ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ.

Самостоятельная работа студентов - один из основных средств овладения учебным материалом во время, свободное от аудиторных учебных занятий.

Самостоятельная работа есть одной из составных учебного процесса, на которую приходится значительный объем учебного времени. При этом студент является активным участником учебного процесса, приобретает привычки самоорганизации, самостоятельного поиска информации, принятие решений и т.д. Правильная организация самостоятельной работы разрешает максимально индивидуализировать обучение, повысить эффективность учебного процесса в целом.

Главной целью самостоятельной работы есть закрепления, расширение и углубление приобретенных в процессе аудиторной работы знаний, умений и привычек, а также самостоятельное изучение и усвоения нового материала под руководством преподавателя, но без его непосредственного участия.

Одним из видов самостоятельной работы есть обработки практического материала, определение главного в содержании лекции, усвоение ее основных моментов. При этом не следует дословно записывать за лектором, а своими словами фиксировать важное: тему, ее основные вопросы и положения. Записывать следует аккуратно и четко, лучше в тетради в клеточку (через клеточку). На страницах конспекта оставлять широкие поля для дополнительных пометок во время самостоятельной работы над литературой.

Чтобы понять и хорошо усвоить материал, к каждой следующей теме следует тщательно готовиться: систематически обрабатывать предыдущий материал и, если это необходимо, обработать рекомендованную литературу, повторять пройденный материал.

Самостоятельная работа слушателей над учебной дисциплиной «Компьютерное обеспечение» для специальности «художественно – компьютерная графика» включает такие формы:

- самостоятельное изучение теоретических вопросов тем дисциплины, которые не были рассмотрены, или сжато рассмотренные на занятиях;
- самостоятельная работа с литературными источниками с целью лучшего усвоения программного материала после посещения занятий;
- окончательное выполнение задач практических занятий и оформление отчета о выполнении практических задач;
- подготовка реферата.

Все задачи самостоятельной работы слушателей являются обязательными, выполняются в установленные сроки, с соответствующей максимальной оценкой и предусматривают определенные формы отчетности относительно их выполнения. Обязательные задачи выполняются каждым без исключения студентом в процессе изучения дисциплины.

Вопросы, которые возникают у студентов относительно выполнения запланированных задач, решаются на консультациях, которые проводятся согласно графикам, утвержденным кафедре.

Преподаватель систематически контролирует самостоятельную работу студентов: проверяет конспекты первоисточников, выполнение задач творческого характера, предоставляет необходимую помощь для активизации учебной деятельности студентов.

II курс - 1 семестр дисциплины «Компьютерное обеспечение»

Тема 1. Знакомство с Flash

Учебные вопросы.

1. Что такое Flash?
2. Форматы файлов.
3. Стартовое меню.
4. Исследуем готовый фильм.
5. Интернет-ресурсы по Flash.

Вопрос для самостоятельной работы.

1. Что такое Flash?
2. Возможности технологии *Flash* .
3. Недостатки технологии *Flash.*
4. Форматы файлов, связанные с технологией *Flash.*
5. Опишите главное окно программы.
6. Для чего предназначенная панель инструментов.
7. Что такое временная шкала.
8. Как запустить просмотр со всеми эффектами.

Термины: Flash, расширения файлов Flash, сцена, публикация, экспорт файла.

Литература:[1 С.28-58, С.554-614 ; 2 С.47-58]

Тема 2. Контуры

Учебные вопросы.

1. Цвета
2. Линии
3. Выделение объектов
4. Перекрашивание контура
5. Направляющие
6. Изменение формы контура

Практика - Рисуем сердце.

7. Добавление заливания
8. Перо

Рисуем волну и закрашиваем ее.

Задача для работы. Рисуем сердце и волну. Рисуем прямоугольник карандашом и делаем из него *дельфина*.

Пример: Используя инструмент **Pen**, нарисуйте контур, который изображает профиль морской волны, как показано на рисунке. Закрасьте контур синими цветами с помощью инструмента **Paint Bucket.**

Вопрос для самостоятельной работы.

1. Какие инструменты рисуют только контуры.
2. Какие инструменты рисуют только заливание.
3. Какие инструменты рисуют контуры и заливания.
4. Какой параметр управляет прозрачностью.

5. Какие виды кадров существуют в Flash.
6. Что делают клавиши F5, F6, F7.

Термины: параметр Alpha, виды кадров, направляющие.

Литература:[1 С.62-86]

Тема 3. Покадровая анимация

Учебные вопросы.

1. Работа с кадрами
2. Трансформации объектов
3. Меню панели
4. «Луковая кожура»
5. Редактирование нескольких кадров
6. Практикум «Редактирование нескольких кадров »

Редактирование по кадровой анимации письмец и рост стебля с цветком посредине. Можно прибавить пчелку, которая летит, или ползучего жука.

Задача для работы – покадровая анимация «Рост цветка и персонаж, который двигается, » 6-8 кадров.

7. Кисти
8. Панели Color и Swatches
9. Практикум (кисти) - Медведь.

Задача для работы : *Покадровая анимация* Медведя на 6-8 кадров. Медведь моргает глазами. Появляется бочка меда. В бочке уменьшается мед.

***Пример** медведя*:

Вопрос для самостоятельной работы.

1. В чем суть покадровой анимации.
2. Что разрешает делать инструмент **Free Transform.**
3. Когда применяется **Onion Skin** («луковая кожура»).
4. Зачем нужен **Edit Multiple Frame.**
5. Что означает режим **Paint Behind** для инструмента **КИСТЬ.**

Термины: параметр Alpha, виды кадров, направляющие, режим Onion Skin, инструменте Brush и режимы его работы

Литература:[1 С.62-86, 168-177]

Тема 4. Геометрические фигуры

Учебные вопросы.

1. Геометрические фигуры
2. Режим слияния

3. Практикум (градиенты)

Задача для работы: - Рисуем логотип Aplle.

Пример:

Практика : Вставьте новый пустой ключевой кадр в кадр 2. Включите инструмент **Rectangle**, установите прозрачный контур. С помощью панели **Color** выберите цвета заливания с параметрами **R=0**, **G=116**, **B=179** и прибавьте его в палитру (пункт **Add Swatch** в меню палитры **Color**).

Практика : Проверьте, чтобы кнопка **Object Drawing** в нижней части панели инструментов не была нажата. Установите радиус скругления углов 20 и нарисуйте квадрат в верхней части сцены (рисунок 1).

Практика : Установите белые цвета заливания, радиус скругления 0, и нарисуйте внутри белый квадрат меньшего размера, который будет изображать яблоко (рисунок 2).

1 2 3 4

Практика : С помощью инструмента (добавление узла) прибавьте новые узлы в середину каждой стороны внутреннего квадрата.

Практика : Чтобы узлы при перетаскивании не «прилипали» к сторонам объектов, отключите флажок **View—Spnapping—Snap to Objects**. С помощью инструмента **Subselection** (клавиши **A**) переместите узлы так, как показано на рисунке 3, а потом (при нажатой клавише **Alt**) сделайте их тучными (рисунок 4).

Практика : Теперь нарисуем письмецо над яблоком. Постройте прямоугольник белых цветов без контура и превратите его в письмецо, перетаскивая и сглаживая узлы с помощью инструмента **Subselection** (рисунок 5).

Практика : Выберите из палитры темно-синие цвета заливания, такой же, как у первого квадрата (когда появится палитра, можно просто щелкнуть на нем мышью). Постройте круг без контура, который «вырезает» часть яблока (рисунок 6).

5 6

Практика : Удалите две белые области (яблоко и письмецо). Выделите все что остались (синие) области и объедините их в один объект, выбрав пункт меню **Modify—Combine Objects—Union**.

Практика **:** Выберите для обеих движков синие цвета, добавленный в палитру в начале упражнения. Для того, чтобы сделать цвета левого движка светлее,

выделите его и переместите треугольный черный движок (у правой границы панели) вверх. Этот движок регулирует яркость цветов (параметр **B**rightness в цветовой модели HSB).
Теперь фигура залита градиентом по левую сторону по правую сторону, нам же нужно, чтобы верхняя часть была света, а нижняя — темнее. Для **преобразования градиента** (размер, направление, угол поворота) служит инструмент **Gradient Transform**, что включается с помощью той же кнопки, что и **Free Transform**. Для быстрого включения можно также использовать клавишу **F**
Практика : Включите инструмент **Gradient Transform** и сделайте так, чтобы градиент шел сверху вниз.
Пракитка : Постройте отражение объекта. Для этого нужно скопировать его (перетянув при нажатой клавише **Alt**) и применить команду меню **Modify—Transform—Flip Vertical**. Перетянете копию точно под первый объект.
Практика : Включите инструмент **Free Transform** и уменьшите высоту копии приблизительно в 2 раза.
Практика : Перейдите на панель **Color** и измените прозрачность цветов градиента для отражения: установите **Alpha=0%** для левого движка и **Alpha=50%** для правого.

4. Режим рисования объектов
5. Фигуры с настраиванием
6. Практикум (лассо, фильтры)

Задача для работы: - Рисуем логотип Adobe CS3.
Пример:

Задача для работы: - Контрольная работа (логотип Microsoft)

Вопрос для самостоятельной работы.

1. Какой инструмент применяется для рисования овала.
2. Что такое градиент.
3. Какие виды градиентов можно выбрать в Adobe Flash CS3.
4. Что разрешает делать режим рисования фигуры с настраиванием.
5. Для чего употребится панель **Align.**

Термины: Привязка к объектам, градиент, фильтр, виды анимации, форма в анимации.

Литература:[1 С.62-86, 168-177]

Тема 5. Анимация формы

Учебные вопросы.

1. Введение

Задача для работы: Сделать ролик с анимацией формы.

Пример:

Практика: Выберите инструмент (прямоугольник), установите черные цвета контура, без заливания. Обязательно **отключите режим рисования объектов** (кнопка в нижней части панели инструментов не должна быть нажата). Нарисуйте в центре поля квадрат размером приблизительно 300 на 300.

Практика : Вставьте новые ключевые кадры в кадры 10, 20 и 30 (клавиши **F6**). Используя инструмент **Selection** и **Free transform**, измените контуры в кадрах 10 и 20 так, как показано на рисунках ниже. Кадр 30 оставьте без перемен (он совпадает с кадром 1).

Кадр 1,30 **кадр 10** **кадр 20**

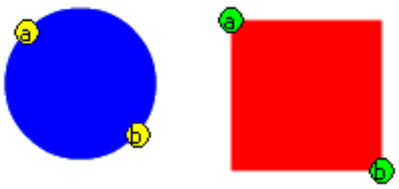
Практика : Чтобы включить анимацию формы для первого перехода, щелкните правой кнопкой мыши по кадру 10 и выберите из контекстного меню пункт **Create Shape Tween** (создать анимацию формы). Повторите то же самое для кадров 10 и 20.

2. Контрольные точки
3. Оптимизация контура
4. Цвета и движение

Задача для работы: Анимация формы – Круг переходит в квадрат , квадрат переходит у мотылька.

Пример:

Практика : Перейдите на кадр 1 и установите две контрольных точки так, как показано на рисунке 1. Потом перейдите на кадр 10 и переместите контрольные точки в противоположные углы квадрата (рисунок 2).



1 **2**

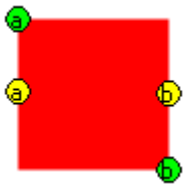
Практика : Прибавьте пустой ключевой кадр в кадр 20 и нарисуйте кистью в пределах квадрата с направляющий контур мотылька синих цветов (рисунок 3 ниже).

3

Практика : Прибавьте к кадру 10 анимацию формы (переход квадрата у мотылька).

Практика : Перейдите на кадр 10. Вы должны увидеть две зеленые контрольные точки в углах квадрата (от предыдущей анимации). Если вы их не видите, выберите пункт меню **View—Show Shape Hints**. Прибавьте две контрольных точки и перетянете их на середины сторон квадрата (точки желтых цветов на рисунке 4).

