Programación con *Mathematica*

Prof. Enrique Vílchez Quesada

¿Qué es Mathematica?

- Es un software desarrollado por la compañía Wolfram Research, Inc
- Constituye una aplicación de cálculo simbólico con su propio lenguaje de programación
- Integra múltiples funciones para el tratamiento de diversas áreas, tales como: cálculo diferencial e integral, álgebra lineal, teoría de números y estadística

¿Por qué es importante?

- Mathematica ofrece la posibilidad de simular la resolución de problemas relacionados con la toma de decisiones
- Permite la creación de experimentos
- Es una herramienta de alto nivel que integra el cálculo simbólico y la programación

Simplify and FullSimplify

Comandos para simplificar una expresión algebraica.
Un error muy común consiste en el uso de paréntesis cuadrados dentro de una expresión.

```
Simplify[3 x + 4 y + 2 √x + y - 5 x × y + 10 √x + x^3 + 10 x y]
```

$$\text{Simplify}\left[\frac{(6\,x^2-10)}{2}+\frac{6\,x^2-6\,x+12}{-6}-\frac{8\,x\text{^}2+12\,x-20}{4}\right]$$

```
Simplify[((a + b - c) - (a - b + c)) - ((b - c + a) - (b + c - a))]
```

Expand

Expande una expresión algebraica.

```
Expand[(3 x^2 y z - 4 x y^2 z^3) (2 x y^2 z^4)]
```

```
Expand[(a^3 + a b^2 + a^4 + b^2) (a^3 + a b^2 - a^4 - b^2)]
```

```
Expand[
 8 p^3 q^4 r^5 (4 p^2 q^3 - 3 q^5 r^6) - 4 p^3 q^7 r^4 (8 p^2 r - 3 p^3 q - 6 q^2 r^7)]
```

```
Expand[(1 - x) ^10]
```

Operaciones con polinomios

```
Apart[(6 x^3 + 3 x^2 - 24 x - 9)/(2 x - 3)]
```

```
P[x_] := (1 - x)^1000
Coefficient[P[x], x, 1000]
```

```
PolynomialGCD[P[x], 6 x^3 + 3 x^2 - 24 x - 9]
```

```
PolynomialLCM[P[x], 6 x^3 + 3 x^2 - 24 x - 9]
```

```
H[x_] := Tan[x] Abs[Sqrt[x]]
PolynomialQ[H[x], x]
PolynomialQuotient[6 x^3 + 3 x^2 - 24 x - 9, 2 x - 3, x]
PolynomialRemainder[6 x^3 + 3 x^2 - 24 x - 9, 2 x - 3, x]
```

Como práctica resuelva los ejercicios de la página 18

Primera sesión

Factor

Factoriza una expresión algebraica.

```
Factor[x (a + b + c) - y (a + b + c)]
Factor[-3 d p + p^2 + 6 d q - 2 p q + 3 d x - p x]
Factor[12 x^ (2 m) - 7 x^m - 12]
Factor[x^3 + 4 x^2 b + 4 x b^2 - 2 x^2 - 4 x b]
```

```
Clear[i]
For[i = 1, i ≤ 10, Print[Factor[x^2 - i]]; i++]
```

Expresiones algebraicas fraccionarias

Para resolver operaciones con expresiones algebraicas fraccionarias, se utilizan los comandos Simplify, FullSimplify, Factor y Together.

```
Simplify[(2 d^2 - d v - v^2)/(4 d^2 - 4 d v + v^2) * (8 d^2 + 6 d v - 9 v^2)/(4 d^2 - 9 v^2)]
Factor[%]
```

```
Simplify[(a^2 - b^2)/(2 a^2 - 3 a b + b^2) * (2 a^2 + 5 a b - 3 b^2)/(a^2 + 4 a b + 3 b^2) * (a^2 - 2 a b - 3 b^2)/(a^2 - 4 a b + 3 b^2)]
```

```
Together[b/(b^2 - b - 2) - 1/(b^2 + 5 b - 14) - 2/(b^2 + 8 b + 7)]
```

```
Together[(2 - (a+5)/(a+2))/(a^2 - 1) + (a^2 - (3 a^2 - a)/(a+2))/(a^3 + 1)]
```

