(angle)
Пример: **Math.sin(degrees_to_radians(rotation))**

Циркулярные функции

Данные функции имеют прямое отношение к описанию движения по окружности. В этой теме мы рассмотрим связь между циркулярными и тригонометрическими функциями. Взгляните на следующий рисунок, и вы сами обратите внимание на связь между меняющейся гипотенузой и движением по окружности (циркулярной функцией).

Перпендикулярные стороны (катеты) прямоугольника соответствуют X и Y координаты точек, составляющих окружность. Найти эти координаты - значит найти местоположение точек, составляющих окружность. Сделать это, можно учитывая, что координаты соответствуют длинам катетов получающегося треугольника. Длины катетов можно найти с помощью функций синуса и косинуса угла в центре окружности. Посмотрите на следующий рисунок, и вы все поймете сами:



Примеры

1. Движение по окружности

Откройте файл **Trig Circular Motion.aep** из папки **Circular**

В этом примере мы зададим вращение по окружности какого-то объекта. Отметьте, что при вращении объект не будет менять ориентацию, не будет поворачиваться и вокруг своей оси. Если вы захотите, то сможете добавить и эту возможность.

1. Необходимо определить центр нашей окружности:
center=this_comp.layer("Center").position;

2. Теперь определяем длину радиуса. Задаём её в пикселах:
radius= 120;

3. Теперь определимся с углом. Мы используем как управляющую переменную, *«время»*. Именно время будет определять угол A - угол в центре окружности. Кроме того, всегда можно будет сделать вращение быстрее, "умножив время", скажем на 2: **time*2.** В таком случае полный оборот будет осуществляться за Pi секунд, так как полная окружность равна 2Pi радиан, а углы измеряются именно в радианах. **angle=time*2**;

4. Теперь напишем основу нашего выражения:

x=radius*Math.cos(angle);
y=radius*Math.cos(angle);

5. Если Вы все же хотите, что бы вращение происходило именно вокруг нашего центра, то нужно изменить центр координат, точнее говоря сдвигать каждую точку, составляющую окружность. И сделать это можно с помощью следующего оператора: **add(center, [x,y]);**

2. Размещение объектов по окружности.
Откройте файл **DUBLICATE.AEP**

Для получения этого эффекта мы не будем создавать новое выражение, а лишь модифицируем выражение из предыдущего примера.

В уже рассмотренном нами выражении входной переменной было время. Время шло, заставляя меняться угол, и объект двигался по окружности. В новом же примере, в входной переменной будет индекс слоя, его положение в стеке окна **Timeline**. Так как индекс от слоя к слою возрастает, то и угол будет меняться, возрастать. Надублируйте слои, и они сами расположатся по окружности.

1. Вот вам основная идея выражения:
layer_num=index-1;
interval=30;
angle=degrees_to_radians(layer_num*interval);

Единица вычитается из индекса, потому что считать надо с нуля. То есть, первый угол будет равен 0°.

От переменной *«Interval»* зависит, насколько далеко друг от друга будут находиться копии. Умножая **layer_num** на **interval**, мы преобразуем серию номеров слоев {0, 1, 2...} в серию углов {0, 30, 60...}. Далее преобразуем углы в радианы, используя метод **degrees_to_radians().**Это необходимо сделать, потому что тригонометрические функции в Javascript задаются углами, выраженными только в радианах.

Итак, первый угол разместится под углом в 0°, следующий - 30°, далее - 90° и т.д.

2. После того, как вы создадите столько копий, сколько надо, лучше всего будет с помощью превратить выражения в последовательности ключевых точек: **Animation/Convert Expression to Keyframes**. Теперь у вас не будет нужды каждый раз при просчете включать режим игнорирования выражений.

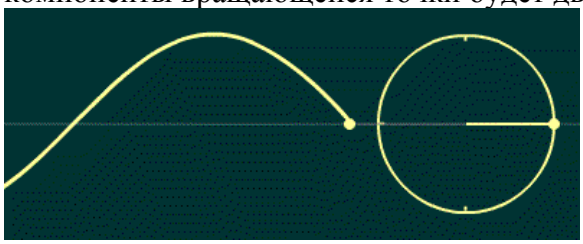
3. Вот полное выражение, привязанное к параметру местоположения (Position):
center=this_comp.layer("Center").position;
radius=80;
interval=30;
layer_num=index-1;
angle=degrees_to_radians(layer_num*interval);
x=radius*Math.cos(angle);
y=radius*Math.sin(angle);
add(center, [x, y]);

Гармоническое движение

В теме *«Циркулярные функции»* мы создавали движение по окружности, используя функции синуса и косинуса для нахождения x- и y-координат точек, которые составляют окружность. В этой теме мы с помощью функции синуса (или косинуса) будем извлекать, например, вертикальную составляющую объекта, движущегося по окружности.

В следующей анимации точка, движущаяся по окружности, будет управлять «пером» виртуального плоттера. Мы извлечём вертикальную компоненту вращающейся

точки, абсолютно игнорируя горизонтальную. И именно в зависимости вертикальной компоненты вращающейся точки будет двигаться «перо».



Как вы, наверное, обратили внимание, «перо» движется не в постоянном ритме. В минимальных и максимальных значениях перо замедляет свой ход так, как бы это происходило при применении функций интерполяции **«Easy-ease»**. Такой род движения называется простым гармоническим движением. Наиболее часто используемый пример такого движения - движение качающегося маятника.

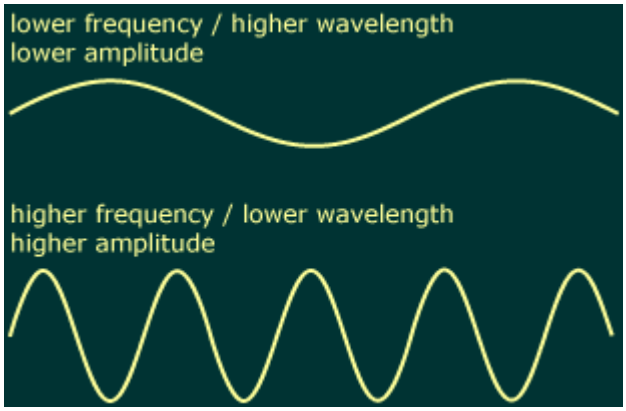
Распознать такой тип движения в следующем примере легко, обратив внимание на жёлтые точки, одна из которых двигается горизонтально, другая - вертикально. Эти две точки - демонстрируют соответственно горизонтальную и вертикальную составляющие движения. Двигаясь по синусоиде, они демонстрируют простое гармоническое движение:

Еще один пример простого гармонического движения – это движение объекта на пружине. Тут надо учитывать, что в реальном мире действие пружины постепенно убывает, уменьшается амплитуда движения пружины. Об амплитуде и пойдёт дальше речь.

Длина волны, частот и амплитуда

Любое волнообразное движение, в том числе и простое гармоническое движение, характеризуются двумя базовыми характеристиками: длиной волны и амплитудой. Длина волны, это как бы скорость колебаний, амплитуда, это как бы сила колебаний. Длина волны определяется, как расстояние между двумя минимумами или максимумами функции. Амплитуда - это расстояние от нулевой линии до минимума или максимума функции.

Под частотой понимается количество минимумов (максимумов) за определенный интервал времени, чаще всего за секунду.

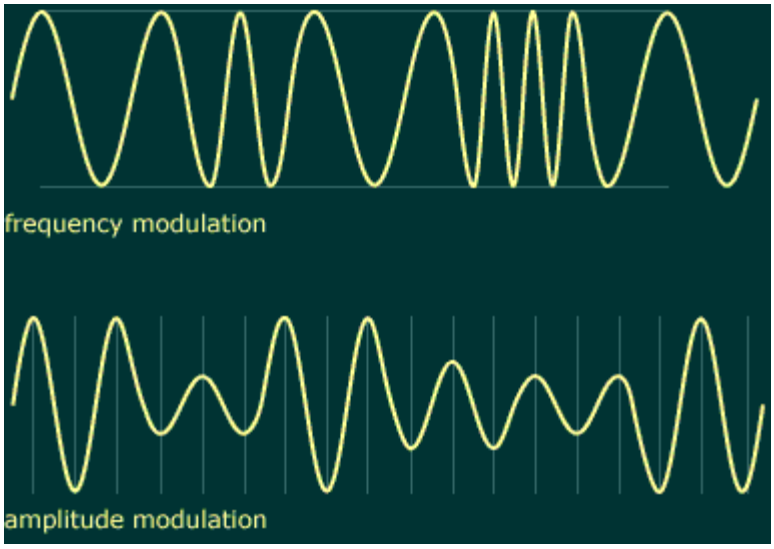


При необходимости, мы можем увеличить частоту тригонометрической функции. Для этого введём множитель к аргументу функции. Теперь функция будет меняться быстрее - увеличится частота. К примеру, функция **Math.sin(time*6)** будет меняться в 6 раз чаще, чем функция **Math.sin(time).**

Амплитуду так же можно увеличить, умножив на определенный множитель результат функции. К примеру, амплитуда функции **5*Math.sin(time)** будет в пять раз больше, чем амплитуда функции **Math.sine(time),** так как функция теперь меняется в пределах от –5 до +5, в то время, как раньше она менялась в пределах от –1 до +1.

amplitude*Math.sin(angle*frequency)

Указанные множители не обязательно должны быть константами - они могут меняться и быть даже теми же тригонометрическими функциями. Ниже вы можете увидеть рисунки к примерам, когда множитель при частоте (длине волны) - тригонометрическая функция, и далее, когда множитель амплитуды - тригонометрическая функция. Такие варианты называются модуляциями. Частотной и амплитудной модуляциями соответственно. А бывает и модуляция сразу и частотная и амплитудная.



Пример. Затухающие колебания пружины

Откройте файл **Spring_d.aep** из папки **Spring**

В предыдущей теме, когда мы изучали принципы простого гармонического колебания, мы создавали анимацию колеблющейся пружины. Тогда был упомянут факт, что в реальном мире колебания пружины постепенно затихают. Теперь изучив особенности затухания колебания, как изменения его амплитуды, мы можем создать более реалистичную анимацию.

Ранее выражение выглядело так:

```
x=Math.sin(time*3)*60;
rest=position;
add(rest, [x,0]);
```

Анимация выглядела так:

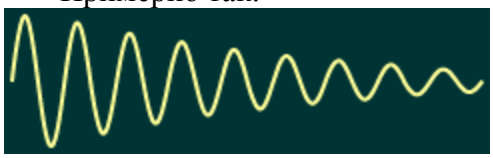
Теперь нам нужно сымитировать, что бы колебания постепенно затухали. Нам придется плавно уменьшать амплитуду до самого нуля.