Практика : Перейдите на кадр 20 и перетянете контрольные точки на контур мотылька, как показано на рисунке 5. Просмотрите ролик - анимация должна улучшиться.



4

5

5. Слои
6. Звук
7. Текст

Задача для работы - сделать анимацию текста ИМЕНИ на фоне с градиентным заливанием.

Пример: Сначала появляется первая буква имени, потом друга букв. С появлением второй буквы имени первая исчезает. Дальше появляется третья буква, вторая при этом исчезает. И так далее.

Чтобы использовать анимацию формы, текст надо превратить в форму, т.е. у заливание. Для этого дважды применим команду **Break Apart** из контекстного меню или с меню **Modify** (клавиши **Ctrl+B**).

В кадре 1 сделайте (с помощью панели **Color**) все буквы полностью прозрачными (**Alpha=0**).

Перейдите в кадр 10. Увеличьте первую букву ровно в 2 раза (меню **Modify—Transform—Scale and Rotate—200%**), а другие буквы сделайте прозрачными.

Аналогично в кадрах 20, 30 и в зависимости от количества букв имени сделайте ,соответственно одни буквы видимыми, а другие невидимыми.

8. Слои-маски

Задача для работы : Студент делает слайд-шоу с 4 фото, применяет слои-маски.

Вопросы для самостоятельной работы.

1. Какие виды автоматической анимации применяются в Adobe Flash CS3.
2. Что разрешает делать анимация формы.
3. Зачем применяют контрольные точки на форме
4. Какие форматы звуковых файлов разрешает прибавлять Adobe Flash CS3.
5. Какие типы текста предлагает Adobe Flash CS3.
6. Для чего нужен Слой-Маска.

Термины: контрольные точки, слои, типы текста, маска.

Литература:[1 С.87-99,148-159,168-189,386-399 ; 2 С.180-217]

II курс - 2 семестр дисциплины «Компьютерное обеспечение»

Тема 6. Символы. Анимация движения

Учебные вопросы.

1. ***Символы***
2. ***Анимация движения***
3. ***Изменение символа при анимации***
4. ***Направляющие***
5. ***Обращение***

Задача для работы: Нарисовать баскетбольный мяч, превратить его в символ. Создать анимационный ролик так, чтобы мяч вылетал по левую сторону, ударялся об подлога и летел вправо за экран. По законам физики, мяч летит вниз ускоренно, а вверх — замедленно. В момент удара мяч меняет форму («сплющивается»). Необходимо прибавить **тень**, которая падает от мяча. В полете мяч должен оборачиваться.

Пример:

Вид мяча.

По законам физики, мяч летит вниз ускоренно, а вверх — замедленно. В программе *Flash* это можно сделать с помощью параметра *Ease* на панели **Properties**. Кнопка **Edit** по правую сторону от поля **Ease** разрешает точно настраивать кривую изменения скорости анимации.

Практика : Перейдите в кадр 1 слоя **Мяч** и установите значение **Ease** равное **-100**. В кадре 10 установите этот параметр равным **100.**

Сейчас мы никак не учитываем, что в момент удара мяч меняет форму («сплющивается»). Если просто уменьшить высоту символа в кадре 10, то мяч начнет «сплющиваться» из самого верха, а в самом деле это не так — быстрое изменение формы идет только при ударе.

Чтобы смоделировать изменение формы мяча в момент удара, мы прибавим приложению кадры анимации.

Практика : Выделите кадры 10-20 слоя *Мяч* и перетянете их по временной шкале вправо на 4 кадры. Скопируйте (при нажатой клавише **Alt**) кадр 14 в кадры 10 и 12. Прибавьте новые промежуточные кадры в пластов *Фон* и *Звук* так, чтобы длина временных шкал всех пластов совпадали.

Практика : Включите инструмент  (клавиши **Q**). Перейдите в кадр 10 слоя *Мяч* и перетянете белый кружок (центр обращения) на середину нижней границы символа. Сделайте то же самое для кадра 14.

Практика : Перейдите в кадр 12 слоя *Мяч* (он должен быть ключевым!). Немного уменьшите высоту мяча, передвинув его верхнюю границу.

Теперь прибавим **тень**, которая падает от мяча. Будем припускать, что осветительные лампы расположенные сверху, поэтому

- тень падает вертикально вниз от мяча;
- самая темная и резкая тень - в нижней точке;
- при подъеме мяча тень размывается и увеличивается.

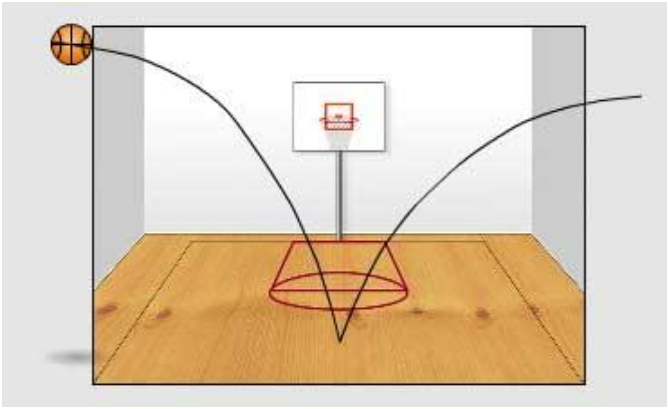
Для создания тени будем использовать полупрозрачные овалы, залитые черными цветами.

Практика : Прибавьте пласта *Тень* между пластами *Фон* и *Мяч*. Перейдите на кадр 1 и нарисуйте овал черных цветов без контура. Выделите это заливание и на панели **Colors** установите радиальный градиент, оба цвета — черные, но у левого движка (в центре) параметр **Alpha** равняется **30%**, а в правого — **Alpha=0%** (заливание полностью прозрачное). С помощью инструмента  **Gradient Transform** настройте градиент так,

как показано на рисунке.
Теперь построим тень для других кадров:

- тень в кадре 24 точно такая же, как и кадре 1;
- самая темная и самая маленькая тень — в кадрах 10-14 (мячик в нижней точке);
- для изменения тени между кадрами 1-10 и 15-24 используем *анимацию формы*.

Практика : Перейдите в кадр 2 слоя *Тень* и вставьте новый ключевой кадр (**F6**).
Пратика : Вставьте новый ключевой кадр в кадр 10 этого же пласта (клавиши **F6**).
Уменьшите овал, который изображает тень от мяча в момент прикосновенья пола.
Измените заливание: используйте градиент с переходом от **Alpha=90%** через **Alpha=60%** (в середине) к **Alpha=0%**. Для всех движков устанавливается черные цвета.
Пратика : Включите анимацию формы для кадров 1 и 14 слоя *Тень*.
Теперь изменим ролик так, чтобы мяч вылетал по левую сторону, ударялся об подлога и летел вправо за экран.
Практика :Перейдите на кадр 1 слоя *Мяч*. Выделите сразу мяч и тень (надо обвести их мышкой при нажатой клавише **Ctrl**) и переместите мяч влево за границы поля. В кадре 24 таким же способом переместите мяч с тенью за правую границу.
Практика : Перейдите в кадр 12 и включите режим редактирования соседних кадров, щелкнув на кнопке под временной шкалой. Установите движки в верхней части шкалы так, чтобы они захватывали кадры 10-14. Выделите мышкой изображения мяча и тени и перетянете в середину сцены. Уменьшите частоту кадров до 12 и просмотрите ролик.
Практика: Выделите пласта *Мяч* и щелкните по кнопке (**Add Motion Guide**, создать пласта направляющих) ниже списка пластов.
Практика : Перейдите в кадр 1 слоя *Guide: Мяч*. Включите инструмент / и проверьте, чтобы режим рисования объектов был отключен (кнопка / в нижней части панели инструментов не должна быть нажата). Нарисуйте кривую приблизительно так, как показано на рисунке ниже.



Практика : В кадрах 10, 12 и 14 установите мяч так, чтобы центр обращения совпал с самой нижней точкой траектории. В кадре 24 также переместите мяч на траекторию.
Практика : Выделите кадр 1 слоя *Мяч* и прибавьте к анимации обращения — 2 оборота по часовой стрелке. То же самое сделайте для кадра 14. Чтобы лучше увидеть результат, уменьшите частоту кадров до 5.

6. ***Растровые рисунки***
7. ***Ориентация вдоль пути***

Задача для работы. Нарисовать машину, дорогую, газон, дом. Создать анимацию движения машины по траектории дороги, которая петляет.

Пример:

Практика : Создаем новый пласта «*Фон*» и рисуем фон.
Практика : Создаем новый пласта «*Дорога*» и рисуем серпантин дороги (петляющую дорогую).
Практика : Создаем новый пласта «*Путь*» и рисуем с помощью инструмента  линию пути, которая совпадает с серединой дороги и повторяет все повороты.
Практика : Создаем символ Graphic «*Машина*», и рисуем машину вид сверху.
Практика : Создайте новый пласта с именем *Машина* и перетаскиваем на него символ *Машина* из библиотеки. Поставьте машину на дорогую на правый край сцены, уменьшите ее так, чтобы она поместилась на дорогую, и разверните вдоль дороги в направлении к дому.

Чтобы развернуть машину более точно, иногда приходится снимать флажок **View—Snapping—Snap to Objects**, отменяя привязку к существующим объектам.
Практика : Вставьте новый ключевой кадр в кадр 70 на пласте *Машина*. Вставьте промежуточные кадры в кадр 70 всех других пластов (изображение в них не меняется).
Практика : На пласте *Машина* в кадре 70 расположите машину на дороге возле дома и разверните в нужном направлении (как она должна приехать). Прибавьте анимацию движения к кадру 1 слоя *Машина*. Просмотрите ролик.
Конечно, машина двигается неправильно. Надо связать ее движение с пластом направляющих. На этот раз мы не будем создавать новый пласт, а используем пласта *Путь*, который именно и содержит контур пути.
Практика : Перетянете пласта **Путь** на самый верх в списке пластов (прямо над пластом *Машина)*. Нажмите правой кнопкой мыши на названии слоя *Путь* и выберите вариант **Guide** (направляющая).
Практика : Нажмите правой кнопкой мыши на шары *Машина*, выберите **Properties** (свойства) и отметьте вариант **Guided** (направляемый). Просмотрите ролик.
Теперь машина двигается по нужной траектории, но неправильно - она двигается то боком, то задом. Конечно, можно было поставить между конечными точками еще несколько ключевых кадров, в которых развернуть машину правильно. Однако можно одним щелчком заставить машину принимать нужен направление на всей траектории.
Практика : Выделите первый кадр слоя *Машина* и отметьте флажки **Orient to Path** (ориентировать относительно пути) и **Snap** (привязать объект к пути) на панели **Properties**. Чтобы исключить дополнительное обращение, установите в окне **Rotation** вариант **None** (нет обращения). Снова посмотрите ролик.

В конце концов, сделаем так, чтобы машина немного постояла была дома прежде чем начнется новый цикл проигрывания ролика.
Практика : Выделите мышкой кадр 100 **во всех пластах** и прибавьте промежуточный кадр (клавиши **F5**). Просмотрите окончательный результат.

8. ***Вложенная анимация - студент рисует все сам.***
9. ***Изменение скорости анимации***

Задача для работы : Студент рисует свою машину, пейзаж. Создает анимацию движения машины по траектории. При подъеме в горку движение автомобиля замедляется. *Колеса*

автомобиля должны оборачиваться. Для создания такой анимации нужно создать символ *Машина*, составной частью которого будут два одинаковых (вложенных) клипы *Колесо*, которые имитируют вращающиеся колеса.

Пример :

Для создания такой анимации нам нужно создать символ *Машина*, составной частью которого будут два одинаковых (вложенных) клипы *Колесо*, которые имитируют вращающиеся колеса.

Таким образом, **один клип расположен внутри другого**, причем каждый из них имеет свою временную шкалу: для клипа *Машина* установлена анимация движения в виде движения по заданной траектории, клип *Колесо* обеспечивает вложенную анимацию — обращение колес.

Практика : Переименуйте пласта *Layer 1* в Фон и нарисуйте фоновый рисунок. Заблокируйте пласта от изменений.

