Unidad 4

Matemáticas: Funciones Aritméticas

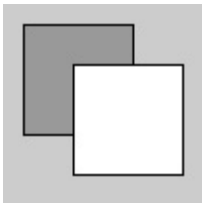
Elementos que se introducen en esta Unidad:

```
+ (suma), - (resta), * (multiplicación), / (división), % (módulo), () (paréntesis)
++ (incremento), – (decremento), += (asignar de suma), -= (asignar resta),
*= (asignar multiplicación), /= (asignar división),
- (negativo)
ceil(), floor(), round(), min(), max()
```

Tener amplios conocimientos en matemáticas puede ser importante a la hora de programar. Sin embargo, **no es necesario** ser un basto conocedor de la misma para disfrutar de la programación. Hay muchas formas y estilos para programar, y cada quien elige individualmente a que aspectos otorgarles un mayo grado de minuciosidad y a cuales descuidarlos un poco.
Es muy común que aquellos que tuvieron una cantidad elevada de matemáticas en la escuela (por ejemplo, en orientaciones en ciencias naturales o en alguna técnica) se sientan atraídos al ver su matemática convertida en movimiento. No obstante, no es necesario ser un verdadero conocedor. Processing es un lenguaje que se presta a la matemática desde un aspecto más tranquilo. Si bien pueden realizarse complejas operaciones, también pueden lograrse resultados similares con una matemática más intuitiva. En esta clase de manuales, se explican funciones aritméticas sencillas para lograr grandes resultados, puesto funciones de mayor complejidad caen fuera de los objetivos de este libro.

-Aritmética

En programación, las propiedades visuales de una imagen son asignadas por **números**. Esto quiere decir que podemos controlarlas de forma aritmética. El siguiente es un claro ejemplo de eso:



```
int grisVal = 153;
fill(grisVal);
rect(10, 10, 55, 55);         //Dibuja un rectángulo gris
grisVal = grisVal + 102;      //Asigna 255 a grisVal
fill(grisVal);
rect(35, 30, 55, 55);         //Dibuja un rectángulo blanco
```

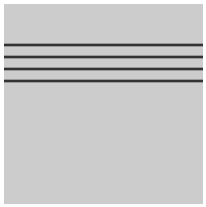
Utilizando los operadores de suma, resta, multiplicación y división, podemos controlar, por ejemplo, la posición de los elementos. El signo + se utiliza para la suma, el signo – para la resta, el signo * para la multiplicación y el signo / para la división.

```
int a = 30;
line(a, 0, a, height);
a = a + 40;             //Asigna 70 a la variable a
strokeWeight(4);
line(a, 0, a, height);
```

```
int a = 30;
int b = 40;
line(a, 0, a, height);
line(b, 0, b, height);
strokeWeight(4);
line(b - a, 0, b - a, height);     //Pueden usarse cálculos
                                   //entre variables como
                                   //valores
```

```
int a = 8;
int b = 10;
line(a, 0, a, height);
line(b, 0, b, height);
strokeWeight(4);
line(a * b, 0, a * b, height);
```

```
int a = 8;
int b = 10;
line(a, 0, a, height);
line(b, 0, b, height);
strokeWeight(4);
line(a / b, 0, a / b, height);
```



```
int y = 20;
line(0, y, width, y);
y = y + 6;              //Asigna 26 a la variable y
line(0, y, width, y);
y = y + 6;              //Asigna 32 a la variable y
line(0, y, width, y);
y = y + 6;              //Asigna 38 a la variable y
line(0, y, width, y);
```

Los signos `+`, `-`, `*`, `/` y `=` posiblemente sean muy familiares, pero el signo `%` es mucho más exótico. El operador `%` es el de módulo. Determina cuando un número es dividido por otro, o sea, da como resultado el *resto* del mismo. El número a la derecha es dividido por el que pongamos a la izquierda. Devolverá como resultado el resto de la operación.

Expresión	Resultado	Explicación
`9 % 3`	`0`	El 3 solo entra tres veces en el 9, no sobra nada.
`9 % 2`	`1`	El 2 solo entra cuatro veces en el 9, sobra 1.
`35 % 4`	`3`	El 4 entra ocho veces en el 35, sobra 3.

Sin muchos problemas, puede explicarse más claramente con un cuento. Cuatro personas están hambrientas y van a comer panes a un restaurante. Si hay solo 35 panes, entonces esas 4 personas solo podrán comer 8 panes cada uno, y sobrarán 3 panes. Eso que sobra es el resultado del módulo. Hay muchos usos para el módulo, como por ejemplo saber cuando un número es par o impar. Sin embargo, en Processing es muy común utilizarlo para mantener a los números en rango. Por ejemplo, si una variable sigue incrementándose (1, 2, 3, 4, 5, 6, 7, 8...), el módulo puede convertirla en una secuencia. Un incremento continuo puede convertirse en un ciclo entre 0 y 3 por aplicar `%` 4:

x	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
x%4	*0*	*1*	*2*	*3*	*0*	*1*	*2*	*3*	*0*	*1*	*2*

Muchos ejemplos a lo largo de las explicaciones usarán el módulo de esta forma.