В первоначальном выражении частота и амплитуда были заданы константами (3 и 60, соответственно) и не менялись. Теперь давайте выведем эти значения в соответствующие переменные:

```
frequency=3;
amplitude=60;
x=Math.sin(time*frequency)*amplitude;
rest=position; add(rest, [x,0]);
```

Теперь колебания затухают в течение анимации, но всё равно неестественно. Видимо, они затухают они уж очень медленно.

Примерно так:



Нам нужно ускорить процесс. Возьмём множителем обратную величину: *"amplitude=60/time."* теперь все происходит слишком быстро. Да и в самом начале, когда **time=0**, генерируется ошибка деления на ноль. Лучше воспользоваться уже изученным принципом линейной интерполяции linear():

```
amplitude=linear(time, 0, out_point-1, 120, 0);
```

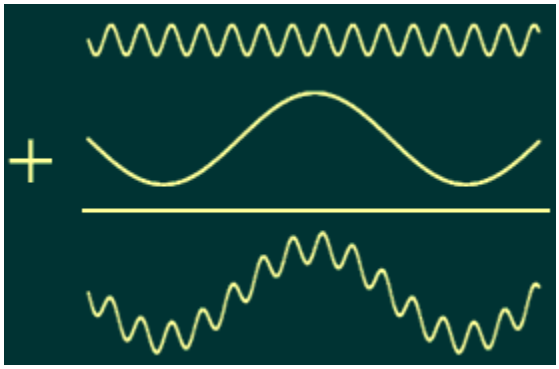
Наконец выражение примет следующий вид:

```
frequency=10;
amplitude=linear(time, 0, out_point-1, 120, 0);
 rest=position; x=Math.sin(time*frequency)*amplitude;
add(rest, [x,0]);
```

И всё же, как бы мы не старались, абсолютно реалистичной иммитации пружины мы не получим. Как я уже упоминал в начале руководства – иногда гораздо эффективнее бывает применить скрипты **MotionMath**. Как в этом случае (**spring.mm** и **dbspring.mm)**. Эти скрипты вы найдете в дистрибутиве **Adobe After Effects**, в папке **Application**.

Сложение и умножение колебаний.

Для создания более сложных колебаний вы можете использовать приемы сложения и умножения колебаний. К примеру, на следующем рисунке отображен процесс сложения высокочастотного низкоамплитудного колебания и низкочастотного высокоамплитудного колебания:



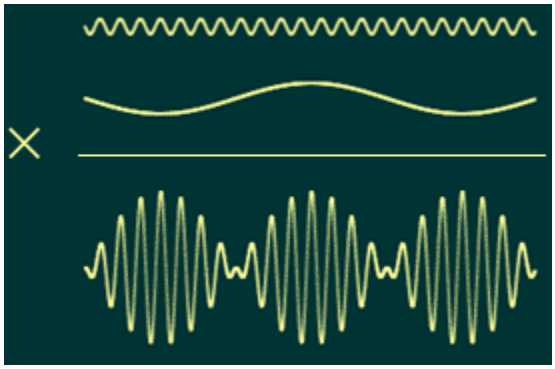
В виде выражения этот процесс выглядит так:

```
frequency_one=10;
amplitude_one=10;
frequency_two=1;
```

```
amplitude_two=30;
wave_one=Math.sin(time*frequency_one)*amplitude_one;
wave_two=Math.sin(time*frequency_two)*amplitude_two;
y=wave_one+wave_two;
x=time*32;
add(effect("Write-on").param(1), [x, y]);
```

Процесс сложения колебаний обслуживают все строки, кроме двух последних, отвечающих за правильный вывод на экран результатов.

С умножением колебаний дело обстоит несколько сложнее. Обратите внимание, что амплитуда результирующего колебания стала больше, чем была амплитуда любого из исходных колебаний. На следующей иллюстрации вы видите процесс перемножения высокочастотного и низкочастотного колебаний с примерно равными амплитудами.



В виде выражения это выглядит так:

```
frequency_one=15;
amplitude_one=5;
frequency_two=1;
amplitude_two=10;
wave_one=Math.sin(time*frequency_one)*amplitude_one;
wave_two=Math.sin(time*frequency_two)*amplitude_two;
y=wave_one*wave_two;
x=time*32;
add(effect("Write-on").param(1), [x, y]);
```

Процесс перемножения колебаний обслуживают все строки, кроме двух последних, отвечающих за правильный вывод на экран результатов.

Так как бы выглядел пример с анимацией пружины, если бы им управляло колебание, полученное аналогичным перемножением:

Обратные тригонометрические функции

К обратным тригонометрическим функциям относятся **Arcsine, Arccosine** и **Arctangent.**

Обратные тригонометрические функции это функции, определяющие угол по значению соответствующей тригонометрической функции.

К примеру, **arcsine(n)** даст нам в результате угол, синус которого равен n.

Из-за того, что тригонометрические функции являются периодическими, многие углы дают одни и те же значения. Так, к примеру, синус углов 0° и 360° равен одному и тому же значению - нулю. По этому и обратные значения и обратная тригонометрическая функция будет давать один результат.

Следуя синтаксису **Javascript**, обратные тригонометрические функции возвращают углы в радианах, но вы можете преобразовать их в градусы, используя встроеннный метод **radians_to_degrees(radians).**

Если вы подадите на вход **Math.asin()** или **Math.acos(),** значение которых будет меньше -1 или больше 1, то **Adobe After Effects** выдаст сообщение об ошибке и блокирует выражение.

В AE имеются ещё две тригонометрические функции: **Math.atan()** и **Math.atan2().** Первая из них в качестве аргумента требует просто число, вторая же требует два числа - x и y компоненты вектора. **Math.atan()** возвращает значение, лежащее в диапазоне от -**Pi/2** до **Pi/2**. **Math.atan2()** возвращает значение в более широком диапазоне от -Pi до Pi. Чаще используется вторая функция.

тангенс угла равен отношению противолежащего к углу катета к катету прилежащему.

Когда вам понадобится значение угла между сторонами, его можно будет вычислить с помощью следующего выражения:

```
temp=opposite/adjacent;
angle=Math.atan(temp);
```

Просчитывая таким выражением последовательность углов знайте, что в момент, когда прилежащая сторона станет равной 0, AE выдаст сообщение об ошибке, так как на ноль делить нельзя. Следовательно, угла в 90° таким выражением вы не просчитаете. Функция **Math.atan2()** предназначена для того, что бы избежать ошибок деления на ноль. Давайте ей на вход ряд значений длин прилежащих, противолежащих катетов - всё будет работать за счет исключения операции деления.

Пример на обратные тригонометрические функции

Откройте файл **Point_at.aep** из папки **Point_at**

С помощью функции **Math.atan2()** напишем выражение, позволяющее автоматизировать ориентацию слоя на другой слой. Слой будет поворачиваться, как ориентированный на управляющий слой.

Анимация является результатом применения следующего выражения:

```
this_point=position;
that_point=this_comp.layer("target").position;
delta=sub(this_point, that_point);
angle=Math.atan2(delta[1], delta[0]);
radians_to_degrees(angle);
```

Переменные *«This_point»* и *«that_point»* содержат координаты точек концов вектора, указывающего на ориентир. Мы высчитываем вектор между двумя позициями вычитанием одной из другой, и назначаем результату переменную *«delta».*

Далее выделяем x- и y-компоненты просчитанного вектора **delta** и передаем их на вход функции **Math.atan2(),** причем y-компонента передается первой. В конце концов мы конвертируем результат из радиан в градусы с помощью метода **radians_to_degrees** и передаем параметру поворота, которому и назначено все данное выражение.

Переменные *«This_point»* и *«that_point»* содержат координаты точек концов вектора, указывающего на ориентир. Мы высчитаем вектор между двумя позициями вычитанием одной из другой и назначим результату переменную delta.

Затем выделим x- и y-компоненты просчитанного вектора *«delta»* и передадим их на вход метода **Math.atan2(),** причем y-компонента передается первой. В конце концов мы конвертируем результат из радиан в градусы с помощью метода **radians_to_degrees** и передаем параметру поворота, которому и назначено все данное выражение.

8. Операторы и циклы

Если вы хотите писать по настоящему сложные выражения, то вам придётся научиться работать с операторами и циклами. Больших познаний в программировании у вас это не потребует, но кое-какие понятия придётся вынести из стандартного **JavaScript.** Эта глава представлена несколько обзорно, так как общие знания языка выражений у вас уже есть и вам требуется лишь уяснить особенности синтаксиса этих операторов и их возможности.

В языке **JavaScript** имеется достаточно большое количество операторов. В документации по языку выражений приведены лишь две их группы – условные операторы и операторы циклов. Да и среди этих достаточно обширных групп в официальной документации приведёны лишь по одному оператору из каждой из этих групп **«IF…..ELSE»** и **«while»** соответственно. По поводу остальных операторов – вполне возможно, что и они поддерживаются в языке выражений. Но так как острой необходимости в них я не ощутил, то дальше разбираться с этим я не стал.

Условные операторы

Сначала давайте поговорим о правилах синтаксиса для условных операторов.

Схематически условное выражение выглядит так:

if (Условие)
{Блок "условие истинно"}
else
{Блок "условие ложно"}

Как действует эта система:

1.Сначала условие проверяется на истинность **if (**Условие**).**
2. Затем в зависимости от условия выполняются соответствующие блоки:
а) Если условие истинно - выполняется блок "условие истинно";
б) если условие ложно, - выполняется блок "условие ложно".

Разберём эту тему на следующем примере:

```
if (time < 1)
{seed_random(1,true)}
else
{seed_random(2,true)}
```

В этом выражении говориться, что если время равно 0, то функция **seed_random** будет иметь такой вид **seed_random(1,true).** Если значение времени не равно 0, то **seed_random(2,true).**

Операторы циклов

Что делает оператор **WHILE?** Он выполняет блок операторов, пока условие остаётся верным (проверка производиться в начале каждого цикла).
Оператор **WHILE** имеет следующий синтаксис:

while(Условие)
{Тело цикла}

Сначала условие, которое является логическим или арифметическим выражением - проверяется на истинность (или неравенство 0). Затем в зависимости от условия выполняются соответствующие блоки:
1) Если условие истинно (или не равно 0 в случае арифметического выражения) - выполняется тело цикла и цикл запускается заново;
2) если условие ложно или равно 0 в случае арифметического выражения - цикл прерывается.
Пример:

```
seg_end_time = 0;
i =1;
tmin = 5;
tmax = 2;
while (time >= seg_end_time)
{i = i+1; seed_random(i,true);
seg_start_time = seg_end_time;
seg_end_time = seg_end_time + random(tmin,tmax);}
```

Человеческим языком говоря: если время больше ли равно **seg_end_time**, то выполняется тело цикла

```
i = i+1; seed_random(i,true);
seg_start_time = seg_end_time;
seg_end_time = seg_end_time + random(tmin,tmax);
```

Если же время меньше **seg_end_time**, то цикл прекращает выполняться.