Como práctica resuelva los ejercicios de la página 30

Evaluando expresiones

```
FullSimplify[1 + 1/x (1 + 1 / x (1 + 1 / x (1 + (1 + 1/x)/x))) /. x → 1 + y^2]
```

```
Together[1 + 1/x (1 + 1 / x (1 + 1 / x (1 + (1 + 1/x)/x))) /. x → 1 + y^2]
```

```
Together[1 + 1/x (1 + 1 / x (1 + 1 / x (1 + (1 + 1/x)/x))) /. {x → 1 + y^2, z → 10}]
```

```
FullSimplify[
 1 + 1/x (1 + 1 / x (1 + 1 / x (1 + (1 + 1/x)/x))) /. {{x → 1 + y^2}, {x → 100}, {x → -2}}]
```

Ecuaciones e inecuaciones

Para resolver ecuaciones con polinomios:

```
Roots[4 (3 x/4 - 1/2) - 1/2 (4 x + 12) == 4, x]
```

Para resolver ecuaciones en general:

```
Solve[a/(2 y) + y^2 == x + (y a)/(2 x), a]
```

```
Solve[10 x^2 + 59 x + 62 == 0 && x ≤ 0, x]
N[%]
```

```
NSolve[10 x^2 + 59 x + 62 == 0, x]
```

```
Solve[8 h^(-6) - 65 h^-3 + 8 == 0, h]
```

```
Solve[Sqrt[a^2 + 4 a + 1] - Sqrt[2 a + 4] == 0, a]
```

Reduce resuelve ecuaciones con radicales e inecuaciones:

```
Reduce[Sqrt[a^2 + 4 a + 1] - Sqrt[2 a + 4] == 0, a]
```

```
Solve[Sqrt[x] + Sqrt[1 + x] - Sqrt[2 + x] == Sqrt[k + x], x]
```

```
Reduce[Sqrt[x] + Sqrt[1 + x] - Sqrt[2 + x] == Sqrt[k + x], x]
```

Solve resuelve también sistemas de ecuaciones lineales:

```
Solve[{16 h - 11 b + 17 == 11 h + 6 b + 17, 9 h - 9 b + 21 == -14 h + 8 b - 23}, {h, b}]
```

Resolviendo inecuaciones en el campo de los números reales:

```
Reduce[(x^2 - 4) (x^2 - 4 x + 4) (x^2 - 6 x + 8) (x^2 + 4 x + 4) < 0, x]
```

```
Reduce[Abs[t + 3] < 4 , t, Reals ]
```

```
Reduce[Abs[t + 3] < 4 , t]
```

```
Reduce[Abs[(a^2 - 3 a - 1)/(a^2 + a + 1)] ≤ 3, a, Reals]
```

```
Reduce[12 x^5 + 56 x^4 - 11 x^3 - 252 x^2 - 127 x + 70 < 0 , x]
```

Como práctica resuelva
los ejercicios de la
página 61 y 68

Definiendo una función

Se puede utilizar también :=.

```
A[r_] = (2 r - 1)/(r^2 + 2 r - 3); {A[3], A[1/2], A[-1], Simplify[A[x^2 + 1]]}
```

ClearAll elimina de la memoria las funciones declaradas.

```
ClearAll[f, g]
f[x_] = 2 x^2; g[x_] = x^2 - 5 x + 6;
{Suma[x_] = Simplify[f[x] + g[x]], Resta[x_] = Simplify[f[x] - g[x]],
 Producto[x_] = Simplify[f[x] * g[x]], Divi[x_] = Simplify[f[x]/g[x]],
 Composition[f, g][x], f[g[x]], Composition[g, f][x], g[f[x]]}
```

Función definida a trozos:

```
H[x_ /; x > 0] = x;
H[x_ /; x ≤ 0] = x^3;
{H[1], H[-4]}
G[x_] := If[x > 0, x, x^3]
{G[1], G[-4]}
```

Graficando pares ordenados

```
ListPlot[{{0, -1}, {1, 2}, {2, 4}}, PlotStyle → PointSize[0.025]]
```

Calcule la distancia entre los puntos (0,-1) y (2,4).
Determine el punto medio del segmento con extremos (1,2) y (2,4).

```
distancia[x1_, y1_, x2_, y2_] := √((x2 - x1)^2 + (y2 - y1)^2)
PM[x1_, y1_, x2_, y2_] := {(x1 + x2)/2, (y1 + y2)/2}
distancia[0, -1, 2, 4]
PM[1, 2, 2, 4]
```

EJEMPLO **1.66** Compruebe que los puntos $(5,-1)$, $(2,5)$, $(-1,-4)$ corresponden a los vértices de un triángulo rectángulo, y determine su área A.

