Занятие 12
Математика и анимация

Математические функции

Поддержка математических вычислений в JavaScript обеспечивается объектом **Math**. Данный объект не требует создания переменных типа Math при помощи оператора **new** - его можно использовать непосредственно в любой точке сценария. Использование объекта Math сводится к обращению к его свойствам или вызову его методов.
Рассмотрим применение объекта Math на примере вычисления длины окружности по формуле: пи*r^2

```
var r=10; // радиус окружности
var len = Math.PI*Math.pow(r,2); // вычисление длины окружности
```

В приведенном фрагменте сценария используется свойство **PI** объекта Math для получения значения числа ? и метод **pow**() для возведения радиуса окружности во вторую степень.
Ниже в таблице перечислены наиболее популярные свойства и методы объекта Math:

Свойства и методы объекта Math

Свойство	Пояснение
Math.**E**	Возвращает значение константы **E** (основание натурального логарифма), равное приблизительно 2.718
Math.**LN2**	Возвращает значение натурального логарифма числа 2
Math.**LN10**	Возвращает значение натурального логарифма числа 10
Math.**LOG2E**	Возвращает значение логарифма числа **E** по основанию 2
Math.**LOG10E**	Возвращает значение логарифма числа **E** по основанию 10
Math.**PI**	Возвращает значение числа **PI**, приблизительно равное 3.14159
Math.**SQRT1_2**	Возвращает значение корня квадратного из 1/2 (приблизительно 0.707)
Math.**SQRT2**	Возвращает значение корня квадратного из

Свойства и методы объекта Math

Свойство	**Пояснение**
	2 (приблизительно 1.414)
number Math.**abs**(*number1*)	Возвращает абсолютное значение числа (модуль числа)
number Math.**acos**(*number1*)	Возвращает угол, косинус которого равен *number1*
number Math.**asin**(*number1*)	Возвращает угол, синус которого равен *number1*
number Math.**atan**(*number1*)	Возвращает угол, тангенс которого равен *number1*
number Math.**ceil**(*number1*)	Возвращает результат округления числа *number1* до ближайшего большего целого
number Math.**cos**(*number1*)	Возвращает косинус угла; *number1* - угол в радианах
number Math.**exp**(*number1*)	Возвращает результат возведения числа **E** в указанную степень
number Math.**floor**(*number1*)	Возвращает результат округления числа *number1* до ближайшего меньшего целого
number Math.**log**(*number1*)	Возвращает натуральный логарифм числа *number1*
number Math.**max**(*number1*[, *number2*[, ...]])	Возвращает максимальное значение из числа переданных аргументов (*number1*, *number2*, ...)
number Math.**min**(*number1*[, *number2*[, ...]])	Возвращает минимальное значение из числа переданных аргументов (*number1*, *number2*, ...)
number Math.**pow**(*number1*, *number2*)	Возвращает результат возведения числа *number1* в степень *number2*
number Math.**random**()>	Возвращает случайное число в диапазоне от 0 (включительно) до 1 (не включительно): 0<=random()<1
number Math.**round**(*number1*)	Возвращает результат округления числа *number1* по правилам округления: если дробная часть меньше 0.5, происходит округление в меньшую сторону, в противном случае - в большую сторону.
number	Возвращает синус угла; *number1* - угол в

Свойства и методы объекта Math

Свойство	**Пояснение**
Math.**sin**(*number1*)	радианах
number	Возвращает квадратный корень числа
Math.**sqrt**(*number1*)	*number1*
number	Возвращает тангенс угла; *number1* - угол в
Math.**tan**(*number1*)	радианах

Совершенно очевидно, что все эти методы и математические константы вы вряд ли будете использовать повсеместно в своих сценариях. Однако есть ряд задач, где использование некоторых математический функций является необходимостью. К примеру, многие задачи, в которых обрабатываются числовые данные, могут потребовать использования методов округления чисел или возведения их в различные степени.

Кроме того, при помощи JavaScript можно *анимировать* различные элементов страницы. Серьезные задачи по анимации объектов так же требуют математического описания законов движения.

Примеры применения математических методов

Рассмотрим несколько примеров применения математических методов в реальных расчетах.