9. Нововведения в версии Adobe After Effects 5.5

В версии 5.5 Adobe After Effects было введено много новшеств. Они затронули и аппарат выражений. Я специально выделил отдельную главу, для того чтобы отделить нововведения от остальных материалов.

Использование средств управления выражениями

В Adobe After Effects 5.5 были введены 6 новых эффектов в категорию **Expression Controls.** Они внешне никак не воздействуют на слой, к которому вы их добавили. Но вместо этого они обеспечивают контроль выражений. Один такой эффект может регулировать несколько свойств сразу. Вы можете добавлять эффект к нулевому слою, называть слой и эффект согласно функции, и пользоваться средствами контроля в дальнейшем с удобством.

Пример:

Вы можете создать нулевой слой, назвать его *«Controls»*. Затем добавить к нему эффект **slider,** дать эффекту имя *«Springiness».* Далее вы можете соединить свойство другого слоя с этим эффектом и управлять этим свойством.

```
spring = this_comp.layer("Controls").param("Springiness");
```

Expression Controls включает в себя следующие эффекты:

1. Slider Control

Ползунок ограничен по умолчанию в диапазоне от 0 до 100. Для изменения пределов щёлкните правой кнопкой на подчёркнутом значении ползунка и задайте нужное вам ограничение.

2. Angle Control

Содержит средство управления углом.

1. Checkbox Control

Содержит единственную флаговую кнопку.

4. Color Control

Содержит элемент управления образцам цвета и пипетку.

5. Point Control

Содержит инструмент управления точкой.

6. Layer Control

Содержит меню слоёв, в котором находятся все слом имеющиеся в активной композиции.

Создание выражения для единственного свойства

Когда вы пишете выражение в AE 5.5, то вы не нуждаетесь в указании свойства, на котором выражение написано. По умолчанию, объект для выражения – это свойство на котором оно написано. Это уменьшает размер выражения. Например, выражение с методом **wiggle** написано на свойстве **position**. Мы напишем на свойстве лишь:

wiggle(5,10)

В AE 5.0 это выражение писалось бы так:

position.wiggle(5,10);

Но вам придётся точно указать слой и свойство, когда вы извлекаете их из другого слоя. Например:

this_comp.layer("OtherLayer").position.wiggle(5, 10);

Использование операторов в выражениях

Как вы помните, при использовании векторной математики у нас получались весьма немаленькие выражения. В версии 5.5 появилась возможность пользоваться вместо громоздких операторов векторной математики обычными операторами. Например, это выражение **div(add(pos1, add(pos2, pos3)), 3),** теперь можно записать так (pos1 + pos2 + pos3) / 3.

Использование новых методов зацикливания ключевых кадров

AE 5.5 включает четыре новых метода зацикливания ключевых кадров.

Это следующие методы:

1. loop_in(type = "cycle", num_keyframes = 0)

Зацикливает участок времени, начиная от первого ключевого кадра вперёд по направлению к точке выхода. Цикл идёт от точки входа до первого ключевого кадра слоя. Сегмент для зацикливания задаётся количеством ключевых кадров.

2. loop_out(type = "cycle", num_keyframes = 0)

Зацикливает участок времени, начиная от последнего ключевого кадра назад по направлению к точке входа. Цикл идёт от последнего ключевого кадра до точки выхода слоя. Сегмент для зацикливания задаётся количеством ключевых кадров.

3. loop_in_duration(type = "cycle", duration = 0)

Зацикливает участок времени, начиная от первого ключевого кадра вперёд по направлению к точке выхода. Цикл идёт от точки входа до первого ключевого кадра слоя. Промежуток времени до следующего цикла задаётся определённой пользователем длительностью в секундах.

4. loop_out_duration(type = "cycle", duration = 0)

Зацикливает участок времени, начиная от последнего ключевого кадра назад по направлению к точке входа. Цикл идёт от последнего ключевого кадра до точки выхода слоя. Промежуток времени до следующего цикла задаётся определённой пользователем длительностью в секундах.

Типы установочных параметров

Доступные типы приведены ниже:

1. Cycle

Повторяет определённый участок.

loop_in(type = "cycle", num_keyframes = 20)

2. Pingpong

Повторяет определённый участок, но чередует повторение вперёд и повторение назад.

loop_out(type = "pingpong", num_keyframes = 40)

3. Offset

Повторяет определённый участок, но сдвигает каждый цикл на разность величины свойства в начале и в конце участка, умноженную на число раз которое проходит цикл.

4. Continue

Не повторяет определённый участок, но продолжает создавать анимацию, основываясь на скорости в первом или последнем ключевом кадре свойства. Например, если последний ключевой кадр свойства **scale** показывает, что слой имеет в этой точке масштаб 100% и свойству назначено выражение **loop_out(type="continue"),** тогда слой вместо зацикливания продолжит увеличиваться от 100% к точке выхода. Этот тип не имеет *«keyframes»* или *«duration»* параметров.

Просмотр аргументов функций и значений по умолчанию

Меню функций выражений содержит перечень аргументов и значений по умолчанию. Это упрощает запоминание элементов, которые вы можете контролировать, когда напишите выражение.

Для примера, в меню функций есть функция **wiggle**, с которой приводиться перечень его аргументов **wiggle(freq, amp, octaves = 1, amp_mult =5, t = time).** Вводный перечень указывает, что функция **wiggle** имеет пять аргументов. Знак (=) в последних трёх выражениях говорит о том, что использование этих аргументов является необязательным.

Использование функции "has_parent()"

Вы можете указать в вашем выражении, что вы хотите извлечь свойство из родительского слоя. Для этого используйте функцию **"has_parent()".**

Пример:

```
idx = index;
if (has_parent)
{idx = parent.index}
this_comp.layer(idx).position.wiggle(5, 20)
```

Получение случайного значения независящего от времени

Используя функцию **seed_random()** вы можете писать выражения для получения случайного значения, которые будут зависеть от номера слоя и не зависеть от времени.

Например, следующее выражение передаёт **true**, как второй параметр для функции **seed_random()**, задающей случайное число между 0 и 1.

```
seed_random(<any number>, true);
r = random()
```

Использование атрибута ".name"

After Effects 5.5 добавляет атрибут **".name"** к объектам **Comp, Footage, Layer, Mask, и Effect.** Это полезно, когда вы хотите применить одинаковые выражения к нескольким слоям. Изменение действия будет базироваться на имени объекта. Например, следующее выражение применяет **wiggle** на **position** слоя различно, в зависимости от того является ли данный слой слоем с именем *"hero":*

```
amp = 20;
if (name == "hero")
{amp = 40;}
wiggle(5, amp)
```

Доступ к ключевым кадрам и маркерам в выражениях

Сейчас вы можете писать выражения, которые имеют доступ к ключевым кадрам и маркерам. Эти выражения полезны, когда требуется, что бы свойство в определённый момент времени имело определённое значение. Для доступа к маркеру или ключевому кадру используем следующие атрибуты и методы:

- this_comp.marker(marker_num)

Возвращает время маркера композиции. Вы можете использовать это, например, для ухода слоя в 100% прозрачность.

```
mark_time = this_comp.marker(1);
linear(time, mark_time - 5, mark_time, 100, 0)
```

- num_keys

Возвращает число ключевых кадров свойства.

- key(idx)

Возвращает ключевой кадр по номеру. Например, **key(1)** это первый ключевой кадр композиции.

Когда вы имеете доступ к объекту **key**, то вы можете извлечь из него свойства **time**, **index** и **value**. Например, следующее выражение даёт вам доступ к свойству **value** третьего ключевого кадра.

```
position.key(3).value
```

Следующее выражение, будучи написанным, на слое с анимированным свойством **opacity,** игнорирует значение ключевых кадров и использует только их положение во времени. Оно определяет, когда должны происходить вспышки.

```
d = Math.abs(time - nearest_key(time).time);
ease_out(d, 0, 1, 100, 0)
```

- marker

Возвращает свойство **marker** для слоя. Это свойство отлично от свойств типа **position** тем, что поддерживаются только такие методы и атрибуты как: **key(), nearest_key** и **num_keys.** Вы можете извлечь ключи двумя путями:

1. По индексу **marker.key(1)**
2. По имени маркера **marker.key("foo")**

А вот вам ещё один пример использования маркеров в выражениях:

```
m1 = marker.key("Start").time;
m2 = marker.key("End").time;
linear(time, m1, m2, 0, 100);
```

Это выражение написано на свойстве opacity и создаёт линейное нарастание непрозрачности от маркера *"Start"* до маркера *"End".*

10. Примеры выражений

На протяжении руководства вы встретили и выполнили достаточно большое количество различных примеров. Но большинство из них не имели практического значения и были достаточно небольшие и простые. В этой же главе я постарался сделать упор на практическое применение выражений. Я долго думал, какие же примеры можно ввести в руководство. В конечном итоге, я взял несколько уроков с **Creative Cow** и **CyberMotion**, несколько модифицировал их и представил вашему вниманию.

Синхронизация анимации со звуком, используя маркеры слоя и выражения

Этот урок – один из нескольких уроков, опубликованных **Дэном Эббертсом** на **creativecow.net**. Он показался мне интересным, и я решил включить его в набор уроков по выражениям.

Перед нами стоит следующая задача – получить такую вот анимацию (барабанщик бьет по ударным в такт ритму), причем желательно приложа к этому минимум усилий. Я предлагаю использовать для этой цели выражения.

Откройте, пожалуйста, файл **markers.aep** из папки **Drummer**. В нем имеются следующие композиции: **cymbal comp** (содержит 6 кадров анимации ударной тарелки), **head comp** (содержит 10 кадров анимации головы барабанщика), **l arm comp** (содержит 6 кадров анимации левой руки), **r arm comp** (содержит 6 кадров анимации правой руки), **drummer.ai** (в этой композиции вы и будете работать с выражениями), **main comp** (композиция в которую вложена **drummer.ai** и применен эффект drop shadows). Исходные файлы в формате **Adobe Illustrator** находятся в папке **drummer.ai**. Кроме того, имеется звуковой файл **drums.wav**, с которым и будет проводится синхронизация.

Откройте композицию **drummer.ai**.

Идея заключается в следующем – поставить маркеры в моменты удара в барабан, а затем привязать к этим маркерам анимацию частей тела барабанщика.

Для начала, давайте расставим маркеры, управляющие анимацией. Создаем два нулевых объекта (на них мы и будем ставить управляющие маркеры). Маркеры слоя Null 1 будут управлять частотой ударов по тарелке и анимацией левой руки (она и бьет по тарелке), а Null 2 частотой кивков головой. Маркеры на звуковом слое будут обозначать моменты, когда барабанщик должен ударить по барабану.

Будем считать, что вы расставили маркеры соответственно ритму и готовы перейти к следующей части упражнения.