Al trabajar con operaciones matemáticas y variables, es importante tener en claro que tipo de datos estamos manejando. La combinación de dos enteros (`int`) nos dará como resultado otro entero. La combinación de dos decimales (`float`) siempre nos dará como resultado un decimal. Pero la combinación de un entero (`int`) con un decimal (`float`) siempre nos dará un número decimal, a pesar de la operación diera un resultado exacto. Es importante ser cuidadoso al momento de combinar tipos de datos para evitar errores:

```
int i = 4;
float f = 3.0;
int a = i / f;          //ERROR – Imposible asignar un número decimal a una
                        //variable tipo entero
float b = i / f;        //Asigna 1.3333334 a la variable b del tipo flotante
```

Es también importante prestar atención a algunos conceptos matemáticos, como por ejemplo el concepto de límite, donde un número dividido por cero tiende a *infinito*. En esos casos el programa producirá un error:

```
int a = 0;
int b = 12 / a;         //ERROR – Excepción Matemática: producida por el cero.
```

-Prioridad en el Operador. Agrupación

La prioridad del ordenador determina que operaciones se realizan **antes que otras**. Esto es de gran importancia ya que puede ser determinante para nuestro resultado. Por ejemplo, la expresión `3 + 4 * 5`, nos dará como resultado `23`, puesto que tiene prioridad la multiplicación. Primero, `4 * 5` resultará en `20` y a continuación se le sumará `3`, dando por resultado `23`. La prioridad puede ser cambiada agregando paréntesis. Si tuviéramos la expresión `(3 + 4) * 5`, entonces la prioridad estaría en la suma, dando como resultado final 35.

```
x = 3 + 4 * 5;          //Asigna 23 a la variable
y = (3 + 4) * 5         //Asigna 35 a la variable
```

La siguiente tabla, a modo explicativo, muestra que elementos tiene prioridad sobre otros. Siendo, pues, que los que se encuentran mas arriba, tiene mas prioridad que los que están debajo:

Multiplicativos	`*  /  %`
Aditivos	`+  -`
Asignación	`=`

-Atajos

Cuando programamos, recurrimos mucho a estructuras cíclicas. Muchas veces es necesario utilizar atajos de escritura para reducir el tamaño del código y que sea mas legible. El operador de incremento es `++`, mientras que el de decremento es `--`. Ambos son usados como atajos de suma y resta.

```
int x = 1;
x++;              //Equivale a x = x + 1;
println(x);       //Imprime 2

int y = 1;
y--;              //Equivale a y = y - 1;
println(y);       //Imprime 0
```

Si deseáramos que se actualicé el valor antes de la expresión, solo cambiaremos de lugar el operador de incremento/decremento:

```
int x = 1;
println(++x);     //Imprime 2
println(x);       //Imprime 2
```

Al asignar el operador de suma con el operador de asignación (+=) obtendremos el operador de suma

asignada. En el caso de combinar el de resta con el de asignación (`-=`) obtendremos el operador de resta asignada:

```
int x = 1;
println(x);       //Imprime 1
x += 5;           //Equivale a x = x + 5;
println(x);       //Imprime 6

int y = 1;
println(y);       //Imprime 1
y -= 5;           //Equivale a y = y - 5;
println(y);       //Imprime -4
```

Al asignar el operador de multiplicación con el operador de asignación (`*=`) obtendremos el operador de multiplicación asignada. En el caso de combinar el de división con el de asignación (`/=`) obtendremos el operador de división asignada:

```
int x = 4;
println(x);       //Imprime 1
x *= 2;           //Equivale a x = x * 2;
println(x);       //Imprime 8

int x = 4;
println(x);       //Imprime 1
x /= 2;           //Equivale a x = x / 2;
println(x);       //Imprime 2
```

El operador de negación (`-`) es utilizado cuando deseamos cambiar el signo (positivo/negativo) de algún número:

```
int x = 5;
x = -x;           //Equivale a x = x * -1
println(x);       //Imprime -5
```

-Limitando Números

Las funciones `ceil()`, `floor()`, `round()`, `min()` y `max()` son utilizadas para perfeccionar las operaciones matemáticas. La función `ceil()`, redondea un número decimal convirtiéndolo en un entero. Lo convierte en el valor igual o mayor. Redondea hacia arriba.

```
int x = ceil(2.0);      //Asigna 2 a x
int y = ceil(2.1);      //Asigna 3 a y
int w = ceil(2.5);      //Asigna 3 a w
int z = ceil(2.9);      //Asigna 3 a z
```

La función `floor()`, redondea un número decimal convirtiéndolo en un entero. Lo convierte en el valor igual o menor. Redondea hacia abajo.

```
int x = floor(2.0);     //Asigna 2 a x
int y = floor(2.1);     //Asigna 2 a y
int w = floor(2.5);     //Asigna 2 a w
int z = floor(2.9);     //Asigna 2 a z
```

La función `round()`, redondea un número decimal convirtiéndolo en un entero. Lo convierte en un entero mayor o menor, dependiendo el decimal. Si el decimal es menor a .5, entonces redondea hacia abajo. Si el decimal es mayor o igual a .5, redondea hacia arriba.

```
int x = round(2.0);     //Asigna 2 a x
int y = round(2.1);     //Asigna 2 a y
int w = round(2.5);     //Asigna 3 a w
int z = round(2.9);     //Asigna 3 a z
```

Además, las funciones `ceil()`, `floor()` y `round()` funcionan con variables tipo decimales. A pesar de no ser muy útil, puede utilizarse.

```
float y = round(2.1);   //Asigna 2.0 a y
```

La función `min()` determina el número mas chico en una secuencia de números. La función `max()` determina el número más grande en una secuencia de números. Ideal para determinar el número mayo o menor de una secuencia. Ambas funciones pueden admitir dos o tres parámetros.

```
int u = min(5, 9);            //Asigna 5 a u
float t = min(12.6, 7.89)     //Asigna 7.89 a t
int v = min(4, 88, 65);       //Asigna 4 a v

int y = max(5, 9);            //Asigna 9 a y
float x = max(12.6, 7.89)     //Asigna 12.6 a x
int w = max(4, 88, 65);       //Asigna 88 a w
```