Практика : Создайте новый пласта *Путь* и нарисуйте на нем с помощью инструмента  траекторию движения машины.

Практика : Создайте новый символ *Машина* (клавиши **Ctrl-F8**), выберите тип символа **Movie Clip**. Нарисуйте корпус машины .

Для того, чтобы колеса оборачивались, нам нужно вместо простого графического символа *Колесо* использовать **клип** (символ типа **Movie Clip**). После этого мы прибавим анимацию (обращение) **внутри** этого клипа.

Практика : Создайте новый символ (**Ctrl+F8**) типа **Movie Clip** (клип) с именем *Колесо* Нарисуйте на поле клипа *Колесо* так, чтобы центр обращения оказался в том же месте, что и крестик (точка регистрации).

Практика : В кадре 30 клипа *Колесо* вставьте новый ключевой кадр и включите анимацию движения для кадра 1. На панели **Properties** установите обращение (**Rotation**) по часовой стрелке (**CW**) на 2 оборота (**2 times**). Просмотрите результат.

Практика : Откройте клип *Машина* в режиме редактирования. Выберите со списка клип *Колесо*. Состыкуйте Колесо с корпусом. В такой же способ вставьте второе колесо.

Практика : Возвратитесь к редактированию сцены. Создайте пласта *Машина* и прибавьте в кадр 1 машину из библиотеки так, чтобы она оказалась на красной линии по левую сторону от видимой области.

Практика : Сделайте так, чтобы пласт *Путь* стал пластом направляющих для пласта *Машина*. Для этого надо в свойствах *Путь* (правая кнопка мыши — **Properties**) выбрать вариант **Guide**, а в свойствах *Машина* установить **Guided**.

Практика : В кадре 80 пласта *Машина* вставьте новый ключевой кадр (**F6**) и расположите машину по правую сторону от видимой области так, чтобы ее центр обращения (белый кружок) оказался на направляющей линии. В кадре 80 других пластов вставьте новые промежуточные кадры (**F5**).

Практика : Для кадра 1 слоя *Машина* включите анимацию движения и отметьте флажки **Orient to Path** и **Snap** на панели **Properties**.

Если перейти в кадр, где включенная анимация движения, и щелкнуть по кнопке **Edit** в

панели **Properties**, мы увидим линию, которая задает скорость движения вдоль траектории. На оси абсцисс отмеченные номера кадров, а на оси ординат - процент пройденного пути.
Практика : Перейдите в кадр 1 пласта *Машина* и измените кривую.
Чтобы движение автомобиля выглядело более натурально, прибавим падающую тень.
Практика : Откройте символ *Машина* в режиме редактирования. Прибавьте новый пласта *Тень* и расположите его ниже основного пласта *Layer 1*. Нарисуйте тень в виде заливания черных цветов с прозрачностью 50%.

10. Анимация текста

- изменение размера текста,

Задача для работы: Построить анимацию, в ходе которой текст в виде цифр **2014**будет увеличиваться, начиная из центра циферблата. *Во второй половине фильма* циферблат затенить, перекрыв его черным прямоугольником, для которого будет изменять параметр **Alpha** от **0%** (полностью прозрачный) до **80%**.

Пример:

Практика : Создайте пласта *Текст* и поместите на него текст **2014**: шрифт *Arial*, жирный, размер 96, дополнительный интервал между буквами 4, цвета **R=255** (красный), **G=229** (зеленый), **B=172** (синий). Для точного введения цветов используйте кнопку / в правому верхнему кованые палитры.
Практика : Примените к тексту фильтры **Drop Shadow** и **Bevel**. Для фильтра **Bevel** установите специальные параметры:

- тип (*Type*): **Inner** (внутренний);
- сила (*Strength*): **40%**;
- дистанция (*Distance*): **-14**.

Практика : Превратите текст в символ с именем *2014* (клавиши **F8**).

Теперь построим анимацию, в ходе которой текст будет увеличиваться, начиная из центра циферблата. Отведем на это увеличение 6 секунд (72 кадра при частоте 12 кадров в секунду), а потом зафиксируем надпись еще на 3 секунды.
Практика : На пласте *Текст* вставьте в кадры 72 ключевой кадр. Потом прибавьте в кадр 108 обеих пластов промежуточный кадр
Практика : Перейдите в кадр 1 слоя *Текст* и уменьшите текст, расположив его в центре циферблата. Включите анимацию движения для кадра 1.
Практика : Создайте пласта *Затенения* между фоном и текстом и нарисуйте на нем черный прямоугольник, который перекрывает всю сцену. Прибавьте в этом пласте ключевой кадр в кадр 72. В кадре 1 установите с помощью панели **Color** прозрачные цвета (**Alpha=0**), а в кадре 72 — **Alpha=80%** (почти непрозрачный). Включите для кадра 1 анимацию формы (**Shape Tween**).

- побуквенная анимация текста «Лугань»

Задача для работы: Построить анимацию, в ходе которой ***Темные буквы*** влетают с правой стороны сцены. Выстраиваются в слово **Лугань** посреди Фона. Дальше, используя фильтр **Bevel**, сделать так, чтобы буквы постепенно становились золотистыми.
Пример:

Практика : Создайте новый пласта *Текст*, введите в нижней части сцены текст ***Лугань*** (шрифт *Arial*, черный, жирный, размер 70) и выровняйте его по середине сцены (панель **Align**, включить режим **To Stage**).

Теперь мы разобьем текст на отдельные буквы и разместим каждую из них на отдельном пласте.

Практика : Выделите текст и примените команды **Break Apart** и **Distribute to Layers** из контекстного меню. Удалите пласта *Текст* (он теперь пустой).

Программа создала несколько новых пластов, их имена совпадают с буквами текста.

Практика : Выделите кадр 10 во всех пластах с буквами и прибавьте новый ключевой кадр (**F6**). Потом выделите кадр 1 во всех слоях-буквах (выделятся все буквы) и клавишей «вправо» переместите буквы за правую границу сцены (нажатие клавиши **Shift** ускоряет процесс).

Практика : Включите анимацию движения для всех слоев-букв, вставьте новый промежуточный кадр в кадр 10 слоя *Фон* и просмотрите результат.

Если посмотреть на библиотеку, можно заметить, что у нее добавленные новые символы, созданные автоматически из букв. Это необходимо программе для построения анимации движения.

Практика : Выделите кадры 1-10 слоя, где находится буква **в**, и перетянете вправо, так чтобы анимация начиналась из кадра 6. Сдвиньте и другие пласты с буквами так же, на 5 кадров от предыдущей буквы. Выделите кадр 35 всех слоев-букв и вставьте новый ключевой кадр, в тот же кадр фонового пласта вставьте промежуточный кадр. Просмотрите результат.

Дальше, используя фильтр **Bevel**, сделаем так, чтобы буквы постепенно становились золотистыми. Однако применить фильтры можно только к тексту или символам типа **Movie Clip**. Сейчас буквы имеют тип **Graphic**, это можно увидеть на панели **Properties**. Поэтому нужно будет изменить тип символов на **Movie Clip.**

Практика : Перейдите в кадр 35, выделите все буквы и на панели **Properties** измените тип символа на **Movie Clip**.

Сейчас все готово для создания анимации, которая сводится к постепенному применению фильтра.

Практика : Выделите кадр 50 во всех слоях-буквах и вставьте новый ключевой кадр. Не снимая выделение из букв, перейдите на панель **Filters** и примените фильтр **Bevel**, изменив некоторые параметры:

- цвета тени (*Shadow*): **R=116**, **G=89**, **B=78**;
- цвета светлых областей (*Highlight*): **R=253**, **G=202**, **B=139**;
- дистанция (*Distance*): **-6.**

Прибавьте анимацию движения ко всем пластов с буквами в кадре 35.

Остается сделать небольшую паузу перед началом нового цикла.

Практика : Вставьте промежуточные кадры в кадр 70 во всех пластах

Вопрос для самостоятельной работы.

1. Что такое Символ?
2. Какие три типа символов различают в программе *Flash* ?
3. Как задать нестандартную траекторию движения объекта?

Термины: Символы, *слои, типы текста*

Литература:[1 С. 100-120,168-189; 2 С.277-324]

Тема 7. Знакомство с ActionScript
(продолжительность 14 часа)

Учебные вопросы.

1. ***Зачем нужен ActionScript?***
2. ***Сменные***
3. ***Массивы***
4. ***Условные операторы***
5. ***Циклы***
6. ***Функции***
7. ***Управление проигрыванием. Кнопки***

Задача для работы: В **Теме 6** Студент рисовал свою машину, пейзаж. Создавал анимацию движения машины по траектории. При подъеме в горку движение автомобиля замедлялось. *Колеса автомобиля оборачивались*. Теперь нужно прибавить кнопки управления роликом. ***Прибавить кнопки «Стоп», «Пуск», «Перейти в начало». ».***
Машина должна останавливаться перед подъемом.
Пример:
Практика : Откройте файл.
Практика : Перейдите кадр слоя *Машина* , где начинается подъем, и нажмите клавишу **F9**, чтобы вывести на экран панель **Actions**. Введите код:
stop();
Просмотрите результат.
Для того, чтобы продолжить проигрывание фильма, мы прибавим на сцену кнопку - символ специального типа.
Кнопки
При создании символа можно выбрать тип **Button** (кнопка). Так называется клип, который имеет 4 кадры:

- **Up** — обычное состояние кнопки;
- **Over** — вид кнопки, когда на нее приведенный курсор мыши;
- **Down** — нажатая кнопка;
- **Hit** — область реагирования.

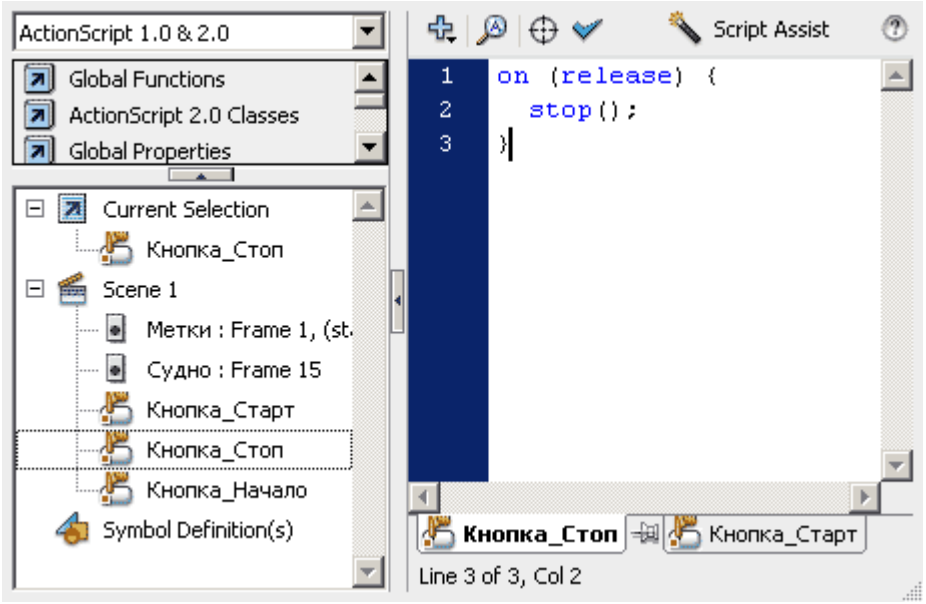
В каждом из них могут быть любые рисунки. Больше того, можно создавать многослойные изображения и прибавлять звук для каждого с состояний.
Практика : Создайте новый символ (клавиши **Ctrl+F8**) типа **Button** (кнопка) и назовите его *Кнопка_Старт*. Нарисуйте приблизительно вот такую кнопку:  Вставьте новый ключевой кадр в кадре *Over* и измените цвета стрелки на светло-зеленый. Потом вставьте ключевой кадр в кадре *Down* и измените цвета стрелки на красный.
Мы не задали никакого изображения в кадре *Hit*, это значит, что областью реагирования есть вся непрозрачная область последнего кадра *Down*.
Практика : Возвратитесь к сцене. Создайте новый пласта *Кнопки*, выделите кадр 1, перетянете кнопку из библиотеки в нижнюю часть поля и отрегулируйте ее размер.