Determina el número de casos:

```
Binomial[3, 2]
```

Forma de resolución:

```
TrRec[x1_, y1_, x2_, y2_, x3_, y3_] =
  If[((distancia[x1, y1, x2, y2])^2 + (distancia[x1, y1, x3, y3])^2 ==
      (distancia[x2, y2, x3, y3])^2) || ((distancia[x1, y1, x2, y2])^2 +
       (distancia[x2, y2, x3, y3])^2 == (distancia[x1, y1, x3, y3])^2) ||
    ((distancia[x1, y1, x3, y3])^2 + (distancia[x2, y2, x3, y3])^2 ==
      (distancia[x1, y1, x2, y2])^2), Return[True], Return[False]];
TrRec[-1, -4, 2, 5, 5, -1]
semip[x_, y_, z_] = (x + y + z)/2;
s = semip[distancia[x1, y1, x2, y2],
  distancia[x1, y1, x3, y3], distancia[x2, y2, x3, y3]];
Area[x1_, y1_, x2_, y2_, x3_, y3_] = √(s (s - distancia[x1, y1, x2, y2])
    (s - distancia[x1, y1, x3, y3]) (s - distancia[x2, y2, x3, y3]));
If[TrRec[-1, -4, 2, 5, 5, -1] == True, N[Area[5, -1, 2, 5, -1, -4]],
 Print["No corresponde a un triángulo rectánculo"]]
```

Determinar si los puntos $(6,12)$, $(0,-6)$ y $(1,-3)$ son o no colineales

```
ListPlot[{{6, 12}, {0, -6}, {1, -3}}, PlotStyle → PointSize[0.025]]
```

```
pen[x1_, y1_, x2_, y2_] = (y2 - y1)/(x2 - x1);
bb[x1_, y1_, x2_, y2_] = y1 - pen[x1, y1, x2, y2] * x1;
y[x_] = pen[6, 12, 0, -6] * x + bb[6, 12, 0, -6];
If[y[1] == -3, Print[True], Print[False]]
```

Segunda sesión

Plot y otros comandos ...

Elabore la gráfica de $y = \dfrac{x}{x^2+1}$ en el intevalo $[-4, 4]$

PlotRange define un rango sobre el eje y.

```
Plot[{x/(x^2 + 1), x^2}, {x, -4, 4}, PlotRange → {-1, 4}]
```

Elabore la gráfica de $y = \begin{cases} x^2 : x \leq 0 \\ -x : x > 0 \end{cases}$ en el intervalo $[-5, 5]$

```
g[x_ /; x ≤ 0] = x^2;
g[x_ /; x > 0] = -x;
Plot[g[x], {x, -5, 5}]
```

O bien:

```
Plot[If[x <= 0, x^2, -x], {x, -5, 5}]
```

Determine si $f(x) = \dfrac{1}{x+1}$ es decreciente en $[-3, 3]$

```
Plot[1/(x + 1), {x, -3, 3}]
```

Se observa que la función es decreciente.

EJEMPLO **1.75** Determinar a de suerte que $3x + ay = 9$ tenga la misma pendiente que la recta que pasa por $(7, -2)$ y $(5, -1)$.

```
Solve[3 x + a y == 9, y]
```

```
Solve[-3/a == pen[7, -2, 5, -1], a]
```

Realice un programa para analizar una función cuadrática.

```
Clear[a, bbb, c];
a = Input["Digite el valor de a: "];
bbb = Input["Digite el valor de b: "];
c = Input["Digite el valor de c: "];
Print["f(x)=", a x^2 + bbb x + c];
Print["Intersecciones con el eje x"];
If[bbb^2 - 4 a c ≥ 0, Print["Las raíces son:"];
 Print["x1= ", N[(-bbb + Sqrt[bbb^2 - 4 a c])/(2 a)], ", x2= ", N[(-bbb - Sqrt[bbb^2 - 4 a c])/(2 a)]],
 Print["No hay raíces reales"]]
If[a > 0, Print["Se abre hacia arriba"], Print["Se abre hacia abajo"]]
Print["El vértice es: ", "(", Simplify[-bbb/(2 a)], ",", Simplify[-(bbb^2 - 4 a c)/(4 a)], ")"]
If[a > 0, Print["En este par ocurre un mínimo absoluto"],
 Print["En este par ocurre un máximo absoluto"] ]
Print["La intersección con el eje y ocurre en: (0," , c, ")"]
t1 = Input["Valor menor del intervalo del plot: "];
t2 = Input["Valor mayor del intervalo del plot: "];
Plot[a x^2 + bbb x + c, {x, t1, t2}]
```

Realice una animación para una función cuadrática.