Назначаем на вложенные композиции **head comp, r arm comp, l arm comp, cymbal comp – enable time remap**. Это нам нужно для того, чтобы в любой момент времени мы могли проиграть анимацию, заключенную в этих вложенные композициях.

Назначим для time remap слоя **head comp** следующее выражение:

```
hit=marker.key(1).time;
t=time-this_comp.layer("Null 2").marker.nearest_key(time).time;
t+hit
```

Расшифруем данное выражение. Переменная hit возвращает значение времени первого маркера из слоя **head comp.** На самом деле, можно просто указать значение времени первого маркера цифрой (в данном случае 5). Но тогда вы потеряете потрясающую гибкость. Ведь для управления вам достаточно лишь двигать маркер, а если вы введете просто текущее значение, то этой возможности вы лишитесь. Переменная **t** равна разнице текущего времени и времени ближайшего маркера слоя

Null 2. В конечном итоге, голова будет кивать начиная с маркера 1 слоя **head comp** в течение hit.

Для слоев **l arm comp** и **cymbal comp** назначим одно и тоже выражение:

hit=marker.key(1).time;
t=time-this_comp.layer("Null 1").marker.nearest_key(time).time;
t+hit

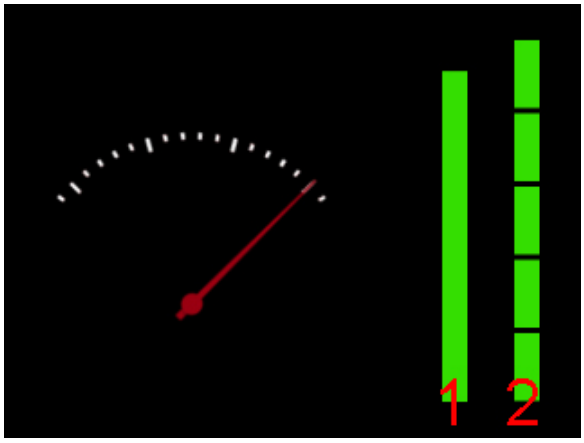
Смысл его такой же, что и у предыдущего. Единственно, что, управляющие маркеры находятся в слое **Null 1**, а первые маркеры в слоях **l arm comp** и **cymbal comp.**

Осталось назначить выражение слою **r arm comp.** Вот оно:

hit=marker.key(1).time;
t=time-this_comp.layer("drums.wav").marker.nearest_key(time).time;
t+hit

И опять тот же смысл, но управляющие маркеры находятся в слое **drums.wav**, а первый маркер в слое **r arm comp.**

Приборы и шкалы (Math.round)



Идею этого упражнения, я откопал где-то в кладовых супругов Мэйер. Несколько переработанное и дополненное представляю его вашему вниманию.

Нам требуется получить следующий эффект – при колебании стрелки прибора происходит колебание уровней шкал, причем одна из шкал секторная.

Важно: Обратите внимание на расстановку опорных точек в слоях.

Откройте файл **levels.aep** из папки **levels_tut**.

Для начала зададим случайное колебание стрелки прибора. Для этого назначим на свойство **rotation** слоя «dial» следующее выражение: **random(1,100).** Отлично, стрелка колеблется, как бешеная. Но… частота колебаний стрелки слишком высокая. Что же, будем с этим бороться. Конвертируем выражение в ключевые кадры и воспользуемся **smoother**ом. Ну вот, теперь анимация действительно похожа на колебание стрелки некоторого прибора.

Теперь нам необходимо заняться привязкой уровней шкал к колебанию стрелки. Назначим свойству **scale** слоя «bar 1» выражение:

[value[0], this_comp.layer("dial").rotation]

В этом массиве **value[0]** извлекает значение **x** свойства **scale,** а **this_comp. layer("dial"). rotation** извлекает значение **y** из свойства **scale.**

Со 2 шкалой дело обстоит сложнее. Если вы помните, эта шкала состоит из 5 секторов. И нам необходимо, что бы сектор появлялся полностью. Для этого можно воспользоваться функцией **math.round** или **math.ceil** (см. ***«Справочник по языку выражений Adobe After Effects»***). Рассмотрим как работает следующее выражение:

volume = this_comp.layer("dial").rotation / 20;
barheight = Math.round(volume);
[value[0], barheight * 20]

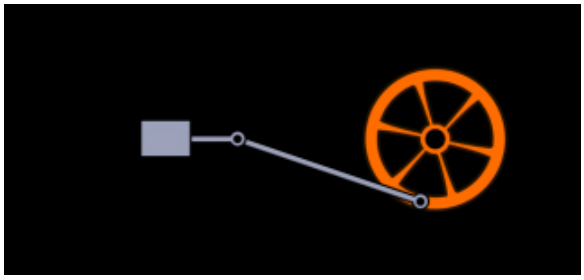
Во-первых введем переменную **volume** (извлекает значение свойства **rotation** слоя **"dial"**) и ограничим ее значение 5-ю. Почему именно пятью и как мы этого достигнем? Пятью потому что пять секторов в шкале, а достигнем этого разделив значение **rotation** на 20. Если вы помните максимальное значение свойства **rotation равно 100,** то есть после деления за значение 5 мы выйти не сможем.

Следующим шагом будет сделать так что бы минимальным делением шкалы стал сектор (у нас их 5). Для этого используем функцию **math.round**. Она округляет значение до целого числа. Переменная **barheight** равна округленной переменной **volume.**

Ну вот и практически все… Теперь вносим нашу переменную **barheight** в массив на место **y** свойства **scale.** Значение **x** меняться не будет (по крайней мере, с помощью выражений).

А теперь, представьте себе, как бы вы это делали с помощью ключевых кадров…

Родительские слои и выражения

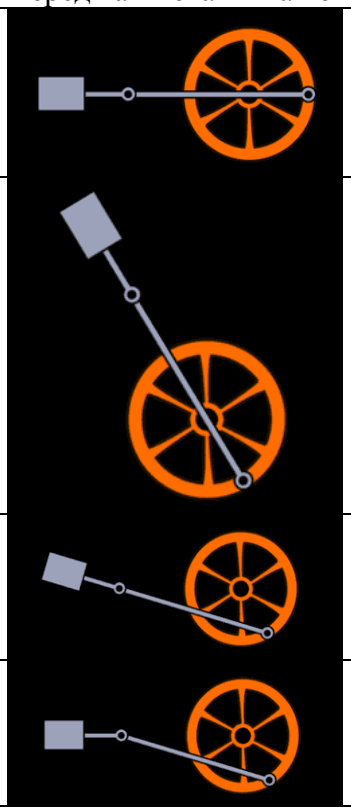


Это упражнение написано Дэном Эббертсом и общедоступно на **creativecow.net**. В нем он показывает возможности совместного использования родительских слоев и выражений для анимации механизма. Откройте файл **mashine.aep** из папки **mashine**.

Давайте взглянем на нашу композицию. Она содержит 3 слоя: **piston, crank, wheel**. Все они являются импортированными файлами **Adobe Illustrator**. Обратите внимание на размещение опорных точек на слоях. Это очень важно для нашей анимации.

Слой wheel является родительским для слоя crank, а слой crank является родительским для слоя piston. Если вы отключите уже имеющиеся выражения, то увидите, что при вращении колеса вокруг него вращается поршень (**piston**) и коленный вал(**crank**). Для получения нужной нам анимации используем выражения.

Перед вами этапы нашей работы:

		Это то, с чего мы начинаем
		Колесо вращается, но вместе с ним вращается поршень и коленный вал
		После настройки вала
		После настройки поршня

Настроим анимацию коленного вала. Откройте свойство rotation слоя ***«crank».*** В нем уже имеется выражение. Рассмотрим, как оно было создано.

Во-первых нам потребуются радиус колеса и длина коленного вала.

Для того, что бы узнать радиус колеса мы создадим следующее выражение:

r=length(this_comp.layer("wheel").position,position);

Значение позиции совпадает с опорными точками (мы же их не просто так настраивали). Так что мы получаем точный радиус колеса. Длина коленного вала вычисляется следующим образом: **l=length(position,this_comp.layer("piston").position);**

К сожалению, эти выражения не будут работать, так как обе точки должны находится в одном пространстве. Здесь нам поможет то, что точки дочернего слоя находятся в пространстве родительского слоя. Так что ничего страшного, мы можем использовать любое из приведенных ниже выражений:

r=length(this_comp.layer("wheel").anchor_point,position);
r=length(from_comp(this_comp.layer("wheel").position),anchor_point);
r=length(this_comp.layer("wheel").position,to_comp(anchor_point));

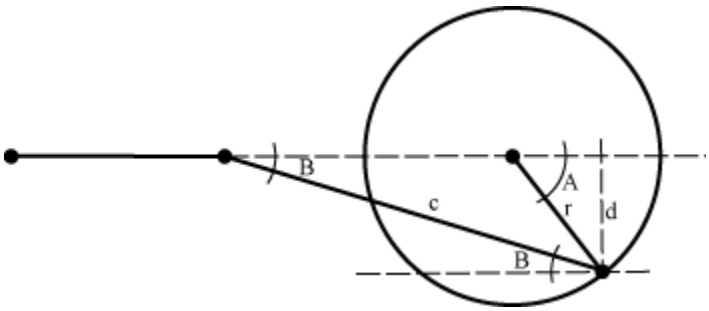
Давайте используем первое из них для радиуса.

Для длины подойдет следующее выражение:

l=length(anchor_point,this_comp.layer("piston").position);

Теперь нам необходимо будет вспомнить математику. Надеюсь у вас с нею все в порядке, а лично я, ее терпеть не могу. Ну что же, впереди немного тригонометрии.

Посмотрите на диаграмму. На ней описан схематично наш «движок», то есть то, что будет оживлять наш механизм.



A – угол вращения колеса

B – угол вала

r – радиус колеса

c- длина вала

Еще из школьной тригонометрии мы знаем (или сейчас вспоминаем), что:

r * sin(A) = d

и

c * sin(B) = d

итак

sin(B) = r * sin(A) / c

или

B = arc sin(r*sin(A) / c)

Для синуса и арксинуса мы используем функции **Math.sin** и **Math.asin**. Но эти функции используют значение угла не градусах, а в радианах. А After Effect как мы знаем тяготеет именно к градусам. Но ничего, к трудностям нам не привыкать. Конвертируем радианы в градусы при помощи **Math.PI**(это число PI). Зная, что в PI радиане 180 градусов мы можем конвертировать в обе стороны с легкостью.

rad2deg = 180 / Math.PI; //radians to degrees
deg2rad = Math.PI / 180; //degrees to radians

Поставим эти строки в начало выражения:

rad2deg = 180 / Math.PI; //radians to degrees
deg2rad = Math.PI / 180; //degrees to radians
r=length(this_comp.layer("wheel").anchor_point,position);
l=length(anchor_point,this_comp.layer("piston").position);

Отлично! Но продолжим…

Теперь вспомним про нашу формулу: **B = arc sin(r*sin(A) / c)**

Запишем ее на языке выражений, что бы получить угол вала:

Math.asin(r*Math.sin(this_comp.layer("wheel").rotation*deg2rad)/l)*rad2deg

Как видите, вал поворачивается на рассчитанный нами угол, но продолжает двигаться вместе с колесом. Для того, что бы исключить этот ненужный момент, давайте прибавим к нашему уже имеющемуся выражению отрицательное значение свойства вращения колеса.