Реакции на событии
Чтобы кнопка «заработала», нужно определить ее реакцию на нажатие. Если выделить кнопку и нажать клавишу **F9**, на экране появляется панель **Actions** (действия), где можно написать программный код, связанный с кнопкой. Вот пример такого кода:

```
on ( release ) {
  play();
}
```

Слово **on** начинает обработчик события, **release** (*освобождение*) в дужках указывает самое событие — отпускание мыши над кнопкой. Дальше внутри фигурных дужек записывают команды, которые нужно выполнить: в этом случае - продолжить проигрывание фильма.

Язык *ActionScript* использует правила записи команд, очень похожие на язык Си. Каждая команда заканчивается точкой из комы.
Нажав на кнопку (или клавиши **Ctrl+T**), можно проверить правильность записи кода (соответствие правилам языка *ActionScript*). Если вы увидели сообщение *"The script contains no errors."*, ошибок нет. Сообщение об ошибках выводят в специальном окне **Compiler Errors** (оно появляется автоматически или может быть вызвано нажатием клавиш **Alt+F2**).
Кнопка разрешает закрепить активное окно с кодом. Дело в том, что код может быть разбросан по нескольким местам (кадрам, кнопкам, клипам). В активном окне мы видим тот блок, который выделен в левой части панели **Actions**. Если прикрепить какой-либо блок, он будет всегда показываться как вкладыш в основном окне:



На рисунке активное окно — это код, связанный с кнопкой *Стоп*, а код кнопки *Стоп* был прикреплен и показывается в виде вкладыша.
Практика : Выделите кнопку и прибавьте в окне **Actions** код
on (release) { play(); }
Щелкните по кнопке / , чтобы проверить правильность набранного кода, а потом - по кнопке / для автоматического выравнивания.

В левой части панели **Actions** находится справочная информация (*Toolbox*). Сверху — списки функций и операторов языка (их можно вставлять в код щелчком мыши), ниже — список всех объектов, с которыми связанный код *ActionScript*. Сейчас видно, что какой-либо код есть в кадре слоя *Машина* и у кнопки (элементы блока *Scene 1*). В этот момент мы редактируем код кнопки (*Current Selection*). Это окно удобно использовать для быстрого перехода от одного блока кода к другому.
С помощью кнопки можно забрать панели *Toolbox*, оставив только редактор кода.
Практика : Выделите кадр 1 и прибавьте к нему код
stop();
Закройте панель *Actions* и проверьте работу фильма.

Теперь прибавим еще 2 кнопки: для остановки фильма и для перехода в начало. Поскольку на них должен быть другой рисунок, в библиотеке нужно создать еще две символа-кнопки. Например, можно продублировать существующую кнопку и изменить рисунок, выбрав команду **Duplicate** из контекстного меню, которая появляется при нажатии правой кнопки мыши на названии символа в библиотеке.

Практика : Создайте кнопку с именем *Кнопка_Стоп* для остановки проигрывания. Для этого продублируйте кнопку в библиотеке, дайте ей новое имя и исправьте рисунки во всех кадрах (*Up*, *Over*, *Down*). Прибавьте новую кнопку на пласта *Кнопки* и настройте так, чтобы она останавливала фильм.

Практика : Прибавьте еще одну кнопку , что возвращает фильм в начало (к кадру 1) с помощью кода

```
on (release) { gotoAndPlay(1); }
```

Проверьте ролик.

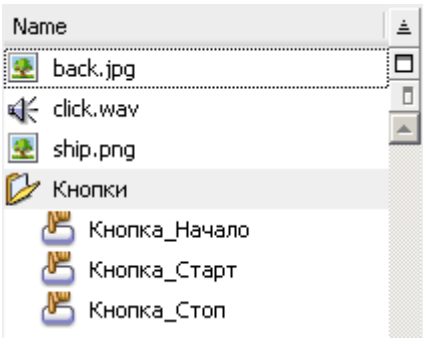
Практика : Измените команду, связанную с последней кнопкой, на

```
gotoAndStop(1);
```

и посмотрите, что изменилось.

Когда в библиотеке очень много элементов, список выходит очень длинный и в нем сложно разобраться. Чтобы облегчить жизнь, подобные элементы объединяют у **папки.** Папку в библиотеке можно открывать и закрывать двойным щелчком мыши.

Практика :Выделите все кнопки в библиотеке (щелкая на них при нажатой клавише **Ctrl**) и выберите пункт **Move to New Folder** (переместить в новую папку) из контекстного меню. Дайте папке имени *Кнопки*.



Теперь надо сделать, чтобы кнопки на сцене были одинакового размера, и выровнять их. Для этого употребится панель **Align** (выравнивание, клавиши **Ctrl+K**). Проверьте, чтобы режим *To Stage* был отключен (он употребится для выравнивания относительно всей сцены).

Практика : Выделите все кнопки и включите панель **Align**. Сначала сделайте одинаковой высоту кнопок (в группе *Match Size*). Потом выровняйте центры по вертикалу (в группе *Align*) и распределите равномерно по горизонтали (в группе *Distribute*).

Для кнопок, так же, как и для клипов, можно применять фильтры (панель **Filters**).

Практика : Для придания кнопкам больше красивого вида примените фильтр **Bevel** и настройте его параметры на ваш вкус.

Метки кадров

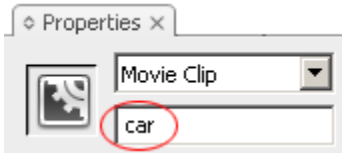
В ходе работы кадры часто перемещаются, поэтому переходы по номерам кадров нежелательные. Вместо этого кадра (только **ключевому**!) можно присвоить имя (**метку**) и выполнять переход по метке, а не по номеру. Для меток обычно создается отдельный пласт.

Практика : Создайте новый пласта *Метки* и выделите кадр 1. На панели **Properties** введите метку кадра, как показано на рисунке.

Замените команду, связанную с последней кнопкой, на
gotoAndStop ("start");
и проверьте ролик.
Обратите внимание, которое на временной шкале в кадре 1 слоя *Метки* появилась метка *start*:

8. ***Адреса: дети, родители и корень***
9. ***Звук: новые детали***

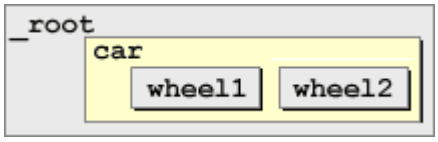
Задача для работы: В ролике где прибавляли кнопки управления роликом - ***«Стоп», «Пуск», «Перейти в начало».*** Прибавьте в код пласта *Программа* команды, которые останавливают обращение колес, а в код обработчика нажатия кнопки — команды для запуска обращения колес. Сохраните файл и просмотрите результат.
Пример:
Когда машина стоит на месте, но ее колеса оборачиваются. Это происходит потому, что команда stop() остановила только проигрывание главной временной шкалы, а внутренние клипы (колеса) продолжают работать.
Чтобы ответить на вопрос «Как остановить колеса?» нужно разобраться с адресами внутри флеш-ролика. Но прежде всего, дадим имена объектам на сцене, к которым мы будем обращаться: машине и двум колесам внутри нее.
Практика : Выделите машину и на панели **Properties** введите имя этого элемента car.
Откройте клип *Машина* для редактирования и в такой же способ дайте колесам имена wheel1 и wheel2.



Итак, в этом фильме 4 монтажных столы со своими временными шкалами:

- **главный** (корневой) монтажный стол, он называется _root;
- монтажный стол клипа **Машина**;
- два монтажных стола **для каждого колеса.**

Связи этих монтажных столов показанные на схеме:



Клип car находится внутри главного монтажного стола _root, а клипы wheel1 и wheel2 — внутри клипа car.
Главный монтажный стол _root есть **«отцом»** (*parent*) для клипа car.
Клип car есть **«сыном»** (*child*) для _root и отцом для клипов wheel1 и wheel2.
Клипы wheel1 и wheel2 — сыны для car, в них самых нет потомков.

Абсолютные адреса
Каждый объект имеет свой абсолютный адрес, по которому его можно найти из любого места. Например, квартира Васи Пупкина может иметь такой абсолютный адрес:

ЛНР, Луганск, ул. Оборонительная, буд. 5, кв. 25

Здесь на первом месте стоит страна, потом - город и т.д.
Для объектов *Flash-фильма* абсолютный адрес начинается с _root, так что абсолютные адреса всех клипов, показанных на схеме, выглядят так:
_root
_root.car
_root.car.wheel1
_root.car.wheel2
Как видим, для обращения к сынам употребится точка.
Абсолютные адреса не зависят от того, внутри какого монтажного стола они употребятся. Чтобы остановить проигрывание для какого-то монтажного стола, надо перед командой stop() поставить его адрес и точку:
_root.stop();
_root.car.stop();
_root.car.wheel1.stop();
_root.car.wheel2.stop();
Если мы применяем абсолютные адреса, нужно помнить, что они станут неверными, если машина будет находиться уже не на главном монтажном столе, а станет частью другого клипа (Вася Пупкин переехал).
Относительные адреса
Относительные адреса зависят от того, где находится объект, который использует этот адрес. Вася Пупкин может сказать, что Коля Иванов живое через 2 дома **от него**.
Для обращения к отцу используют слово _parent. Например, чтобы из клипа wheel1 остановить проигрывание для монтажного стола клипа car, надо применить команду
_parent.stop();
Эта команда сработает из из клипа wheel2, поскольку его отцом также есть car.
Если два раза использовать _parent, мы обращаемся к «деду». Например, команда
_parent._parent.stop();
останавливает проигрывание на главном монтажном столе из клипа wheel1 или wheel2.
К сынам можно обращаться просто по имени. Например, команда
car.stop();
допустимая внутри _root. Кроме того, здесь можно было использовать и слово this (текущий монтажный стол):
this.car.stop();
Для остановки wheel1 с _root можно применить одну из команд
car.wheel1.stop();
this.car.wheel1.stop();
В конце концов, самый сложный случай. Обратиться к «брату» можно только через отца. Чтобы остановить второе колесо с wheel1, надо написать
_parent.wheel2.stop();

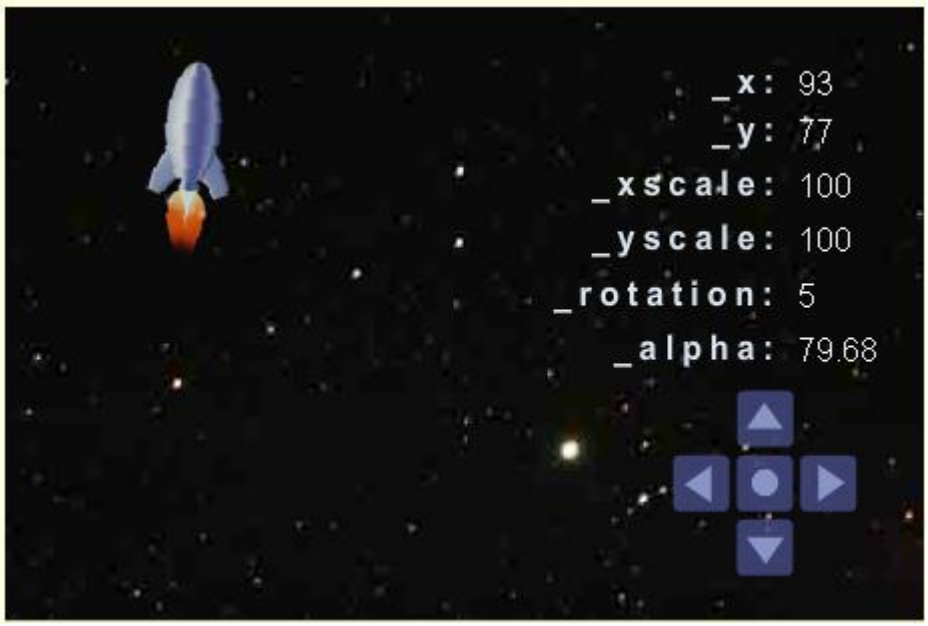
10. Свойства и события клипа

Задача для работы: Построить ролик в котором кнопки со стрелками на поле изменяют масштаб и угол поворота ракеты.
В правой части поля выводят текущие **свойства** клипа *Ракета*:

- _x, _y — координаты точки регистрации (отсчитываются от левого верхнего угла экрана);
- _width, _height — ширина и высота клипа в пикселях;
- _xscale, _yscale — масштабы по осям **X** и **Y** (равные 100 в исходном состоянии);

- _rotation — угол поворота в градусах вокруг точки регистрации (равняется 0 для клипа в библиотеке);
- _alpha — степень непрозрачности (от 0 до 100).

