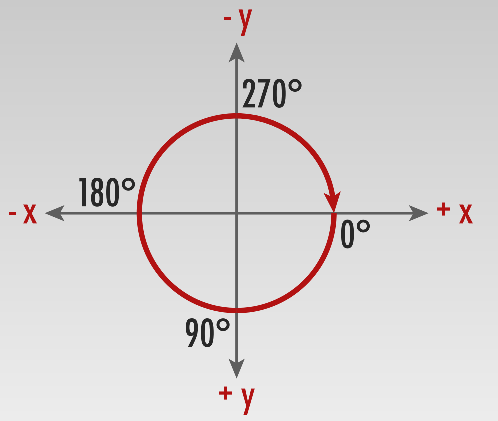
BASIC MATH CHEAT SHEET FOR ACTIONSCRIPT PROGRAMMERS v1.3

COORDINATES SYSTEM



PSEUDO-RANDOM NUMBERS

Math.random() → 0 ≤ n < 1

Generate a random number between min and max

```
number = min + Math.random() * (max - min);
```

Generate a random integer between min and max

```
i=Math.floor(Math.random()*(1+max-min)+min);
```

Generate a random boolean value (true/false)

```
bool = Math.random() > .5 ? true : false;
```

Randomly pick an element from an array

```
myArray[int(Math.random()*myArray.length)];
```

Randomly position a top-left-anchored sprite (s) so its fully visible on stage

```
s.x=Math.random()*stage.stageWidth-s.width;
s.y=Math.random()*stage.stageHeight-s.height;
```

DECIMAL, BINARY & HEX NUMBERS

Decimal 0 1 2 3 4 5 6 7 8 9 (10 digits)

29 → $2\times10^1 + 9\times10^0$

20 + 9

Binary 0 1 (2 digits)

11101 → $1\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

16 + 8 + 4 + 0 + 1

Hexadecimal 0 1 2 3 4 5 6 7 8 9 A B C D E F (16 digits)

1D → $1\times16^1 + D\times16^0$

16 + 13

Trace string representation of decimal number in bin or hex

```
trace( num.toString(2) );  // Binary
trace( num.toString(16) ); // Hex
```

MODULAR ARITHMETIC

Perform an action once every n loops

```
for (var i:uint=0; i<50; i++) {
   if (i%n == 0) trace(i);
}
```

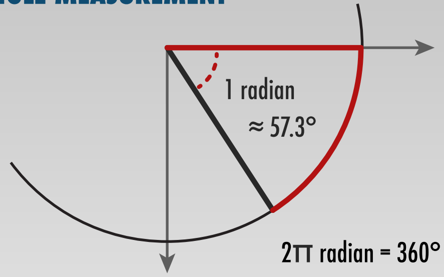
Round number to the previous multiple of n

```
number = x - (x % n);
```

Convert a large angle value to its less-than-360 equivalent

```
angle = angle % 360;
```

ANGLE MEASUREMENT



The `rotation` property of `DisplayObjects` expects degrees but functions, such as `Math.cos()`, expect or return radians. Here's how to convert between the two:

```
degrees = radians * 180 / Math.PI;
radians = degrees * Math.PI / 180;
```

COLOR BINARY ARITHMETIC

Extract red, green & blue from RGB hex color value (24bits)

```
red = rgbColor >> 16;
green = rgbColor >> 8 & 0xFF;
blue = rgbColor & 0xFF;
```

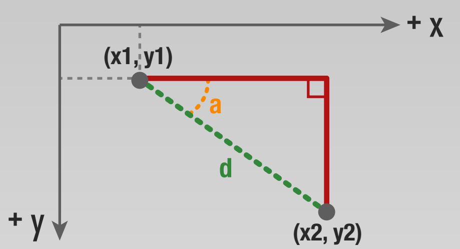
Extract alpha, red, green & blue from ARGB color (32bits)

```
alpha = argbColor >> 24;
red = argbColor >> 16 & 0xFF;
green = argbColor >> 8 & 0xFF;
blue = argbColor & 0xFF;
```

Create color from RGB (24bits) or ARGB (32bits) values

```
color = red << 16 | green << 8 | blue;
color = alpha << 24 | red << 16 | green << 8 | blue;
```

DISTANCES, ANGLES & DESTINATIONS



Given origin (x1, y1), distance (d) and angle (a, in radians), get destination (x2, y2) :

```
x2 = Math.cos(a) * d + x1;
y2 = Math.sin(a) * d + y1;
```

With destination (x2, y2), calculate distance or angle:

```
d = Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
a = Math.atan2(y2-y1, x2-x1);
```

To rotate an object so it follows the mouse:

```
dx = mouseX - sprite.x;
dy = mouseY - sprite.y;
obj.rotation = Math.atan2(dx, dy) *
               180 / Math.PI;
```

OPERATOR PRECEDENCE

Multiplicative × ÷ % BEFORE → Additive + −

Multiplication, division and modulo come before addition and substraction. Anything in parenthesis has priority.

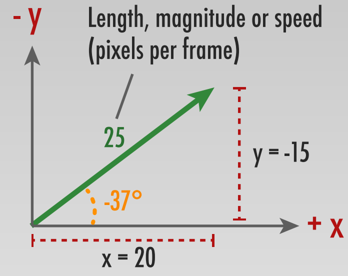
```
trace( 8 + 4 / 2 - 1 )      // 9
trace( (8 + 4) / 2 - 1 )    // 5
trace( 8 + 4 / (2 - 1) )    // 12
```

Prefix increment vs Postfix increment

```
var num:Number = 0;
trace(++num); // 1
trace(num);   // 1
```

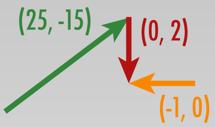
```
var num:Number = 0;
trace(num++); // 0
trace(num);   // 1
```

MOVEMENT VECTORS



Let's say we throw a ball at a -37° angle with a speed of 25 pixels per frame. This can be expressed in x/y coordinates as (20,-15) as you can see above. It means the ball moves 20 pixels right and 15 pixels up per frame. After 2 frames, the ball would be at position (40,-30).

Now, let's factor in gravity pushing down by 2 pixel each frame (0,2) and wind pushing left by 1 pixel per frame (-1,0). How would we calculate the position of the ball on each ENTER_FRAME ?



By repeatedly adding the x and y component of each relevant vector to the velocity and using the result to move the ball :

```
var velocity:Object = {x:20, y:-15};
var gravity:Object  = {x:0, y:2};
var wind:Object = {x:-1, y:0};

addEventListener(Event.ENTER_FRAME, move);

public function move(e:Event):void {
  balle.x += (velocity.x += (gravity.x + wind.x));
  balle.y += (velocity.y += (gravity.y + wind.y));
}
```