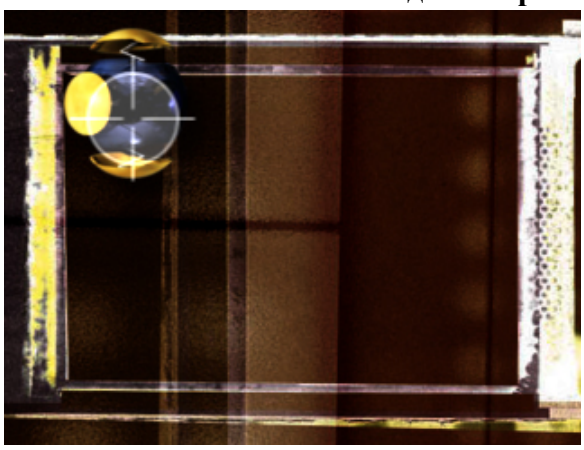
-this_comp.layer("wheel").rotation+
Math.asin(r*Math.sin(this_comp.layer("wheel").rotation*deg2rad)/l)*rad2deg

Для вращения поршня мы так же используем негативную настройку угла:

-1*(this_comp.layer("wheel").rotation+this_comp.layer("crank").rotation)

Ну, вот теперь, когда настройки завершены, мы может поставить несколько ключевых кадров в свойстве **rotation** колеса… и наш механизм оживет!!!

Math методы на примере использования math.clamp



Очередное упражнение производства **CyberMotion**. Нам необходимо получить что-то похожее на компьютерную игру (летает некая сфера, на которую нацеливается сенсор). Но, сенсор должен иметь меньшую зону движения, чем сфера. Для этого мы используем выражения.

Откройте файл **expressions1.aep** из папки **Meyer_tut**. В нем есть композиция **clamp**. В ней и будем работать.

Сфере задано движение с помощью ключей.

На сенсор назначено выражение:

upper_left = [60,60];
lower_right = [260,180];
clamp(this_comp.layer("gizmo").position, upper_left, lower_right)

Переменная **upper_left** обозначает верхнюю левую границу (как следует из названия), а переменная **lower_right** обозначает нижнюю правую границу движения сенсора.

Строка **clamp(this_comp.layer("gizmo").position, upper_left, lower_right**) говорит о том, что позиция слоя **“CM_sensor_crl.tif”** равна позиции слоя **"gizmo"**, но не может быть больше границ **upper_left, lower_right.**

Условные операторы If/Then/Else

В данном упражнении будем разбираться с условными операторами **If/Then/Else**. Урок был разработан в незабвенном **CyberMotion.**

Откройте файл **expressions1.aep** из папки **Meyer_tut**. В нем есть композиция **intelligent marker trig**. В ней и будем работать.

Свойству scale слоя ***«CP_AlarmClock.tif»*** присвоено выражение:

```
scale_value = 0;
time_to_marker = time - marker.nearest_key(time).time;
if (time_to_marker >= 0)
{scale_value = linear(time_to_marker, 0, 0.5, 100, 0)};
[scale_value,scale_value]
```

Давайте разберем его построчно.

scale_value = 0; //переменная **cale_value=0**

time_to_marker = time - marker.nearest_key(time).time; //вводиться переменная time_to_marker, которая равна разнице текущего времени и времени ближайшего маркера.

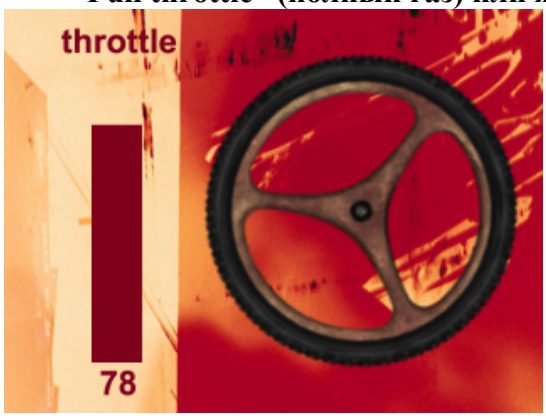
if (time_to_marker >= 0) // условие – пока переменная time_to_marker больше или равна 0 выполняется блок условие истинно

{scale_value = linear(time_to_marker, 0, 0.5, 100, 0)}; //блок условие истинно – значение scale будет уменьшаться со 100 до 0 за время от 0 до 0,5.

[scale_value,scale_value] //массив, возвращающий значение **x** и **y** координат для свойства scale.

Ну вот, собственно, и все.

“Full throttle” (полный газ) или же использование оператора цикла while



Довольно интересный урок из серии **“Deeper modes of expressions”** производства все тех же Мэйер.

Откройте файл **expressions1.aep** из папки **Meyer_tut**. В нем есть композиция **while loop sim**. В ней и будем работать.

Слою с колесом назначен эффект **Slider** (см. главу «***Нововведения в версии Adobe After Effects 5.5*** »). Он то и будет основой для всей анимации.

Для начала создадим анимацию колеса.

Полностью это выражение выглядит так:

```
step_time = 0.01;
total_rot = 0;
while (step_time < time)
    {
    total_rot += effect("throttle").param("Slider").value_at_time(step_time) / 5;
    step_time += this_comp.frame_duration;
    }
total_rot
```

Разберем его.

step_time = 0.01; // переменная равная **0.01**

total_rot = 0; // переменная равная **0**

while (step_time < time) //условие – пока переменная step_time<меньше текущего времени условие истинно и цикл продолжается

Перед тем, как разбирать выражение дальше, необходимо поговорить о такой вещи, как накопление чисел (**accumulating numbers**).

Цитирую из руководства по **Java Script**: «Циклы часто используют числа, которые должны нарастать. Например, при вычислении номера фрэйма. **Java Script** имеет специальную функцию для таких случаев, которая упрощает эту процедуру.»

Например, в нашем упражнении мы используем следующее утверждение **step_time += this_comp.frame_duration.** Символ += называется **«add-and-assign»** оператор. Он действует так: новое значение для переменной **step_time** равно старому значению этой переменной плюс длина кадра.

Теперь строки

total_rot += effect("throttle").param("Slider").value_at_time(step_time) / 5;
step_time += this_comp.frame_duration;

вам понятны.

Ну и строка **total_rot** возвращает конечное значение переменной **total_rot.**

Анимация колеса готова, теперь давайте займемся gas pedal bar.

В зависимости от скорости вращения колеса будет меняться и высота шкалы. Для этого привяжем **y-scale** этого слоя к все тому же **slider**y слоя ***«wheel».***

this_comp.layer("wheel").effect("throttle").param("Slider")

Кроме изменения высоты шкалы будет отображаться и некое значение. Для этого слою ***«text & numbers»*** был назначен эффект **numbers**. Затем значение свойства value/offset/random max эффекта numbers будет привязано к же **slider**y слоя ***«wheel».***

this_comp.layer("wheel").effect("throttle").param("Slider")

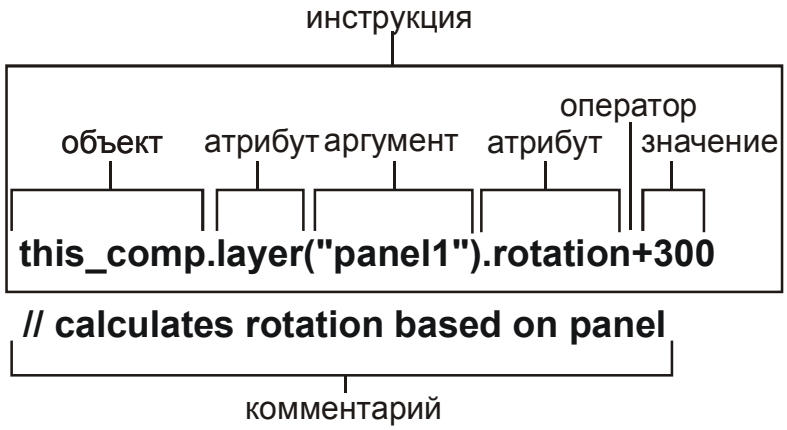
Все… Теперь газ до отказа!!!

11. Справочник по языку выражений Adobe After Effects

Синтаксис выражений

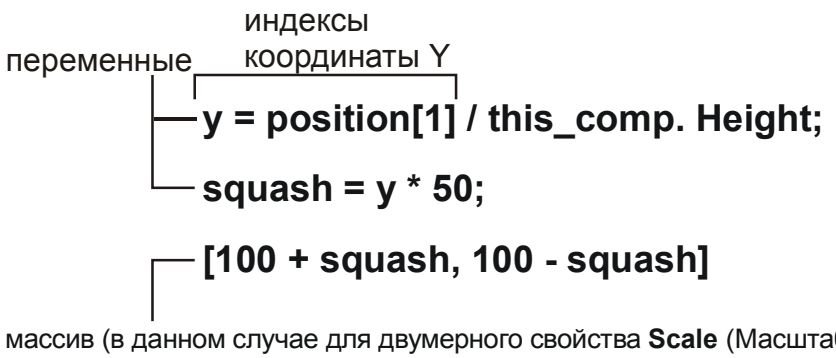
Я специально поставил тему ***«Синтаксис языка выражений»,*** в начало справочника. После того как вы освоитесь с терминологией и основами синтаксиса, вам будет проще пользоваться справочником и писать грамотные выражения.

Синтаксис языка выражений соответствует синтаксису языка **JavaScript.**



На рисунке приведён пример простого выражения, написанного с соблюдением правил синтаксиса. В структуре этого выражения вы можете видеть основные используемые элементы. Подробнее см. в *«Элементы синтаксиса выражений».*

На примере этого выражения разберём такое понятие, как «уровни вложенности». Не секрет, что выражения представляют собой иерархическую структуру элементов. Наверху находится глобальный объект (**this_comp**), в него вложен нижестоящий в иерархии (**layer(“panel1”)**), а в него вложено свойство, к которому прибавляется значение (**rotation+300**). Уровни разделяются точками.



В этом примере мы попытаемся имитировать отскакивающий от пола мяч. Для этого, применяем данное выражение к свойству Scale, слоя изображающего мяч. У нас получиться довольно правдоподобная имитация «сквоша» (расплющивания) мяча при соударении с поверхностью земли. В данном выражении с успехом применяются переменные для задания масштаба слоя через значение его координаты положения Y.