Прикла:
Построить ролик, в котором на фоны темного неюа находится ракета, как показано ниже.



Здесь ракету можно перетаскивать мышкой и перемещать клавишами-стрельцами на клавиатуре. Кнопки со стрелками на поле изменяют масштаб и угол поворота ракеты. Если водить мышью над ракетой, она постепенно становится прозрачной. Чтобы снова сделать ее полностью непрозрачной, надо щелкнуть по центральной кнопке с кружком.
В правой части поля выводят текущие **свойства** клипа *Ракета*:

- _x, _y — координаты точки регистрации (отсчитываются от левого верхнего угла экрана);
- _width, _height — ширина и высота клипа в пикселях;
- _xscale, _yscale — масштабы по осям **X** и **Y** (равные 100 в исходном состоянии);
- _rotation — угол поворота в градусах вокруг точки регистрации (равняется 0 для клипа в библиотеке);
- _alpha — степень непрозрачности (от 0 до 100).

Изменение свойств
Практика: Откройте файл space.fla из папки PRACTICE\7. Переместите точку регистрации клипа *Ракета* на ось симметрии ракеты (для этого надо открыть клип для редактирования и переместить изображение так, чтобы крестик оказался в нужном месте).
Практика: Прибавьте пласта *Кнопки* и разместите на нем 1 кнопку типа *Кнопка0* и 4 кнопки типа *Кнопка* из библиотеки. С помощью панели **Transform** (меню **Windows—Transform** или клавиши **Ctrl+T**) разверните три кнопки со стрелками на 90, 180 и 270 градусов. Расположите кнопки ровно, так как на образце в начале раздела.
Центральная кнопка с кружком будет восстанавливать полную непрозрачность ракеты, кнопки «уліво» и «вправо» оборачивают ее (изменяют свойство _rotation), а кнопки «вверх» и «вниз» — изменяют размеры (свойства _xscale и _yscale).
В языке *ActionScript* (так же, как в *Си* и *Java*) для операций увеличения или уменьшение значений сменных на некоторую величину часто употребится сокращенная запись:
a = a - 5;
b = b + 10;
выполняется точно так же, как
a -= 5;

b += 10;

Практика: Прибавьте код для кнопок. Центральная кнопка:

```
on (release) {
  rocket._alpha = 100;
 }
```

Стрелка «уліво»:

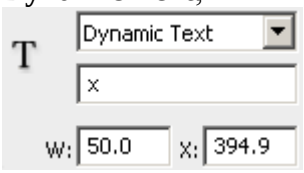
```
on (release) {
  rocket._rotation -= 10;
 }
```

Стрелка «вправо» - то же самое, но со знаком «плюс». Стрелка «вниз»

```
on (release) {
  rocket._xscale += 5;
  rocket._yscale += 5;
 }
```

Стрелка «вверх» - то же самое, но со знаком «минус». Просмотрите результат и сохраните файл.

Практика: Прибавьте текст **_x:** и по правую сторону от него — еще одно текстовое поле для вывода текущей координаты _x. Для него в панели **Properties** установите тип **Dynamic Text**, имя **x** и ширину **W=50**.



События клипа

Клипы (символы типа **Movie Clip**) могут реагировать на те же события, что и кнопки, с помощью обработчиков on (...). Кроме того, есть еще несколько особых событий, которых у кнопок нет. Наиболее важные из них:

- load — клип полностью загружен;
- enterFrame — состоялся переход к следующему кадру клипа;
- mouseDown — нажатая кнопка мыши;
- mouseMove — мышь переместилась;
- mouseUp — кнопка мыши отпущена;
- keyDown — нажатая клавиша на клавиатуре;
- keyUp — клавиши отпущенная.

Для обработки этих событий употребятся обработчики onClipEvent (...), которые можно прибавить на панели **Actions**, предварительно выделив клип.

При частоте кадров 12, событие enterFrame возникает 12 раз в секунду, даже если клип содержит единый кадр. Мы используем это событие для того, чтобы постоянно обновлять x-координату ракеты на экране.

Динамическое текстовое поле с именем x имеет свойство text, которых можно изменять из программы (это и есть надпись на экране). Для обращения к свойству объекта употребится точка, т.е. адрес текста принимает вид x.text. Таким образом, запись текущей x-координаты клипа rocket в текстовое поле x должна выглядеть так:

```
x.text = rocket._x;
```

Практика: Выделите ракету на сцене и откройте панель **Actions** (клавиши **F9**). Введите код обработчика события enterFrame:

```
 onClipEvent (enterFrame) {
    x.text = rocket._x;
 }
```

Просмотрите результат.
Вы увидели, что x-координата на экране как и раньше равняется нулю, хотя клип находится не на левой границе экрана, и значит его действительная x-координата больше нуля.
Чтобы проявить ошибку, мы применим **трассирование** — вывод (в дополнительное окно) отладочных сообщений, которые разрешат понять, что в самом деле происходит внутри программы.
Практика: Прибавьте в обработчик строка
trace (x);
При запуске появится окно **Output** (включается также через меню **Window—Output** или клавишей **F2**), в котором появляются строки undefined.
Происходит следующее: мы попросили программу сказать, что же такое x в этой точке, и в ответ получили, что x — это непонятно что (*undefined*, не определено). Значит, текстовое поле x не найдено — неверно заданный его адрес. Сообщения появляется 12 раз в секунду, каждый раз, когда происходит событие enterFrame. Заметим, что точно такой же результат даст строка
trace (rocket);
Значит, обращение к клипу тоже неверно.
Дело в адресах. В обработчиках событий клипа текущим монтажным столом считается сам клип. Поэтому для того, чтобы обратиться к текстовому полю x, что принадлежит не клипу, а главному монтажному столу, его адрес надо записать в виде _root.x (абсолютный адрес) или _parent.x (относительный адрес).
В правой части оператора надо обратиться к свойству _x текущего объекта, т.е. написать this._x (относительный адрес) или _root.rocket._x (абсолютный адрес). Но можно т же именно написать проще, ведь _x обозначает то же самое, что this._x.
Важно: Чтобы не ошибиться при обращении к объекту, можно нажать на кнопку  и выбрать со списка существующий объект. Программа автоматически вставит его адрес, относительный (при выборе варианта **Relative**) или абсолютный (вариант **Absolute**).
Практика : Выделите ракету на сцене и откройте панель **Actions** (клавиши **F9**). Исправьте код обработчика события enterFrame:

```
onClipEvent (enterFrame) {
   _root.x.text = _x;
}
```

Попробуйте вставить ссылку на текстовое поле с помощью кнопки / .
Практика: Прибавьте аналогичные пары (статический текст и динамическое текстовое поле) для других свойств клипа и дополните обработчик события командами, которые заполняют эти поля. Просмотрите результат.
Важно: Заметьте, что координата _x и другие свойства клипа — целые числа. При записи в текстовые поля они будут автоматически преобразованы в символьный вид.
Значение свойств клипа можно не только читать, но и изменять. После загрузки клипа на упоминание мы установим его начальные координаты в обработчике события load.
Практика: Прибавьте в код клипа *Ракета* обработчик события load (клип загружен на упоминание):

```
onClipEvent (load) {
  _x = 100;
  _y = 100;
}
```

11. «Мышиные события»
12. Клавиатура
Задача для работы: Сделать так, чтобы при прохождении курсора мыши над ракетой, ракета становилась немного больше прозрачной (на 5 процентов). Сделать чтобы ракета

оборачивалась против часовой стрелке, если мышь двигается в левой половине поля, и по часовой стрелке, если в правой. Ракету можно перетаскивать мышкой и перемещать клавишами-стрельцами на клавиатуре.

Пример:

Теперь займемся событиями от мыши. Кнопки и клипы могут обрабатывать немного «мышиных» событий:

- press — над объектом нажатая кнопка мыши;
- release — над объектом отпущенная кнопка мыши;
- releaseOutside — кнопка мыши отпущена вне объекта;
- rollOver — указатель мыши приведен на объект (без нажатия);
- rollOut — указатель мыши пошел из объекта (без нажатия);
- dragOut — указатель мыши пошел из объекта при нажатой кнопке;
- dragOver — при нажатой кнопке мышь идет из объекта, а потом приводится на него снова (аналогия — «чистим ботинки»).

Мы сделаем так, чтобы при событии rollOut ракета становилась немного больше прозрачной (на 5 процентов). Для этого надо уменьшить значение _alpha.

Практика : Прибавьте обработчик события мыши для клипа *Ракета*:

```
on (rollOut){
  _alpha -= 5;
}
```

Заметьте, что параметр _alpha по смыслу не должен быть меньше нуля. Будет неплохо, если вы сможете обеспечить это условие, прибавив в обработчик условный оператор if.

Перетаскивание

Перетаскивание мышкой тоже делается очень просто: при нажатии кнопки на объекте (событие press) надо «захватить» объект и начать перетаскивание:

```
startDrag ( this );
```

В дужках указывается адрес объекта (this означает «текущий объект», т.е. клип).

При отпускании кнопки (событие release) надо закончить перетаскивание:

```
stopDrag();
```

Практика: Прибавьте обработчики событий мыши для клипа *Ракета*:

```
on ( press ){
  startDrag(this);
}
on ( release ){
  stopDrag();
}
```

В дужках после слова on можно пересчитывать через кому несколько событий, на которые надо реагировать одинаково. Для этого часто удобно использовать средство **Script Assist** (помощник скриптов).

Практика: Выделите строку кода on(release) и щелкните по словам **Script Assist** в правому верхнему кованые панели **Actions**. Отметьте флажок **releaseOutside** и закройте окно повторным щелчком по словам **Script Assist.**

Теперь при отпускании мыши за пределами клипа перетаскивания также заканчивается.

«Мышиные события» клипа

Клип, как уже говорилось, также может реагировать на событии mouseDown, mouseUp и mouseMove. Важно, что эти обработчики вызываются не только тогда, когда мышь над клипом, а при любом изменении состояния мыши. При этом координаты курсора сохраняются в свойствах _xmouse и _ymouse главного монтажного стола _root.

Например, мы сделаем, чтобы ракета оборачивалась против часовой стрелке, если мышь двигается в левой половине поля, и по часовой стрелке, если в правой. Для этого надо написать обработчик события mouseMove, в котором менять свойство _rotation клипа *Ракета*. Чтобы определить направление обращения, будем сравнивать x-координату клипа с половиной ширины окна, т.е. с _root._width/2.

Практика: Прибавьте еще один обработчик к коду клипа:

```
onClipEvent (mouseMove) {
  if (_root._xmouse < _root._width/2)
      _rotation -= 0.5;
  else _rotation += 0.5;
}
```

При просмотре оказывается неточность: во время перетаскивания ракета продолжает оборачиваться, ведь событие mouseMove происходит и вызывается обработчик. Чтобы прекратить обращение во время перетаскивания, надо где-то запоминать, что мы тянем ракету. Легче всего сделать это с помощью специальной *логической* сменной dragging, что может принимать значение true («так», истина) и false («ни», неправда). В начале перетаскивания мы запишем в эту сменную true, а когда перетаскивание закончено, запишем у нее false.