Пример 1: округление с заданной точностью. Объект Math содержит три функции округления чисел: floor, ceil и round. Однако, все эти функции округляют число до целого, что делает их неприменимыми в финансовых расчетах. Попробуем создать собственную функцию, которая учитывала бы указанное число десятичных знаков при округлении.

Подойдем к задаче теоретически. Пусть число 3.141516 необходимо округлить до второго десятичного знака. Для этого требуется умножить число на 100 (получим 314.1516), затем, округлить его по правилам округления (получим 314) и результат поделить на 100. В итого получим 3.14. При округлении до 3х десятичных знаков необходимо умножать и делить число на 1000, и т.п. Обобщив накопленную информацию делаем вывод, что для округления до *n* десятичных знаков, число нужно умножать и делить на 10^n.

С учетом вышесказанного напишем функцию округления:

```
function round2(x, n) {
   var mult=Math.pow(10, n);
   return Math.round(x*mult)/mult;
}
```

Как видите, код функции весьма прост. Сначала вычисляется множитель, на который будет умножаться, а затем и делиться исходное число "x". Далее выполняются действия по округлению. Результат округления возвращается оператором **return** в программу. Ниже приводится несколько примеров применения данной функции:

```
var x=3.141516;
alert (round2(x,1)); // 3.1
alert (round2(x,2)); // 3.14
alert (round2(x,3)); // 3.142
```

Пример 2: генерация случайного значения в указанном диапазоне. Генерацией случайных чисел в JavaScript "занимается" метод random() объекта Math. Но, как известно, этот метод генерирует действительные случайные значения от 0 до 1 (не вкл.). Что же делать, если в сценарии требуется генерировать целые случайные числа в более широком диапазоне (например, от 0 до 100)?

Для этого сгенерированное методом random() число умножим на верхний предел диапазона и округлим в сторону ближайшего меньшего целого. Новая функция random2(*n*) будет генерировать целые числа от 0 до *n*-1:

```
function random2(n) {
   return Math.floor(Math.random()*n);
}
```

Таким образом, при необходимости получения случайных целых чисел в диапазоне, например от 50 до 100 необходимо воспользоваться следующим выражением:

```
var randomNumber = 50+random2(101);
```

Надеюсь, эти простые, но полезные примеры пролили немного света на использование математических методов в JavaScript.

Пример программы реализующей вывод случайных чисел от 1 до указанного пользователем, без повторений!

```
<script type="text/javascript">
function rand(n) {
   return Math.floor(Math.random()*n);
}


function numbers(nPred,vPred)
{
	var num, vPred, c=0,t=0;

	vPred=parseInt(prompt("Вводим верхний диапазон","10"));

	var a=new Array();

	for (var i=1; i<=vPred;i++)
		{
			num=1+rand(vPred);
			while(true)
				{	c=0;
					for(var j=1;j<=vPred;j++)
						{
							if(a[j]==num) c++;
						}

					if (c) num=1+rand(vPred);
					else
						{
							a[i]=num;
							t++;
							document.write(t+". "+a[i]+"<br/>");
							break;
						}
				}
		}

}

numbers();
</script>
```

Анимация объектов страницы

В данном разделе рассматривается математическое описание и алгоритмы перемещения объекта (по прямой, по окружности или эллипсу и т.п.). Для реализации этих алгоритмов нам понадобятся знания *полярной системы координат*.

Примечание: Далее приводятся математические выкладки. Если вы знакомы с полярной системой координат и преобразованием координат из полярной системы в прямоугольную декартову систему координат или считаете изучение данного материала нецелесообразным (или сложным), вы можете перейти к следующей теме данного раздела (см. ниже).

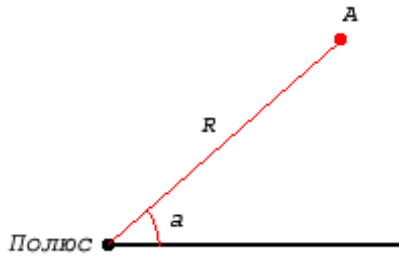
Полярная система координат

В полярной системе координат положение некоторой точки "A" на плоскости задается при помощи двух чисел *R* и *a*, где

- *R* - расстояние до точки от *полюса*,
- *a* - угол между отрезком, соединяющим полюс и точку "A", и вектором, задающим начальный угол (как правило - горизонталь).

Полюс - начало координат.