Se debe añadir <<Graphics`Animation` si la versión es inferior a la 6.

```
Animate[Plot[ x^2 + x + k1, {x, -10, 10}, PlotRange → {-5, 20}],
 {k1, 1, 20, 1}, AnimationRunning → False]
```

EJEMPLO **1.77** Hallar dos números no negativos cuya suma es 30 que hagan máximo el producto del cuadrado de uno por el cubo del otro?

```
Clear[x, y];
y = 30 - x;
Ppro[x_] = x^2 y^3;
Plot[Ppro[x], {x, 0, 31}]
Table[{i, Ppro[i]}, {i, 11, 13, 0.1}]
```

Se intuye un máximo en 12.

Sea $f : \mathbb{R} \to \mathbb{R}$ dada por $f(x) = -4\sqrt[3]{2x + 10}$. Halle la inversa de f

```
Clear[x, y];
Solve[-4 ∛(2 x + 10) == y, x]
% /. {x → y, y → x}
```

Calcule, usando la definición, la derivada de $f(x) = x^2 \operatorname{sen} x$

```
f[x_] = x^2 Sin[x];
Expand[Limit[(f[x + h] - f[x])/h, h → 0]]
D[f[x], x]
```

EJEMPLO **1.82** Sea $f(x) = |x-1| - |x+1|$. Grafique f y determine si es derivable en $a = -1$ y en $a = 1$.

-1 es a la derecha y 1 a la izquierda.

```
f[x_] = Abs[x - 1] + Abs[x + 1];
Plot[f[x], {x, -2, 2}, PlotRange → {-1, 5}]
Limit[(f[-1 + h] - f[-1])/h, h → 0, Direction → -1]
Limit[(f[-1 + h] - f[-1])/h, h → 0, Direction → 1]
Limit[(f[1 + h] - f[1])/h, h → 0, Direction → -1]
Limit[(f[1 + h] - f[1])/h, h → 0, Direction → 1]
```

Se concluye que no es diferenciable en ambos casos.

Suponga que $f(x) = \dfrac{1+\sqrt{1+e^x}}{1+x^2}$. Calcule la tercera derivada de $f(x)$

```
Simplify[∂x,x,x (1 + √(e^x + 1))/(1 + x^2)]
```

Hallar $\dfrac{dy}{dx}$ en el punto $(2,3)$ si $x^2 + xy + 2y^2 = 28$

```
D[x^2 + x y[x] + 2 (y[x])^2 == 28, x];
% /. y'[x] → yp;
Solve[% , yp];
% /. {x → 2, y[x] → 3}
```

EJEMPLO **1.94** Sea $f(x) = 6x - \frac{7x^2}{2} - 12x^3 + \frac{7x^4}{4} + \frac{6x^5}{5}$. Halle los puntos en los que hay tangentes horizontales.

```
f[x_] = 6 x - 7 x^2/2 - 12 x^3 + 7 x^4/4 + 6 x^5/5;
v = Solve[D[f[x], x] == 0, x]
P = {};
For[i = 1, i ≤ Length[v], P = P ⋃ {{x /. v[[i, 1]], f[x] /. v[[i, 1]]}}; i++]
Print["Los puntos estacionarios son: ", P];
```

Calcular, usando la regla de L'Hôpital, $\lim_{x \to 0} \frac{\operatorname{sen} x - x}{x^3}$

```
f[x_] = (Sin[x] - x)/x^3;
While[((Numerator[f[x]] /. x → 0) == 0) && ((Denominator[f[x]] /. x → 0) == 0),
 f[x_] = Together[D[Numerator[f[x]], x]/D[Denominator[f[x]], x]]; Print[f[x]];]
Print[f[0]]
```

EJEMPLO **1.103** Haga un programa en *Mathematica* que calcule los primeros 10 números de Fibonacci.