Практика: Измените обработчики событий:

```
on ( press ){
  startDrag(this);
  dragging = true;
}
on ( release, releaseOutside ){
  stopDrag();
  dragging = false;
}
```

Теперь можно использовать значение этой сменной в обработчике mouseMove — оборачивать ракету только тогда, когда dragging=false.

Практика: Измените обработчик события:

```
onClipEvent (mouseMove) {
  if ( ! dragging ) {
    if (_root._xmouse < _root._width/2)
        _rotation -= 0.5;
    else _rotation += 0.5;
  }
}
```

Восклицательный знак означает операцию НЕ, так что условный оператор if(!dragging) имеет смысл «если не тянем ракету». Только при этом условию изменяется свойство _rotation.

Отличие кнопки от клипа

Как мы видели, кнопки и клипы могут обрабатывать события мыши с помощью обработчиков on (...). Однако код кнопки не может содержать обработчиков onClipEvent (...), которые предназначены только для клипов.

Важно: Главное отличие кнопки от клипа заключается в том, что кнопка (символ типа **Button**) — часть монтажного стола, на котором она находится, а клип (**Movie Clip**) имеет свой собственный монтажный стол.

Это означает, что у кнопок нет методов play() и stop().

Когда мы используем команду stop(); или this.stop(); в обработчике события **кнопки**, мы останавливаем проигрывание на ее монтажном столе (в этом случае — на _root).

Если же эта команда употребится внутри обработчика **клипа**, она останавливает проигрывание этого клипа (внутреннюю анимацию).
Если команда
_rotation += 45;
стоит в обработчике **кнопки**, на 45 градусов возвратится ее монтажный стол, т.е. _root. Та же самая команда в обработчике клипа возвратит на 45 градусов этот клип.
А если нужно возвратить кнопку? Это тоже можно сделать, но иначе, используя ее относительный или абсолютный адрес. На панели **Properties** введем имя кнопки. Пусть, например, кнопка с именем btn находится на главном монтажном столе _root. Тогда ее адрес может быть записан как _root.btn или (при обращении с _root) просто btn.
Возвратить кнопку в ее же обработчики можно с помощью команды
btn._rotation += 45;
или (из любого места программы!)
_root.btn._rotation += 45;

Клавиатура
Проще всего «ловить» нажатие клавиш с помощью события keyPress. В заголовке обработчика после слова keypress в кавычках относится нужен символ, например:
on (keyPress "A") { ... }
Для специальных клавиш указывают название в угловых дужках, например, "<Left>" (уліво), "<Right>" (вправо), "<Up>" (вверх), "<Down>" (вниз).

Практика: В код клипа прибавьте обработчики событий клавиатуры. Для клавиши «уліво»:
```
 on (keyPress "<Left>"){
   _x -= 2;
 }
```
и аналогично для других клавиш-стрелок.
К сожалению, в одном обработчике можно задать только реакцию на одну клавишу. Немного большую волю дает использование событий клипа keyDown и keyUp, но их
Вопросы для самостоятельной работы.

1. Как называется программа на *ActionScript?*
2. С какими элементами фильма может быть связан программный код?
3. Какие три простых типа переменных вы знаете?
4. Назовите **приоритет** (старшинство) арифметических операций в *ActionScript*
5. Объясните *Up, Over, Down* состояния кнопки.

Термины: переменные, объектно-ориентированное программирование, массив, условные операторы, циклы, функции

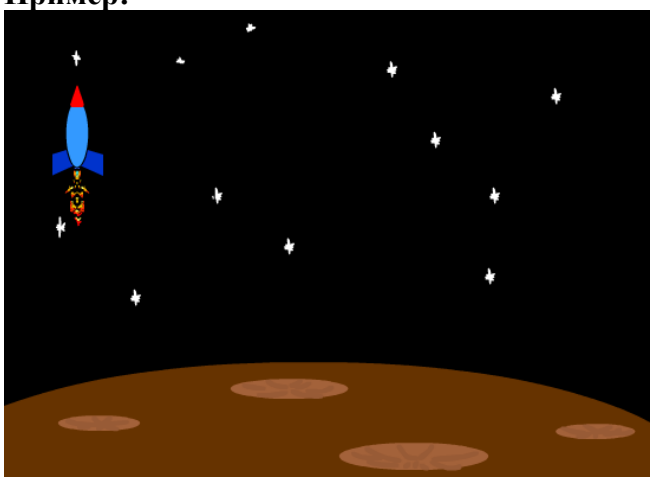
Литература:[1 С.222-267 ; 2 С.97-236]

Тема 8. Объекты среды *Flash*

1. *Key - клавиатура*

Задача для работы: Создать клип «Ракета». Рисуем звездное небо и поверхность планеты. Создаем объект «Ракета», которым можно управлять клавишами «стрелками» вверх, вниз, вправо, влево, а также перемещать мышью. Ракета должна самостоятельно опускаться на поверхность планеты при отсутствии нажатия клавиш вверх и вниз. При нажатии клавиши **Ctrl** скорость движения увеличивается. Также нужно, чтобы Ракета не могла полететь вниз за границы экрана.

Пример:



Практика: Создаем новый пласта Фон. Рисуем звездное небо и поверхность планеты. Создаем графический объект «Ракета1».

Пример:

Создаем объект Movie Clip «Ракета» и вставим наш объект «Ракета1».
Создаем новый пласта Ракета и вставляем у него объект Movie Clip «Ракета.

Практика: Прибавьте код, связанный с ракетой:

```
onClipEvent (keyDown) {
  trace(Key.getCode());
  trace(Key.isDown(Key.CONTROL));
  trace(Key.isDown(Key.SHIFT));
  trace(Key.isDown(Key.ALT));
}
```

Проверьте работу клипа, нажимая на разные клавиши.
Теперь посмотрим, как можно управлять ракетой из клавиатуры. Событие keyDown происходит только при нажатии клавиши, а нам нужно, чтобы ракета двигалась постоянно, если мы нажали клавишу и не отпускаем ее. Для этого мы используем событие enterFrame, где с помощью метода Key.isDown будем проверять, какие клавиши нажатые, и менять соответствующим образом координаты _x и _y клипа.
Скорость движения запишем в сменную v, ее начальное значение зададим в обработчике load (за время одного кадра ракета смещается на 3 пикселя в заданном направлении).

Практика: Прибавьте новые обработчики событий для ракеты:

```
onClipEvent (load) {
  v = 3;
}
onClipEvent (enterFrame) {
  if (Key.isDown(Key.RIGHT)) {
    _x += v;
```

```
  } else if (Key.isDown(Key.LEFT)) {
    _x -= v;
  }
 }
```
Аналогично запишите обработчики для клавиш-стрелок «вверх» и «вниз» (коды Key.UP и Key.DOWN).
Заметьте, что ракета может двигаться одновременно в двух направлениях, например, вверх и уліво.
Теперь сделаем так, чтобы при нажатии клавиши **Home** ракета возвращалась в исходное состояние (где она была в начале). Для этого в обработчике события load надо запомнить ее координаты в сменных x0 и y0, а при нажатии клавиши **Home** присваивать эти значения координатам клипа _x и _y.
Практика: Прибавьте строки в обработчик load:
```
  x0 = _x;
  y0 = _y;
```
Измените обработчик события keyDown на такой:
```
 onClipEvent (keyDown) {
  if (Key.isDown(Key.HOME)) {
    _x = x0;
    _y = y0;
  }
 }
```
Запустив ролик, вы можете с удивлением проявить, что это не работает. Если вставить строку
```
trace("Ку-ку");
```
в обработчик keyDown, мы проявим, что при нажатии клавиши **Home** этот обработчик не вызывается.
Дело в том, что мы запускаем клип из среды разработки, которая перехватывает некоторые клавиши, например, **Enter** и **Home**. При просмотре этого ролика в отдельном окне *Flash-проигрывателя* или в браузере все будет нормально.
Однако есть простой способ избавиться от этого неудобства.
Практика: Запустите ролик и отметьте флажок **Disable Keyboard Shortcuts** в меню **Control** в окне проигрывателя. Проверьте, реагирует ли теперь клип на нажатие клавиши **Home**.

Теперь сделаем так, чтобы при нажатии клавиши **Ctrl** скорость движения ракеты увеличивалась в 3 раза. Начальную скорость мы обозначим через v0.

Практика: Прибавьте в начало обработчика события enterFrame код
```
 if (Key.isDown(Key.CONTROL)) {
   v = v0*3;
 } else {
   v = v0;
 }
```
и измените первую строку в обработчике load на
```
 v0 = 3;
```
Практика: Самостоятельно сделайте так, чтобы ракета опускалась вниз, если не нажатая ни стрелка «вверх», ни стрелка «вниз». Также нужно, чтобы она не могла полететь вниз за границы экрана.

Дальше мы сделаем так, чтобы ракету можно было перетаскивать по экрану мышью. В теме 7.11 мы рассматривали код для перемещения объекта мышь. Используйте

его.

Практика: Подсказка

```
//перемещение объекта мышью
on (press) {
	startDrag(this);
}
on (release, releaseOutside) {
	stopDrag();
}
```

2. **Работа с текстом (*String, Selection*)**

Задача для работы: Построить небольшой редактора текста, который умеет делать выделенные буквы заглавные или строчными, считать количество символов и искать в тексте заданное слово.

Пример:

Объект *String* — строка

Символьные строки в программе состоят в двойные кавычки и представляют собой объекты класса **String** (строка). У строки есть одно свойство

- length — длина строки,

и несколько методов, из которых чаще всего применяются

- charAt(n) — символ в позиции n (номера символов начинаются **из нуля**);
- charCodeAt(n) — код символа в позиции n (целое число);
- substring(from, to) — подстрока, что начинается из позиции from и заканчивается **перед** позицией to (т.е., символ с номером to у нее не входит); если второй аргумент не указан, выделяется фрагмент до конца строки;
- indexOf(sub) — ищет подстроку sub, возвращает номер ее первого символа; если такой подстроки нет, возвращает -1;
- toLowerCase() — превратить в строке (маленькие) буквы;
- toUpperCase() — превратить в заглавные (большие) буквы.

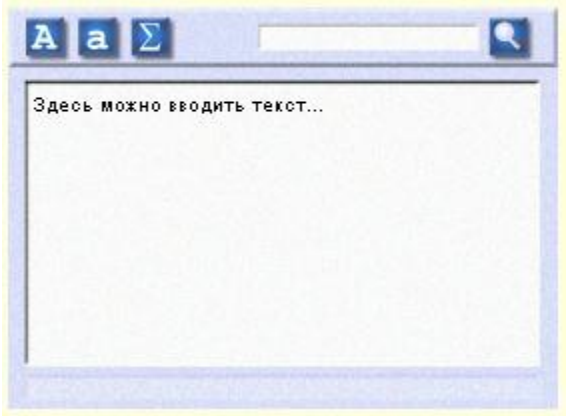
Для того, чтобы соединять строки, употребится знак +, например, так:

```
s = "123";
t = s + 45 + "6789";
trace ( t ); // получим "123456789"
```

Во второй строке число **45** было автоматически преобразовано к символьной форме.

В следующем проекте мы построим небольшой редактора текста, который умеет делать выделенные буквы заголовн или строчными, считать количество символов и искать в тексте заданное слово.

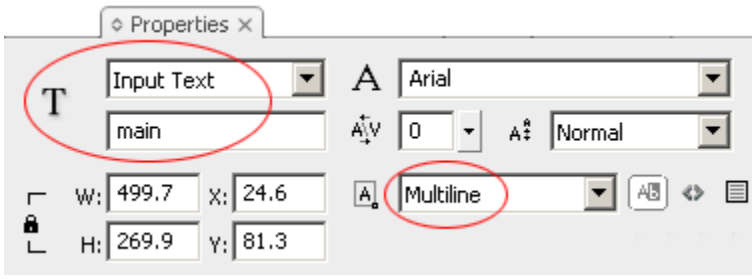
Пример:



Практика: Создаем новый проект, рисуем фон для текстового редактора. Создаем четыре

кнопки / , / , / , / . Разместить кнопки как в примере.