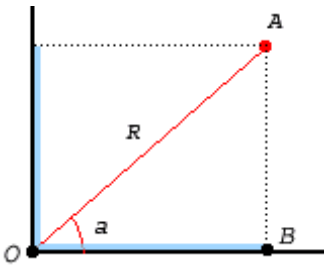
Рисунок ниже демонстрирует применение полярной системы координат:



Полярная система координат

Полярная система координат достаточно удобна для описания движения по прямой под заданным углом к горизонту. Для того, чтобы точка "A" "двигалась", необходимо просто с определенной периодичностью увеличивать ее расстояние до полюса - *R*. Однако, координаты всех элементов страницы задаются в прямоугольной декартовой системе координат. Следовательно, необходимо делать преобразование систем координат.

Рассмотрим следующий рисунок:



Переход в прямоугольную систему координат

Точки "A", "B" и "O" (полюс) образуют прямоугольный треугольник, катеты которого и есть искомые координаты объекта в декартовой системе. При известной гипотенузе (*R*) и углу (*a*) найти оба катета не составит труда. Займемся катетом "OB" (x-координатой объекта):

- Известно, что OB/R = cos(a)
- Из этого следует, что OB = R*cos(a).

Аналогично определяется и длина второго катета ("AB", y-координата):

- Известно, что AB/R = sin(a)
- Из этого следует, что AB = R*sin(a).

Итоговые формулы преобразования:

- **x = R*cos(a)**
- **y = R*sin(a)**

Движение по прямой под углом к горизонту

В данном подразделе урока мы с вами попробуем применить теорию на практике. В первую очередь, реализуем алгоритм перемещения объекта по прямой, направленной под произвольным углом к горизонту.

При реализации данного алгоритма надо помнить следующее:

- элемент, который предполагается перемещать, должен иметь абсолютное или относительное позиционирование,
- точка "0" располагается в верхнем левом углу страницы,
- ось абсцисс (x) направлена вправо,
- ось ординат (y) направлена вниз,
- положительное направление отсчета угла соответствует направлению вращения часовой стрелки (угол 0° - горизонталь),
- функции sin(a) и cos(a) требуют указания угла "a" в радианах (1° = ?/180 радиан).

Ниже приводится листинг примера, реализующего движение по прямой, направленной под углом 30° к горизонту:

Движение по прямой под углом к горизонту

```
<body>
  <span id="elemToMove" style="position: absolute; left: 0; top: 0;">Я
```

```
лечу!!!</span>
  <script type="text/javascript">
    var R = 0,            // расстояние до полюса
      a = 30*Math.PI/180, // угол (в радианах)
      dR = 5,             // приращение расстояния
      delay = 50;         // задержка (в мсек.)

    // функция движения элемента
    function moveElem() {
      // изменение координат элемента
      elemToMove.style.pixelLeft = R*Math.cos(a);
      elemToMove.style.pixelTop  = R*Math.sin(a);
      // увеличение расстояния до полюса
      R+=dR;
    }
    // периодический вызов функции движения
    setInterval ("moveElem()", delay);
  </script>
<body>
```

Движение по окружности

Алгоритм движения по окружности так же использует полярную систему координат, однако, при этом, требуется изменять не расстояние до полюса, а угол (a). Кроме того, для того, чтобы элемент вращался не вокруг точки с координатами (0,0), к координатам объекта необходимо добавлять координаты центра окружности (например - (100,100)).

Ниже приводится код сценария, демонстрирующий вращение объекта вокруг точки с координатами (100,100):

Движение по окружности

```
<body>
  <span id="elemToMove" style="position: absolute; left: 0; top: 0;">Я вращаюсь!!!</span>
  <script type="text/javascript">
    var R = 80,             // радиус окружности
      a = 0,              // угол (в радианах)
      da = 3*Math.PI/180,   // приращение угла
      delay = 30;         // задержка (в мсек.)

    // функция движения элемента
    function moveElem() {
      // изменение координат элемента
      elemToMove.style.pixelLeft = 100+R*Math.cos(a);
      elemToMove.style.pixelTop  = 100+R*Math.sin(a);
      // увеличение расстояния до полюса
      a+=da;
    }
    // периодический вызов функции движения
    setInterval ("moveElem()", delay);
  </script>
<body>
```