Элементы синтаксиса выражений

Statement (Инструкция). Является описанием законченной мысли или команды. Инструкции в одном выражении разделяются точкой с запятой (**;**).

Dot syntax (Точечный синтаксис). Описывает метод конструирования каждой инструкции. Части инструкции должны быть отделены друг от друга точкой.

Comments (Комментарии). Включаются в скрипты только для справки; программа их игнорирует. Комментарии должны вставляться между двойным слешем (//) и окончанием строки: **//это комментарий.**

Objects (Объекты). Относятся к любым поименованным данным и сравнимы с существительными в различных языках. Язык выражений **After Effects** включает объекты, уникальные для этой программы - например, **comp** и **layer, footage**, **camera.**

Attributes (Атрибуты). Являются поименованными значениями объекта и могут быть использованы в нескольких объектах. Например, **width** (ширина) представляет собой атрибут для следующей инструкции: **this_comp.width**. Кроме того, атрибуты могут возвращать в качестве результата другие объекты. Например, фрагмент инструкции **layer(«panel1»)** возвращает имя слоя.

Methods (Методы). Подобны атрибутам, но вместо возвращения данных - например, имен слоев — они обычно служат для задания данных. За именами методов следуют круглые скобки, в которых (в зависимости от метода) содержатся определенные переменные или значения. Например, метод **random(100),** использованный в выражении для значения свойства **Opacity** (Непрозрачность), заставляет непрозрачность слоя случайно меняться от 0 до 100 процентов.

Values (Значения). Определяют количественные данные, например числовые. Однако значение может быть и строковым (буквенно-цифровой строкой) или массивом.

Array (Массив). Определяет набор значений данных, на которые ссылаются по номеру. Массивы, как принято говорить, имеют размерность, которая обычно соответствует различным аспектам значения свойства. Например, свойство **Position** (Положение) (в двумерном пространстве) имеет два измерения: значения координат **x** и

y. Свойство **Scale** (Масштаб) в двумерном пространстве может иметь одно или два измерения: значение масштаба в целом или отдельные значения масштаба по ширине и высоте. В руководстве по выражениям размерность массива указывается в квадратных скобках: **[ ].** Массив может включать в себя числа, объекты и даже другие массивы.

Vector (Вектор). Является числовым массивом. Наиболее распространённый и применяемый в After Effects массив. Например, **position[0,1].**

Index (Индекс). Обозначает схемы нумерации, используемые для ссылки на значения в массивах или объектах. Массивы индексируются, начиная с 0. Например, **position[0]** возвращает координату **x**: положения, а **position[1]** возвращает координату **y** положения. Объекты, подобные слоям, маскам и параметрам эффектов индексируются, начиная с 1. Например, **layer(1)** ссылается на верхний слой композиции. Однако на такие объекты лучше ссылаться по имени, а не по индексу.

Variables (Переменные). Представляют собой имена, ассоциированные со значением и, как говорят, «хранят» значение. Например, выражение может содержать инструкцию **radius=20** для присвоения переменной **radius** значения 20. Далее в выражении можно использовать эту переменную как часть другой инструкции. Переменные и атрибуты, в сущности, означают одно и то же, за исключением того, что вы можете сами придумывать имя переменной.

Operators (Операторы). Символы, используемые для выполнения вычислений, многие из которых вы знаете из основ математики. Операторы, которые присваивают значения, например, **x = y**, известны как присваивания. Кроме того, операторы могут сравнивать значения, например, **x<y** (**x** меньше, чем **y**). Такие операторы называют сравнениями; они имеют значения true (истина) и false (ложь), или 1 и 0.

Arguments (Аргументы). Описывают тип данных (например, число или массив) необходимых для задания объекта. Например, объект **Comp(name)** требует на месте аргумента **name** использовать строковые, или буквенно-числовые данные. Следовательно, аргументом служит строка. В данном случае фактическое имя композиции должно быть заключено в скобки и - поскольку в данном случае это так называемое значение строкового литерала — в кавычки: **Comp(”final comp”).** Напротив, объект **this_comp** не требует аргумента. Этот тип глобальных объектов полностью самодостаточен. Иными словами, ему не требуется указание дополнительных данных.

Returns (Возвращаемые значения). Можно представить себе как еще один способ сказать: *«вернуть в результате данные этого типа».* Например, инструкция **this_comp.layer(«panel1»).rotation** возвращает число, в данном случае, значение свойства **Rotation** (Вращение) слоя с именем **panel1**.

Параметры слоя (Layer parameters)

Прежде чем углубляться в дебри методов, атрибутов, свойств, параметров и т.д., посмотрите на один из основных объектов ***«слой»***. Обратите внимание, что пределы значений возможные и реальные отличаются.

В качестве справки здесь приведены размерности, единицы измерения, пределы значений для основных свойств, с которыми вам предстоит работать.

Параметр слоя	Смысл	Размерность	Единицы	Пределы значений	Реальные пределы значений
anchor_point	Положение	2 или 3 [x, y, (z)]	пиксели	±бесконечность	0–ширина/

	точки привязки на слое				высота слоя
position	Позиция слоя в пространстве	2 или 3 [x, y, (z)]	пиксели	±бесконечность	0–высота/ ширина композиции
scale	Значение масштаба слоя	2 или 3 [width, height, (depth)]	проценты	±бесконечность	±100
rotation	Значение поворота слоя	1	градусы	±бесконечность	0–360 (циклически)
opacity	Непрозрачность слоя	1	проценты	0–100	0–100
orientation (только для трёхмерных слоёв)	Ориентация слоя в трёхмерном пространстве	3 [x, y, z]	градусы	±бесконечность	0-360 (циклически)
audio_levels		2 [левый канал, правый канал]	градусы	-192–24	-48–12
ambient		1	проценты	0–100	0–100
diffuse		1	проценты	0–100	0–100
specular		1	проценты	0–100	0–100
shininess		1	проценты	0-100	0-100

Глобальные объекты (Global objects)

Глобальные объекты это то, с чем вам придётся сталкиваться постоянно. Они помогут вам проложить путь вашему выражению к свойствам любого слоя, композиции, исходника, получить доступ ко времени, к какой-то величине.

Глобальный объект	Тип возвращённого значения	Примеры
this_layer	Слой, источник света или камера	"this_layer.width"
this_comp	Данная композиция	"this_comp.layer(2)."
comp("name")	Композиция с именем *"name"*	“comp("red").layer(5)”
footage("name")	Исходный видеоматериал	footage("GoldHoney.PSD")
time	Число	“this_comp.layer(1).position.value_at_time(time-5)”
value	Число	"value + 5"

Методы векторной математики (Vector math methods)

Из-за ограничений синтаксиса языка **Javascript**, мы не можем напрямую пользоваться стандартными математическими операторами типа +, -, /, и *, осуществляя соответствующие операции над векторами. В случаях, когда нам требуются эти операции, нам придется использовать специальные методы векторной математики, которые приведены в данной таблице.

В версии 5.5 (см. главу «Нововведения в AAE 5.5») мы можем пользоваться и стандартными математическими операторами.

Векторные математические методы	Смысл	объект или тип объекта	Примеры
add(vector1, vector2)	Складывает два вектора покомпонентно	Массив	add([80,20],[70,10]) ⇒[80+70,20+10]
sub(vector1, vector2)	Вычитает два вектора покомпонентно	Массив	sub(90,30)⇒ 90-30
mul(vector, Number)	Перемножает каждый элемент вектора на число	Массив	mul(50,random(10))⇒ 50*(random(10))
div(vector, Number)	Делит каждый элемент вектора на число	Массив	div(gauss_random(50,90),2) ⇒ (gauss_random(50,90))/2

dot(vector1, vector2)	Возвращает внутреннее произведение, которое является результатом покомпонентного перемножения векторов	Число	dot(2,random(50))
cross(vector1, vector2)	Возвращает векторное произведение	Массив	cross([2,3],[0,0])
normalize(vector)	Нормализует вектор таким образом, что его длина становиться равной 1	Массив	Тоже самое что div(*v*, length(*v*))
length(vector)	Возвращает длину вектора	Число	length(5)
length(from_point, to_point)	Возвращает дистанцию между двумя точками	Число	length(90,30) ⇒ length(sub(90,30))
look_at(from_point, to_point)	Ориентация одного объекта относительно другого	Массив	look_at(position, this_comp.layer(1).position)

Случайные числовые методы (Random number methods)

Примечание: значения **min** и **max** это минимум и максимум, задавая которые вы определяете пределы колебаний значения.

Методы по Гауссу отличаются от обычных методов тем, что в заданный вами числовой диапазон попадает 90% значений. Остальные 10% будут лежать вне диапозона.

Случайные число-вые методы	Смысл	объект или тип объекта	Примеры
random(***max***)	Получающийся результат это число между 0 и максимумом.	Число	[random(400),300]
random(**min, max**)	Получающийся результат это число между минимумом и максимумом.	Число	[random(200,500),300]
random(***max_arr***)	Получающееся значение это вектор совпадающий размерностью с ***max_arr,*** каждый компонент его находится между 0 и максимумом.	Массив	[gauss_random(200,300),300]
random(***min_arr, max_arr***)	Получающееся значение это вектор с одинаковой размерностью ***min_arr*** и ***max_arr.*** Каждый компонент его находится между минимумом и максимумом.	Массив	random([100, 200],[300, 400])
gauss_random(***max***)	Получающийся результат это число между 0 и максимумом. В отличие от **random(max),** где все числа находятся в промежутке от 0 до максимума, в **gauss_random()** 10% полученных результатов будут лежать за пределами этого промежутка.	Число	[gauss_random(400),300]
gauss_random(***min, max***)	Получающийся результат это число между минимумом и максимумом. В отличие от **random(min,max),** где все числа находятся в промежутке от минимума до максимума, в **gauss_random (*min, max*)** 10% полученных результатов будут лежать за пределами этого промежутка.	Число	[gauss_random(200,500),300]
gauss_random(***max_ arr***)	Получающееся значение это вектор совпадающий размерностью с ***max_arr,*** каждый компонент его находится между 0 и максимумом. В отличие от **random(max_arr),** где все числа находятся в промежутке от 0 до максимума, в **gauss_random (m*ax_arr*)** 10% полученных результатов будут лежать за пределами этого промежутка.	Массив	[gauss_random(200,300),300]
gauss_random(***min_arr, max_arr***)	Получающееся значение это вектор с одинаковой размерностью ***min_arr*** и ***max_arr.*** Каждый компонент его находится между минимумом и максимумом. В отличие от **random(*min_arr, max_arr*),** где все числа находятся в промежутке от 0 до	Массив	gauss_random([100,100],[200,200])

	максимума, в **gauss_random (*min_arr, max_arr*)** 10% полученных результатов будут лежать за пре-делами этого проме-жутка.		
seed_random(***n***)	Берет имеющийся **seed** и добавляет к нему значение **n** (число)		seed_random(60); random(10,30)
noise(***val***)	Возвращает значения между 0 и 1. обычно применяется тогда, когда вы сделали несколько копий слоя и вам нужно внести разнообразие в их эдентичные свойства	Число	add(position, [noise(position)*50])

Методы интерполяции (Interpolation methods)

Применение этих методов сложно тем, что они имеют большое количество аргументов. Далее приведён список аргументов:

t - "входное" значение переменной. А переменной может быть как любой из параметров, так и собственно любая переменная.

t_min - минимально возможное значение переменной t.

t_max - максимально возможное значение переменной t.

value1 - минимальное значение "на выходе". Если t меньше или равно t_min, то на "выход" попадет величина, заданная value1. Value1 может быть числом или вектором.

value2 - максимальное значение "на выходе". Если t больше t_max, на "выход" попадет величина, заданная value2. Value 2 может быть числом или вектором, но размерности Value2 и value1, должны совпадать, иначе Adobe After Effects их просто проигнорирует.