Практика: Создайте новый пласта *Текст* и разместите большое текстовое поле в середине сцены. Установите тип надписи **Input Text** (поле введения), имя **main** и параметр **Multiline** (многострочный текст), как показано на рисунке ниже.



Содержимое текстового поля — это тоже строку, т.е., объект класса *String*. Она имеет имя text, это значит, что адрес текстовой строки поля main запишется как main.text.
Сделаем так, чтобы при нажатии кнопок / и / все буквы текста становились заглавными (или соответственно строчными).
Практика : Прибавьте к кнопке код

```
 on ( release ){
   main.text = main.text.toUpperCase();
 }
```

а к кнопке / - код

```
 on ( release ){
   main.text = main.text.toLowerCase();
 }
```

Проверьте работу кнопок.

В нижней части поля сделаем информационную строку (динамическое текстовое поле) и при нажатии на кнопку будем выводить там количество символом в тексте, т.е. длину строки main.text.
Практика: Прибавьте в нижнюю часть сцены однорядковый динамический текст (**Dynamic Text**, **Single line**) с именем **info**. Для кнопки / введите обработчик события

```
 on ( release ){
   info.text = "Символов: " + main.text.length;
 }
```

Объект *Selection*
Объект **Selection** — это выделенная часть текста в одном из полей введения. Он связан с тем элементом, который в этот момент имеет **фокус**, т.е. принимает команды от клавиатуры.
Методы объекта **Selection**:

- setFocus("адрес") — передать фокус введения текстовому полю, адрес которого указан в кавычках;
- getBeginIndex() — возвращает номер первого выделенного символа;
- getEndIndex() — возвращает номер символа, на котором выделения заканчивается (этот символ уже не входит у выделение);
- setSelection(start,end) — выделить область от символа с номером start к символу с номером end.

Прибавим возможность поиска слова в тексте.
Практика : Прибавьте в верхнюю часть сцены (по левую сторону от кнопки) однорядковое поле введения (**Input Text**, **Single line**) с именем **find**.

Поиск начинается по щелчку на кнопке / .

Практика : Для кнопки задайте ведите обработчик события

```
on (release) {
  if ( find.text == "" ) return;
  n = main.text.indexOf ( find.text );
  if ( n < 0)
    info.text = "Ничего не найдено.";
  else {
    info.text = "Нашли в позиции " + n;
    Selection.setFocus ( "main" );
    Selection.setSelection ( n, n + find.text.length );
  }
}
```

Сначала проверяем, чи введено что-то в поле для поиска (если оно порожно, ничего делать не надо, и происходит выход из обработчика).
Потом, с помощью метода indexOf ищем образец в тексте. Если такого слова нет, выводит сообщение об этом в информационную строку. Если есть, то устанавливаем фокус на поле main и выделяем найденное слово.
Используя методы объекта **Selection**, можно улучшить работу кнопок и : если что-то выделено, они будут менять только выделенную часть текста.
Сначала мы определим начало и конец выделенной части и запишем номера этих символов в сменные nStart и nEnd. Если они равные, то ничего не выделено и превратится весь текст. Если эти значения не равные, красная строка строится так: содержимое main.text «разрезается» на 3 части с помощью метода substring, к второй части применяется метод toUpperCase.
Важно: Эта часть программы будет работать только для *Flash Player* версии 9 и выше, поскольку в ранних версиях во время щелчка по кнопке текстовое поле теряет фокус.

Практика : Измените обработчик кнопки на такой:

```
on (release) {
  nStart = Selection.getBeginIndex();
  nEnd = Selection.getEndIndex();
  if (nStart == nEnd) {
    main.text = main.text.toUpperCase();
  } else {
    s = main.text;
    s = s.substring ( 0, nStart ) +
        s.substring ( nStart, nEnd ).toUpperCase() +
        s.substring ( nEnd );
    main.text = s;
  }
}
```

Аналогично измените код кнопки / .
Обратите внимание, которое при вызове substring(nEnd) второй аргумент не указан, т.е. выделяется фрагмент из символа nEnd до конца строки.
Заметим, что алгоритм работает даже тогда, когда поле main не имеет фокуса введения (и обе функции getBeginIndex и getEndIndex возвратят -1).

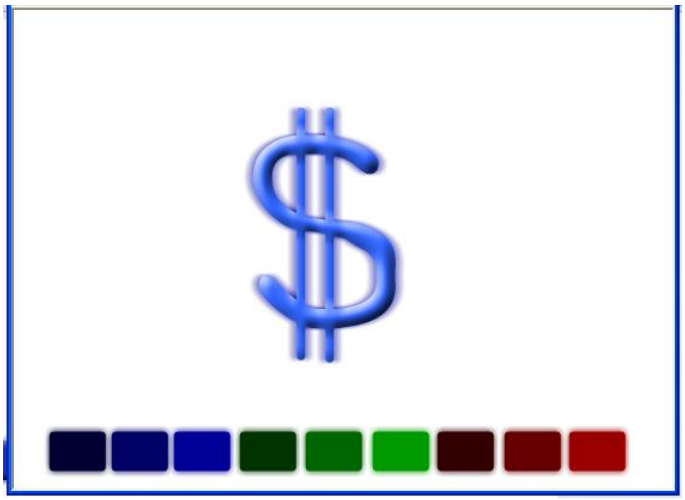
3. ***Color* — цвета**

Задача для работы: Нарисуйте знак доллара, превратите его в объект Movie Clip с именем **logo**. Примените к нему фильтры Bevel и Glow. Создайте кнопку и разместите внизу сцены девять таких кнопок. Написать программу для управления цветами объекта с помощью кнопок в нижней части сцены.

Объект *Color* разрешает изменять цвета клипов из программы. Для этого нужно сначала создать в памяти новый объект, связав его с клипом:
col = new Color (*адрес клипа*);
Таким образом, мы получаем доступ к цветам объекта. Потом можно использовать два метода:

- setRGB(*цвета*) — установить новые цвета;
- getRGB() — получить цвета, установленный последней командой setRGB.

Цвета клипов задается в формате RGB (R=*red*, красный; G=*green*, зеленый; B=*blue*, синий) как длинное целое число, в котором «упакованные» значения составляющих **R**, **G** и **B**, каждая из которых представляет собой целое число от 0 до 255.
Чтобы удобно было разбираться в цветах, принято записывать их в шестнадцатеричной системе исчисления, так что младшие две цифры — это **B**, что вытекают две — **G**, а две старших — **R**. Например, для цвета **0x660000** имеем **R=66_{16}=102**, **G=B=0**.
Пример:
Расположение объекта и кнопок :



Практика: Нарисуйте знак доллара, превратите его в объект Movie Clip с именем **logo**. Примените к нему фильтры Bevel и Glow. Создайте кнопку и разместите внизу сцены девять таких кнопок.

Практика : Выделите самую левую кнопку и перейдите на панель **Properties**. В списке **Color** выберите вариант **Tint** (цветовой оттенок), цвета с кодом **#000033** и смешение 100% (полная замена исходных цветов).

Практика : Выделите первую кнопку и введите код обработчика, который вызывается при отпускании кнопки мыши:

```
 on ( release ) {
   cLogo = new Color(_root.logo);
   cLogo.setRGB(0x000033);
 }
```

Аналогично задайте параметры и обработчик второй кнопки с цветами **0x000066**.
Проверьте работу фильма.
Конечно, можно применить те же шаги для других кнопок, но эту нудную работу можно значительно упростить, если использовать массивы и циклы.

Массивы

Массив — это набор элементов, которые имеют общее имя. Каждый элемент имеет собственный номер, начиная **из нуля**. Для обращения к элементу массива употребятся квадратные дужки.

Важно: Нумерация элементов массива **из нуля** непривычная для обычных людей, но широко употребится в программировании (языка *Си*, *JavaScript*, *Java*, *ActionScript*).

В отличие от многих языков программирования, в одном массиве могут быть разнотипные элементы: целые числа, вещественные числа, строки, логические сменные.

Массив из трех элементов можно объявить и заполнить так:

```
A = new Array(3);
A[0] = 12;
A[1] = 4.56;
A[2] = "Ку-ку!";
```

или так:

```
A = new Array (12, 4.56, "Ку-ку!");
```

или так:

```
A = [12, 4.56, "Ку-ку!"];
```

При работе с массивами, как и с другими объектами (кроме чисел, строк и логических сменных), есть один тонкий момент. Рассмотрим пример кода:

```
A = [1, 2, 3];
B = A;
B[1] = 99;
trace ( A );
```

Здесь создается массив A, потом он копируется в массив B. Потом изменяется B[1] и массив A выводит в окно **Output**. Будто бы бы массив A не должен измениться, однако мы увидим в окне **Output** строка:

1,99,3

Чему же изменился массив A? Дело в том, что оператор B=A НЕ создает новый массив, а просто копирует в B **адрес** массива A, т.е. A и B обращаются до одной области памяти. Поэтому, изменив B, мы изменили и A. Чтобы действительно создать копию массива, нужно заменить оператор B=A на код

```
B = new Array();
for(i=0; i<A.length; i++) B[i] = A[i];
```

Здесь употребится встроенное свойство length объекта Array — длина массива.

В нашем примере мы создадим массив, в котором будут сохраняться коды цветов всех кнопок.

Практика : Удалите раньше созданные обработчики. Создайте новый пласта *Программа* и в первый кадр введите код, который заполняет массив кодами цветов:

```
colors = [0x000033, 0x000066, 0x000099,
          0x003300, 0x006600, 0x009900,
          0x330000, 0x660000, 0x990000];
```

Оборотный внимание, что обработчики события release для всех кнопок почти одинаковые и отличаются только кодом цветов. Поэтому можно создать для них одну единую функцию, которые будут вызвать все кнопки.

Важно : Пласт *Программа* можно сразу же заблокировать, щелкнув по значку в столбце . Это разрешит прибавлять код *ActionScript*, но не даст случайно прибавить что-то еще.

Практика : Прибавьте функцию, которая изменяет цвета логотипа соответственно цветам кнопки, на которой щелкнули мышкой:

```
function changeColor() {
  c = new Color(this);
```

```
  col = c.getRGB();
  cLogo = new Color(_root.logo);
  cLogo.setRGB(col);
 }
```

В этой функции мы сначала определяем цвета кнопки, получив доступ к нему через сменную с и используя метод getRGB. Потом эти цвета переносится на логотип.
Созданная функция пока не работает, поскольку мы не определили, в каких случаях она вызывается. Для того, чтобы она стала обработчиком события release некоторой кнопки, довольно написать
Кнопка.onRelease = changeColor;
Дальше мы с помощью цикла, в котором сменная i (номер кнопки) меняется от 0 до 8, настроим все кнопки сразу: перекрасим их в нужны цвета и прибавим обработчик события.
Чтобы в цикле обращаться к кнопке по ее номеру, можно использовать специальную форму _root[*имя*]. Таким образом, _root["color0"] это то же самое, что и _root.color0, но преимущество первого метода в том, что имя кнопки можно построить динамично, во время выполнения, например, так:
_root["color"+chr(48+i)]
Функция chr превратит код символа в символ. Учитывая, что цифры **0-9** имеют коды **48-57**, при i=2 получаем _root["color2"].
Чтобы эта идея сработала, кнопкам нужно даты соответствующие имена **color0**, **color1** и т.д.
Практика : Дайте имена кнопкам **color0**, **color1** и т.д., начиная из левой. На панели **Properties**, выделив какую-нибудь кнопку.

Практика : Прибавьте к коду кадра 1 цикл:

```
 for (i=0; i<9; i++){
   obj = _root["color"+chr(48+i)];
   c = new Color(obj);
   c.setRGB(colors[i]);
   obj.onRelease = changeColor;
 }
```

Проверьте работу фильма и сохраните его.

Здесь сменная obj — это адреса кнопки с номером i.

4. ***Sound* — звук**

Объекты класса *Sound* служат для управления звуком. С их помощью можно

- изменять **громкость** звука;
- изменять распределение звука между левым и правым каналами, когда источник звука перемещается по экрану;
- создавать звуки динамично, т.е. из программы без использования временной шкалы.