Al ser una relación de recurrencia no se puede usar =.

```
ClearAll[F];
F[n_] := F[n - 1] + F[n - 2]
F[1] = 1;
F[2] = 1;
Table[{i, F[i]}, {i, 1, 10, 1}]
```

O bien:

```
For[i = 1, i ≤ 10, Print[Fibonacci[i]]; i++]
```

Como práctica resuelva
los ejercicios de la
página 95 y 120

Variables en *Mathematica*
Lectura individual hasta la diapositiva 37 (incluida)

- No se declaran.
- Entero (int): para almacenar datos Z
- Real (float): para almacenar datos IR
- Cadena (string): para almacenar más de un carácter ASCII o Unicode
- Carácter (char): para almacenar un carácter ASCII o Unicode
- Booleano (bool): para almacenar datos cuyo valor es verdadero (true) o false (false)

Identificadores en Mathematica

- Son una secuencia de caracteres que permiten identificar de forma única a cada elemento/objeto de un algoritmo (variables o constantes)
- Las constantes en Mathematica están protegidas, por ejemplo: I (imaginario), E (2.71 …)

Presentan algunas restricciones

- El primer carácter debe ser una letra
- No puede haber espacio dentro de un identificador
- Pueden tener cualquier longitud dentro del límite que imponga el compilador
- Las palabras reservadas (tales como if, while, for, do, else, entre otras) del lenguaje no pueden utilizarse como identificadores
- Un identificador no puede contener : * , : /. no son válidos
- Es case sensitive

Operadores en Mathematica

- Operadores de asignación y de lectura
- Operadores de incremento y decremento
- Operadores aritméticos
- Operadores relacionales
- Operadores lógicos

Operadores de asignación y de lectura

Los programadores tiene dos maneras para hacer que los programas almacenen datos en memoria, ellos son:

- La operación de asignación: la instrucción se indica por un "=". Por ejemplo: x=3;
- La operación de lectura: se utiliza en los programas para ordenarle al computador que debe detener la realización del programa y esperar a que se digite el dato por el teclado. Por ejemplo en Mathematica: nombreCiudad=Input["Digite el nombre de la ciudad"]; edad=Input["Digite la edad"]; salario=Input["Digite el salario"];

Otros tipos de asignaciones

Mathematica maneja otros operadores de asignación, a saber:

- + = : suma al valor de la variable de la izquierda el valor que se encuentra a la derecha, por ejemplo: p+=10; es equivalente a p = p+10;
- – = : resta al valor de la variable de la izquierda el valor que se encuentra a la derecha, por ejemplo: p-=10; es equivalente a p = p-10;
- *= : multiplica al valor de la variable de la izquierda el valor que se encuentra a la derecha, por ejemplo: p*=10; es equivalente a p = p*10;
- / = : divide al valor de la variable de la izquierda el valor que se encuentra a la derecha, por ejemplo: p/=10; es equivalente a p = p/10;

```
p = 1;
p += 10
p -= 10
p *= 20
p / 11 // N
```

Operadores ++ y --

- ++ : incrementa en una unidad el contenido numérico de una variable
- -- : decrementa en una unidad del contenido numérico de una variable

Operadores aritméticos

Operador	Nombre
+,-	+ o – unitario
^	Potenciación
+	Suma
-	Resta
*	Multiplicación
/	División
Mod	Módulo, residuo de la división

Orden de prioridad

- Primera + o – unitario y ^
- Segunda *, /, Mod
- Tercera +, -

Los paréntesis alteran estas prioridades

Operadores relacionales

- Se utilizan para establecer una relación entre dos valores, devolviendo un valor lógico
- Comparan valores del mismo tipo (numérico o alfanumérico)

>	Mayor que
<	Menor que
≥, >=	Mayor o igual que
≤, <=	Menor o igual que
≠, <>, !=	Diferente, Distinto
==	Igual de comparación

Por ejemplo ...

```
a = 10; b = 20; c = 30;
a + b > c
a - b < c
a - b == c
a + b == c
```

Operadores lógicos

- Se utilizan para establecer relaciones entre valores lógicos
- Su resultado es verdadero o falso
- Son los siguientes:
  And &&
  Or ||
  Not !