Аргументы t, value1, value2 являются обязательными. А наличие аргументов t_min и t_max не обязательно. Если они не будут указаны, то Adobe After Effects будет считать их равными 0 и 1, соответственно.

Методы интерполяции	Смысл	Возвращаемый объект или тип объекта	Примеры
linear(t, t_min, t_max, value1, value2)	Изменение значения от value1 до value2 за время от t_min до t_max линейно	Число или массив	linear (time, 0, 5, 0, 100)
ease(t, t_min, t_max, value1, value2)	Изменение значения от value1 до value2 за время от t_min до t_max со смягчёнными как входящими, так и выходящими кадрами	Число или массив	ease (time, 0, 5, 0, 100)
ease_in(t, t_min, t_max, value1, value2)	Изменение значения от value1 до value2 за время от t_min до t_max max со смягчёнными входящими кадрами	Число или массив	ease_in (time, 0, 5, 0, 100)
ease_out(t, t_min, t_max, value1, value2)	Изменение значения от value1 до value2 за время от t_min до t_max max со смягчёнными выходящими	Число или массив	ease_out (time, 0, 5, 0, 100)

	кадрами		

Пример: **linear(time, 0, 5, 0, 360)**

Входная переменная **time** меняется от 0 до 5 сек, значение функции меняется от 0 до 360^0, причем меняется по линейному закону. Будучи примененным, к параметру поворота (**Rotation**) слоя, это выражение заставит слой повернуться на 360 градусов в течение первых 5 секунд композиции.

Методы цветовой конверсии (Color conversion methods)

Данные методы используются тогда, когда появляется необходимость конвертировать одну цветовую систему в другую. Например, **rgb** в **hsl**.

Методы цветовой конверсии	Смысл	Возвращаемый объект или тип объекта	Размерность	Примеры
rgb_to_hsl (***rgba)***	Переводит цвета rgb в hsl	Массив	4	rgb_to_hsl(effect ("Change Color"). param("Color To Change"))
hsl_to_rgb (***hsla***)	Переводит цвета hsl в rgb	Массив	4	hsl_to_rgb (effect ("Colorama"). param("Color To Change"))

Атрибуты и методы композиции (Comp attributes and methods)

Среди нижеприведённых атрибутов и методов, конечно, самым используемым является **layer*(*index или *"*name"*)***, но и остальные использовать периодически придётся.

Атрибуты и методы композиции	Смысл	Возвращаемый объект или тип объекта	Размерность	Единицы измерения или пределы	Примеры
active_camera	Возвращает активную камеру	Камера	1		
width	Ширина композиции	Число	1	пиксели	circumference= width*Math.PI
height	Высота композиции	Число	1	пиксели	this_comp. height/10
duration	Длительность композиции	Число	1	секунды	this_comp. duration
frame_duration	Длительность кадра	Число	1	секунды	this_comp.frame _duration*100
bg_color	Цвет фона	Вектор	4	От 0 до 1	
shutter_angle	Значение shutter_angle	Число	1	градусы	this_comp.shutter_ angle/2
shutter_phase	Значение shutter_phase	Число	1	градусы	this_comp.shutter_ phase
num_layers	Число слоёв	Число	1		this_comp.num_ layers
pixel_aspect	Соотношение геометрических размеров пикселя	Число	1	NA (ширина/ высота)	this_comp.pixel_ aspect*30
layer***(*index или *"*name"*)***	Извлекает значение слоя по индексу или имени	Layer, Light или Camera			this_comp.layer ("Solid 1")
Layer ***(*other_layer, rel_index*)***	извлекает слой, который находится выше или ниже слоя **other_layer** на	Layer, Light или Camera			Layer (this_layer, -2)

	значение **rel_index**				

Атрибуты и методы исходного материала (Footage attributes and methods)

Атрибуты и методы исходного материла	Смысл	Возвращаемый объект или тип объекта	Размерность	Единицы измерения	Примеры
width	Ширина исходного материала	Число	1	пиксели	footage("1.PSD"). width
height	высота исходного материала	Число	1	пиксели	footage("1.PSD"). height
duration	Длительность исходного материала	Число	1	секунды	footage("1.mpg"). duration
frame_duration	Продолжительность кадра	Число	1	секунды	footage("raid.avi"). frame_duration
pixel_aspect	Соотношение геометрических размеров пикселя	Число	1	NA (ширина/ высота)	footage("Evil.avi"). pixel_aspect

Атрибуты и методы слоя (Layer attributes and methods)

Собственно - это основа основ. Для большинства ваших задач, приведённых в этой таблице атрибутов и методов, хватит с лихвой.

Атрибуты и методы слоя	Смысл	Возвращаемый объект или тип объекта	Раз мерность	Единицы измерения	Примеры
width	Ширина слоя	Число	1	пиксели	this_layer.width/ 10
height	Высота слоя	Число	1	пиксели	this_layer. height /5
start_time	Время, в котором начинается слой	Число	1	секунды	this_layer. start_time+60
in_point	Точка входа слоя	Число	1	секунды	this_comp.layer ("Camera 1"). in_point+3
out_point	Точка выхода слоя	Число	1	секунды	this_comp.layer ("Spray"). out_point -20
has_video	Является ли слой видео слоем?	Булевское значение	1	Булевское значение	has_video(0)
has_audio	Является ли слой аудио слоем?	Булевское значение	1	Булевское значение	has_audio(1)
layer_active	Активен ли слой?(👁)	Булевское значение	1	Булевское значение	Active(0)
audio_active	Включено ли аудио для данного слоя?(🔊)	Булевское значение	1	Булевское значение	audio_active(1)
audio_levels	Уровень звукового сигнала	Property	2 [левый канал, правый канал]	децибелы	This_comp. layer(1).audio_ levels
index	Место слоя в композиции (какой он по счёту)	Число	1	Число	Index(4)
parent	Тип родительского слоя для данного слоя	Слой, свет, камера			position[0] + parent.width

anchor_point	Положение точки привязки на слое	Свойство	2 или 3 [x, y, (z)]	пиксели (в пространстве слоя)	This_comp. layer(1).anchor_point
position	Позиция слоя в пространстве	Свойство	2 или 3 [x, y, (z)]	пиксели	position[random (200,300), 100, 50]
scale	Значение масштаба слоя	Свойство	2 или 3 [ширина, высота, (глубина)]	Проценты	Scale[random (100),position]
opacity	Непрозрачность слоя	Свойство	1	Проценты	This_comp. layer(1).opacity
rotation	Значение поворота слоя	Свойство	1	Градусы	wert=rotation
orientation (Только для 3d слоя)	Ориентация слоя в трёхмерном пространстве	Свойство	3 [x, y, z]	Градусы	X= orientation(0); Y= orientation(1); Z= orientation(2);
ambient (Только для 3d слоя)	Уровень заполняющего света, распространяющего ся от источника	Свойство	1	Проценты	
shininess (Только для 3d слоя)	Уровень бликов на слое	Свойство	1	Проценты	
casts_shadows (Только для 3d слоя)	Отбрасывает ли слой тень?	Булевское значение	1	Булевское значение	
accepts_shadows (Только для 3d слоя)	Будет ли отражать данный слой тени от других слоёв	Булевское значение	1	Булевское значение	
accepts_lights (Только для 3d слоя)	Будет ли отражать данный слой свет от источников освещения	Булевское значение	1	Булевское значение	
time_remap	Значение time_remap, если он включен	Свойство	1	секунды	
source	Исходный файл для слоя	Композиция или исходный материал			
mask(индекс или имя маски)	Индекс или имя маски	Маска			
effect(индекс или имя эффекта)	Индекс или имя эффекта	Эффект			

Методы трансформации пространства слоя (Layer space transform methods)

Эти методы приобретают свою истинную ценность тогда, когда вам необходимо связать какие-то свойства, а они находятся в разных пространствах (нахождение слоёв в одном пространстве обязательное требование для связи свойств с помощью выражений). Вот тогда-то и придут вам на помощь методы пространственной трансформации.

Методы пространственной трансформации слоя	Смысл	Возвращаемый объект или тип объекта	Размерность	Примеры
to_comp (*point, t = time*)	Преобразует точку пространства слоя в точку пространства композиции	Массив	2 или 3	to_comp(anchor_point)
from_comp	Преобразует точку	Массив	2 или 3	from_comp(this_comp.layer(2).

(*point, t=time*)	пространства композиции в точку пространства слоя			position)
to_world (*point, t=time*)	Преобразует точку пространства слоя в видонезависимую точку мирового пространства	Массив	2 или 3	to_world(effect ("Bulge"). param("Bulge Center")
from_world (*point, t=time*)	Преобразует точку мирового пространства в точку пространства слоя	Массив	2 или 3	from_world(this_comp.layer(2). position)
to_comp_vec (*point, t=time*)	Преобразует вектор из пространства слоя в пространство композиции	Массив	2 или 3	to_comp_vec([1, 0])
from_comp_vec (*point, t=time*)	Преобразует вектор из пространства композиции в пространство слоя	Массив	2 или 3	dir=sub(position, this_comp.layer(2).position); from_comp_vec(dir)
to_world_vec (*point, t=time*)	Преобразует вектор из пространства слоя в мировое пространство	Массив	2 или 3	p1 = effect("Eye Bulge 1"). param("Bulge Center"); p2 = effect("Eye Bulge 2"). param("Bulge Center"); to_world(sub(p1, p2))
from_world_vec (*point, t=time*)	Преобразует вектор из мирового пространства в пространство слоя	Массив	2 или 3	from_world(this_comp.layer(2). position)

Атрибуты и методы камеры (Camera attributes and methods)

Объект *«камера»* имеет те же свойства, что и объект *«слой»*. За исключением **source, effect, mask, width, height, anchor_point, scale, opacity, audio_levels, time_rema**р, а так же **свойств материала**. В дополнение, он имеет, и свои особые свойства (см. таблицу).