Сначала научимся проигрывать звук из библиотеки и останавливать его в нужен момент.
Пример.
Практика: Найдите звук в библиотеке и нажмите на нем правую кнопку мыши. В контекстном меню выберите команду **Linkage...** (связывание), в окне, которое появилось, отметьте **флажки Export for** ActionScript **и Export in frame** frame, введите имя звука например - hey в **поле** Identifier.

Флажок **Export for ActionScript** делает звук доступным из программы, мы будем обращаться к нему по имени **hey**. Флажок **Export in first frame** загружает звук вместе с первым кадром, таким образом, звук будет доступен из самого начала фильма.
Теперь надо

- создать новый объект типа Sound с помощью конструктора new Sound();
- загрузить у него звук с помощью метода attachSound;
- запустить звук на проигрывание (метод start)
- остановит звук с помощью метода stop.

Мы свяжем этот код с клипом, поскольку он будет получать события mouseDown и keyDown вне зависимости от того, где находится мышка. Объект Sound строится сразу после загрузки в обработчике load, звук запускается в обработчике mouseDown и останавливается в случае события keyDown.
Практика: Выделите клип и прибавьте в окно **Actions** следующий код:

```
onClipEvent (load) {
  snd = new Sound();
}
onClipEvent (mouseDown) {
  snd.attachSound ("hey");
  snd.start(0, 1);
}
onClipEvent (keyDown) {
  snd.stop();
}
```

При вызове метода start первое число обозначает время от начала звукового файла в секундах (0 — из самого начала), а второй — количество повторов.
Важно: Если проигрывать звук не с начала, после завершения звуковой дорожки будет проиграна начальная часть.
Важно: С помощью вызова stopAllSounds(); можно прекратить проигрывание всех звуков сразу.
Чтобы «добраться» к звуку внутри клипа, нужно указать адресу клипа при создании объекта Sound. Например, в коде самого клипа можно использовать this:
bounce = new Sound(this);
Для изменения громкости употребится метод setVolume, в дужках нужно указать громкость звука в процентах от исходной.
Звуковой баланс регулируется с помощью метода setPan (*set panoram* — установить баланс), в дужках нужно задать величину от -100 (левая граница зоны, только левый канал) до 100 (правая граница, только правый канал).

5. **Дата и время**

Задача для работы : Построить ролик, который показывает сегодняшнюю дату, день недели и время, а также может служить будильником.

Объект *Date*
В этом разделе мы выучим средства для работы с датами и временами:

- объект Date (дата);
- функцию getTimer() (получить время таймера).

Создать новый объект класса Date можно так:
d = new Date(2013, 4, 09);
Первое число — год, второе — номер месяца, третье — число. Эта дата — 09 мая 2013 года. Здесь нет ошибки, номера месяцев начинаются из нуля, тому місяць 4 в программе *Flash* — это май.

Если параметры не употребятся:
today = new Date();
создается объект с сегодняшней датой.
С помощью методов объекта Date можно определить год, месяц, число и день недели:

- today.getFullYear() — год;
- today.getMonth() — месяц;
- today.getDate() — число;
- today.getDay() — день недели;
- today.getHours() — часы;
- today.getMinutes() — минуты;
- today.getSeconds() — секунды.

Дни недели, так же, как и месяцы, нумеруются из нуля, причем неделя начинается из воскресенья (день 0), понедельник имеет номер 1 и т.д.
Пример :

Практика : Нарисуйте фон и стилизацию отрывного календаря.

Практика : Прибавьте динамические текстовые поля (**Dynamic Text**) для вывода дня недели, числа, месяца года так, как на образце. Дайте им имена day, date, month и year, установите выравнивание по центру, выберите цвета и размер шрифта.

Практика: Прибавьте пласта *Программа* и в первом кадре введите код

```
stop();
today = new Date();
weekDays = ["воскресенье", "понедельник", "вторник",
          "среда", "четверг", "пятница", "суббота"];
months = ["января", "злого", "марта", "апреля",
         "мая", "июня", "июля", "августа", "сентября",
         "октября", "ноября", "декабря"];
day.text = weekDays[today.getDay()];
```

```
 date.text = today.getDate();
 month.text = months[today.getMonth()];
 year.text = today.getFullYear();
```
Проверьте работу фильма.

В первой строке мы остановили проигрывание, поскольку все изменения выполняются из программы. Потом в массивы weekDays и months записанные названия дней недели и месяцев (учитывая, что неделя у американцев начинается из воскресенья).
Потом в текстовые поля записываются данные сегодняшнего дня, причем некоторые из них выбираются из массивов.
С выводом времени дело обстоит немного сложнее, его надо обновлять раз в секунду. Сначала прибавим текстовое поле и напишем функцию, которая должна выводить у него время.
Практика : Прибавьте текстовое поле под листом календаря и дайте ему кодовое имя time. В код кадра 1 прибавьте функцию

```
 function showTime() {
   function str2 ( n ) {
     var s = String(n);
     if ( n < 10) s = "0" + s;
     return s;
   }
   today = new Date();
   h = today.getHours();
   m = today.getMinutes();
   s = today.getSeconds();
   time.text = str2(h) + ":" + str2(m) + ":" + str2(s);
 }
```

Эта функция содержит внутреннюю функцию str2, что принимает целое число n и переводит его в строку с помощью функции String. Потом, если число было меньше 10, к строке впереди прибавляется нуль. Это сделано потом, чтобы, скажем, время 5 часов 3 минуты и 2 секунды изображались как 05:03:02, а не 5:3:2.
В основной части функции создается новый объект Date, что содержит текущую дату и время. С помощью методов getHours, getMinutes и getSeconds из него вытягивают часы, минуты и секунды. Дальше они переводятся в символьный вид с помощью функции str2 и объединенная строка записывается в свойство text поля time.
Все бы хорошо, но функцию showTime надо вызвать периодически с интервалом не менее 1 секунды. К счастью, в среде *Flash* есть такое средство.
Практика : Прибавьте к коду кадра 1 строка

```
 setInterval ( showTime, 200 );
```

и проверьте работу клипа.

Первый параметр функции setInterval — это функция, которую надо вызвать, а второй — интервал между вызовами в миллисекундах.

Таймер
Займемся будильником.

Практика : Прибавьте на поле надписи (**Static Text**): Будильник, *время, сек, повторить* и *раз*, а также два поля введения (**Input Text**) с именами alarmTime (время сигнала) и loops (повторы). Создайте кнопку ПУСК. Прибавьте в нужное место кнопку *Пуск.*

Практика : В кадр 1 слоя *Программа* прибавьте присвоение начальных значений:

```
alarmTime.text = "10";
loops.text = "1";
```

Для отслеживания времени будем использовать внутренний таймер. Вызов функции
t = getTimer();
записывает в сменную t количество миллисекунд, что прошла с начала проигрывания фильма. Для будильника нам нужна разность между текущим временем и временами начала отсчета, что мы запомним в специальной сменной startTime.
В другого сменной, alarmOn будем запоминать состояний будильника (включенный или исключенный). Теперь можно написать код для кнопки: при ее нажатии включается будильник и запоминается время начала отсчета.
Практика : Прибавьте код обработчика для кнопки:

```
on (release) {
  clock.alarmOn = true;
  clock.startTime = getTimer();
}
```

Другая работа выполняется клипом clock: обращение стрелки, тиканье раз в секунду и выдача сигнала будильника.
Практика: Создать объект Movie Clip с именем clock. Нарисовать белый круглый циферблат с указанием 0, 15, 30 и 45 секунд.
Практика: Создать объект Movie Clip с именем *hand*. Нарисовать стрелку секундомера.
Практика : Вставить стрелку в циферблат.

Практика: Разместить готовый секундомер на главной сцене.

Практика: Из темы 7 проект *ship* взять звук *click.wav*. Переименовать его в *tick*. Из темы 6 взять звук *kuranty.mp3*. Переименовать его в *alarm*.
Практика : В контекстном меню выберите команду **Linkage...** (связывание), в окне, которое появилось, отметьте **флажки Export for** ActionScript **и Export in frame** frame, введите имя звуков *соответственно* tick *и* alarm в **поле** Identifier.

Практика : Прибавьте к секундомер-клипу-секундомеру код обработчика enterFrame:

```
onClipEvent (enterFrame) {
  if ( !alarmOn ) return;
  t = Number(_root.alarmTime.text);
  diff = Math.round((getTimer()-startTime)/1000);
  if (hand._rotation != diff*6) {
    hand._rotation = diff*6;
    tick = new Sound();
    tick.attachSound("tick");
    tick.start(0,1);
  }
  if (diff == t) Alarm();
}
```

Если будильник не включили, ничего делать не надо и функция заканчивает работу.
В сменную t записывается числовое значение установленного времени, для перевода из символьной строки в число употребится функция Number.
Разность между текущим и стартовым временем переводится у секунды с миллисекунд (распределением на 1000) и округляется к ближайшему целому числу с помощью функции Math.round, что относится к объекту Math (математика).

Если вспомнить, что за 1 секунду стрельца должна возвратиться на 6(=360/60) градусов, то нужен угол поворота стрелки равняется diff*6. Если стрелку надо «довернуть», изменяем ее свойство _rotation и 1 раз проиграем звук tick из библиотеки.
Если разность diff совпала с установленным временем t, вызывается функция Alarm, что мы сейчас напишем. В ней надо отключить будильник, возвратить стрелку в исходное положение и проиграть звук alarm из библиотеки нужное число раз.
Практика : Введите код обработчика собятия load для клипа-секундомера:

```
onClipEvent (load) {
  function Alarm() {
    alarmOn = false;
    hand._rotation = 0;
    snd = new Sound();
    snd.attachSound("alarm");
    snd.start(0,_root.loops.text);
  }
}
```

Проверьте работу будильника.

Заметим, что эта функция расположена в обработчике load, т.е. она доступная с момента загрузки клипа на упоминание.

Периодический вызов функции
Теперь прибавим надпись «Настало время вставать!», что должна исчезать через 2 секунды после начала сигнала.

Практика : Прибавьте динамический текст «Настало время вставать!» на сцену с часами и дайте ему имя wakeup. Выберите шрифт и примените фильтры на ваш вкус.

Сначала эта надпись надо спрятать.

Практика : Прибавьте в код кадра 1 слоя *Программа* строка

```
wakeup._visible = false;
```

Когда звенит будильник, мы откроем эту надпись, изменив ее свойство _visible на true.
Вспомним, что мы уже использовали средство, которое разрешает вызвать функцию периодически с заданным интервалом. Например, обращение некоторого клипа qq можно организовать так:

```
function rot() {
  qq._rotation += 10;
}
setInterval ( rot, 1000 );
```

Через 1000 мс (или через 1 сек) будет вызваться функция rot, так что каждую секунду клип будет возвращаться на 10 градусов.
Зсылку на интервал можно запомнить в сменной:

```
i = setInterval ( rot, 1000 );
```

Тогда для прекращения этих вызовов нужно вызвать функцию clearInterval:

```
clearInterval ( i );
```

или просто удалить из памяти эту сменную:

```
delete i;
```

Теперь применим этот способ к нашему случаю.
Практика : Прибавьте в конец функции Alarm строки

```
_root.wakeup._visible = true;
```

```
i = setInterval ( removeText, 2000 );
```

После этой функции (внутри обработчика события load) прибавьте еще одну функцию:

```
function removeText() {
  _root.wakeup._visible = false;
  delete i;
}
```

Проверьте работу клипа и сохраните его.

Вопрос для самостоятельной работы.

1. Что означает свойство length объекта класса **String** ?
2. Что означают методы setRGB(*цвета*) и getRGB() ?
3. Что можно сделать со звуком с помощью объекта класса *Sound* ?
4. Как нумеруются дни недели и месяцы в *ActionScript* ?

Термины: переменные, объектно-ориентированное программирование, объект,массив, условные операторы, циклы, функции.

Литература:[1 С.280-462 ; 2 С.100-320]