Атрибуты и методы камеры	Смысл	Возвращаемый объект или тип объекта	Размерность	Единицы измерения
point_of_interest	Точка, на которую наводится камера	Свойство	3	Пиксели
zoom	Масштаб	Свойство	1	Пиксели
depth_of_field	Глубина поля	Свойство	1	0-выключено, 1-включено
focus_distance	Фокусное расстояние	Свойство	1	Пиксели
aperture	апертура	Свойство	1	Пиксели
blur_level	Уровень размытия	Свойство	1	Проценты
active	Активна ли камера? (👁)	Булевское значение	1	Булевское значение

Атрибуты и методы источника света (Light attributes and methods)

Объект *«источник света»* имеет те же свойства, что и объект *«слой»*. За исключением **source, effect, mask, width, height, anchor_point, scale, opacity, audio_levels, time_rema**р, а так же **свойств материала**. В дополнение, он имеет, и свои особые свойства (см. таблицу).

Атрибуты и методы источника света	Смысл	Возвращаемый объект или тип объекта	Размерность	Единицы измерения
point_of_interest	Точка, на которую наводится источник	свойство	3	Пиксели
intensity	Интенсивность освещённости	свойство	1	Проценты
color	Цвет света	свойство	4	Параметры red,

				green, blue, и alpha в пределах от 0 до1
cone_angle	Угол падения светового конуса	свойство	1	Градусы
cone_feather	Мягкость конуса	свойство	1	Проценты
shadow_darkness	Темнота тени	свойство	1	Проценты
shadow_diffusion	Рассеивание тени	свойство	1	Пиксели

Атрибуты и методы эффекта (Effect attributes and methods)

Атрибуты и методы эффекта	Смысл	Возвращаемый объект или тип объекта	Размерность	Примеры
active	Включен ли эффект?	Булевское значение	1	
param(*name***)**	Возвращает параметр внутри эффекта, находя его по специфическому имени	Свойство	1	effect("Bulge").param("Bulge Height")
param(*index***)**	Возвращает параметр внутри эффекта, находя его по индексу. Индексирование начинается с 1.	Свойство	1	effect(3). param(4)

Атрибуты и методы маски (Mask attributes and methods)

Атрибуты и методы маски	Смысл	Возвращаемый объект или тип объекта	Размерность	Примеры
opacity	Непрозрачность	Свойство	1	this_comp.layer("Solid 1"). mask("Mask 1").opacity
feather	Смягчение краёв	Свойство	2	this_comp.layer("Solid 1"). mask("Mask 1").feather[0]
expansion	Расширение	Свойство	1	this_comp.layer("Solid 1"). mask("Mask 1").expansion
invert	Инвертирование	Булевское значение	1	

Атрибуты и методы свойства (Property attributes and methods)

Атрибуты и методы свойства	Смысл	Возвращаемый объект или тип объекта	Примеры
value	Возвращает значение свойства в данное время	Число или вектор	
value_at_time(t)	Возвращает значение свойства в специфическое время	Число или вектор	
velocity	Возвращает значение ускорения в данное время	Число или вектор	
velocity_at_time (t)	Возвращает значение ускорения в специфическое время	Число или вектор	
speed	Скорость в данное время. Этот элемент может быть использован	Число	

	только для пространственных свойств		
speed_at_time(t)	Скорость в специфическое время.	Число	
smooth(width=2, samples=5, t=time)	Добавляет боксовый фильтр к значению свойства в специфическое время и сглаживает результат через промежутки времени. Width - через какое время будут появляться образцы, samples число образцов, t - переменная.	Число или массив	position.smooth(1, 5)
temporal_wiggle (freq, amp, octaves=1, amp_mult=5, t=time)	Создаёт образцы свойства во времени. Freq – количество колебаний в секунду, amp - величина колебаний, octaves – число октав шума, amp_mult – величина на которую amp умножается для каждой октавы , t – базовое время)	Число или массив	scale.temporal_wiggle(5, .2)
wiggle(freq, amp, octaves=1, amp_mult=5, t=time)	Случайным образом «раскачивает» значение свойства. Freq – количество колебаний в секунду, amp - величина колебаний, octaves – число октав шума, amp_mult – величина на которую amp умножается для каждой октавы , t – базовое время)	Число или массив	position.wiggle(7, 30, 3)
loop_in(type = "cycle", num_keyframes = 0)	См. *«Нововведения в AE 5.5» >>> «Использование новых методов зацикливания ключевых кадров»*		
loop_out(type = "cycle", num_ keyframes = 0)			
loop_in_duration (type = "cycle", duration = 0)			
loop_out_duration (type = "cycle", duration = 0)			
key(index)	Возвращает ключевой кадр по номеру. Когда вы имеете доступ к объекту key, то вы можете извлечь из него свойства time, index и value.		position.key(3).value
key(marker_name)	Возвращает ключевой кадр по имени.		position.key(“Splash”).time
nearest_key(t)	Возвращает свойство ближайшего ключевого кадра		nearest_key(time)
num_keys	Возвращает число ключевых кадров свойства.		

Свойства и методы объекта Math в JavaScript (JavaScript Math)

Этот раздел почему-то проигнорирован в справочнике по языку выражений официального руководства. Он, правда, присутствует в разделе ***«Motion Math language reference»***, но и там он представлен, как-то очень сжато. Тогда я обратился за помощью к справочнику по **JavaScript**. Результаты моих изысканий вы можете увидеть ниже.

Во-первых, давайте определимся, что значит понятие *«Math»*.

Math – предопределённый объект, обеспечивающий возможность выполнения математических операций и доступ к математическим константам.

Объект **Math** делится на две части: свойства, содержащие константы и методы для реализации функций. Например, для получения значения числа Пи в уравнении используйте: **Math.PI**

Стандартные тригонометрические, логарифмические и экспоненциальные функции также включены в этот объект. Все аргументы тригонометрических функций выражаются в радианах. Также представлено несколько операций сравнения, например **max** - для определения большего из двух чисел.

Пи = 180^0
$\Rightarrow$ Пи/2=90^0

Свойства

Свойство	Описание	Применение
Math.E	Основание натуральных логарифмов	Также называется константой Эйлера, значение приблизительно равно 2.71828.
Math.LN10	Константа, представляющая натуральный логарифм числа 10	Значение этой константы приблизительно равно 2.30259
Math.LN2	Константа, представляющая натуральный логарифм числа 2.	Значение этой константы приблизительно равно 0.69315
Math.LOG10E	Константа, представляющая собой логарифм числа E по основанию 10	Значение этой константы приблизительно равно 0.43429
Math.LOG2E	Константа, представляющая собой логарифм числа E по основанию 2	Значение этой константы приблизительно равно 1.44270
Math.PI	Возвращает значение числа Пи	Значение числа PI приблизительно равно 3.14159. Это - отношение длины окружности к диаметру.
Math.SQRT1_2	Квадратный корень из 1/2.	Квадратный корень из 1/2 (приблизительно 0.707) может быть также представлен как величина, обратная квадратному корню из 2
Math.SQRT2	Квадратный корень из 2	Значение этой константы приблизительно равно 1.414

Методы

Метод	Описание	Применение
Math.abs(val)	Возвращает абсолютное значение своего аргумента	Math.abs (-10)
Math.acos(val)	Возвращает арккосинус своего аргумента (от 0 до Пи-радиан)	Аргумент должен быть числом в диапазоне между -1 и 1. Если

		значение выходит за пределы этого диапазона, возвращается 0.
Math.asin(val)	Возвращает арксинус своего аргумента	При передаче методу asin числа в диапазоне от -1 до 1 он возвращает арксинус аргумента (от -Пи/2 до Пи/2 радиан). Если передаваемый аргумент выходит за пределы указанного диапазона, возвращается 0.
Math.atan(val)	Возвращает арктангенс своего аргумента.	Метод atan возвращает число между -Пи/2 и Пи/2 радиан. Аргументом является число в диапазоне от -1 до 1, равное тангенсу возвращаемого значения
Math.atan2(y,x)	Возвращает угол между осью X и точкой, образованной аргументами.	Метод atan возвращает число между -Пи и Пи радиан. Y компонента передаётся первой.
Math.ceil(val)	Возвращает ближайшее целое число, большее или равное аргументу	Метод возвращает наименьшее целое число, большее или равное целому или дробному аргументу. Например: Math.ceil(1.01) возвращает 2.
Math.cos(val)	Возвращает косинус своего аргумента	Величина угла должна быть указана в радианах, возвращаемый результат будет находиться в диапазоне от -1 до 1.
Math.exp(val)	Возвращает значение экспоненты	Метод возвращает число E (константу Эйлера), возведенное в степень, равную аргументу
Math.floor(val)	Возвращает ближайшее целое число, меньшее или равное аргументу	Если этому методу передано в качестве аргумента целое или дробное число, он возвращает целое число, меньшее или равное аргументу. Например: Math.floor(2.99) возвращает 2.
Math.log(val)	Возвращает натуральный логарифм для положительного аргумента	Для отрицательного аргумента всегда возвращает -1.797693134862316e+308
Math.max(val1,val	Возвращает наибольший из двух	Принимает любую комби-

2)	аргументов	нацию числовых констант или переменных и возвращает значение большей.
Math.min(val1,val 2)	Возвращает меньший из двух аргументов	Принимает любую комбинацию числовых констант или переменных и возвращает значение наименьшего
Math.pow(val, exponent)	Возвращает основание, возведенное в степень	Возвращает аргумент $1^{\text{аргумент2}}$.

Math.round(val)	Округляет число до ближайшего целого	Округляет аргумент с плавающей точкой до ближайшего большего целого числа, если десятичная часть больше или равна 0.5, или до ближайшего меньшего целого числа, если десятичная часть меньше, чем 0.5.
		Math.round(2.1) // Возвращает 2
		Math.round(2.9) // Возвращает 3

Math.sin(val)	Возвращает синус аргумента	Аргументом является величина угла в радианах, возвращаемое значение может быть от -1 до 1.
Math.sqrt(val)	Возвращает квадратный корень из положительного числа	Если аргумент отрицателен, возвращается число 0
Math.tan(val)	Возвращает тангенс аргумента.	Аргументом является величина угла в радианах

Логические операции

Здесь приведены все логические операции языка **JavaScript**, которые вы можете использовать в своих выражениях.

&& и
|| или
! нет
== равно
!= не равно
< меньше чем
<= меньше чем или равно
> больше чем
>= больше чем или равно
+=прибавить и назначить
-+вычесть и назначить
*+умножить и назначить
/+разделить и назначить

Зарезервированные слова

В данном разделе приведён список ключевых слов языка **JavaScript**. Эти слова не могут быть использованы в качестве идентификаторов, имён функций или методов и имён объектов.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	