Prioridad de los operadores lógicos:
Not
And
Or

Idempotencia	$A \wedge A = A$ $A \vee A = A$
Conmutativa	$A \wedge B = B \wedge A$ $A \vee B = B \vee A$
Asociativa	$A \wedge (B \wedge C) = (A \wedge B) \wedge C$ $A \vee (B \vee C) = (A \vee B) \vee C$
Absorción	$A \wedge (B \vee A) = A$ $A \vee (B \wedge A) = A$
Distributiva	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
Ley del ínfimo	$A \wedge \square = \square$ $A \vee \square = A$
Ley del supremo	$A \wedge \blacksquare = A$ $A \vee \blacksquare = \blacksquare$
Complementario	$A \wedge \neg A = \square$ $A \vee \neg A = \blacksquare$

Por ejemplo ...

```
H = 6; J = 12;
(H > 0) && (J > 10)
(H > 25) || (H < 50) && (J < 50)
(H < 4) || (J > 5)
Not[H > 6]
```

Estructuras de control

En Mathematica un algoritmo puede ser escrito con tres estructuras de control básicas:

- Secuenciales
- Selectivas o condicionales
- Repetitivas o de ciclo

Estructura secuencial

Las estructuras secuenciales son aquellas en las que una acción o instrucción sigue a otra en secuencia (la salida de una es la entrada de la siguiente). Es decir:

{ acción 1;
acción 2;
…}

Estructuras selectivas o condicionales

- Son aquellas en las que se evalúa una condición y en función del resultado se realiza una operación
- Se utilizan para tomar decisiones lógicas y se suelen denominar estructuras de decisión o alternativas
- Estas estructuras pueden ser:
  - Simples
  - Dobles
  - Múltiples

Estructura de selección simple

- A esta estructura de control se denomina instrucción If, su sintaxis en Mathematica 5.0 es la siguiente:
  If [condición, acción 1; …; acción n;]
- Por ejemplo:

```
J = Input["Digite el valor de J"];
If[J > 3, J = J * 0; Print[J]]
```

Estructura de control selectiva doble

- Permite elegir entre dos opciones o alternativas posibles en función del cumplimiento de una determinada condición. Su sintaxis se presenta a continuación:

  If [condición, acción 1; …; acción n, acción 1'; …; acción m';]

- Si la condición es falsa se ejecutan las acciones primas ('). Por ejemplo:

```
J = Input["Digite el valor de J"];
If[J > 12, J = J * 0; Print[J], Print[J + 3]]
```

Estructura de control alternativa múltiple

- Se utiliza cuando existe la necesidad de contemplar más de dos elecciones posibles. Se denomina instrucción Switch. Su sintaxis en Mathematica 5.0 es:

  Switch [expresión, valor1, instrucción1, valor2, instrucción2, … ,valor n, instrucción n]

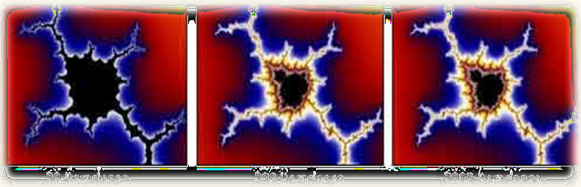
- Por ejemplo:

```
x = 32;
Switch[Mod[x, 3], 0, x = x + 1, 1, x = x + 2, 2, x = x + 3]
```

```
35
```

Estructuras de control iterativas

- Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces
- Son denominadas también bucles, lazos o ciclos
- Por ejemplo, al generar un fractal es necesario ejecutar un número determinado de iteraciones:



Estructuras de control iterativas

- Toda estructura de control de repetición está constituida por:
  El cuerpo del ciclo
  La iteración
  Y la condición de salida o término del ciclo
- Mathematica posee tres tipos de estructuras repetitivas que estudiaremos: While, Do y For

Instrucción While

- La estructura de control While posee la siguiente sintáxis en Mathematica:
  While [condición, acción 1; acción 2;]
- Se ejecuta reiteradamente (acciones), mientras la condición sea verdadera. La condición se evalúa antes de ejecutar las acciones y si es falsa no se ejecuta y sale del ciclo. Por ejemplo:

```
i = 1;
While[i ≤ 10, Print[Prime[i]]; i++]
```

Instrucción Do

- Esta instrucción iterativa ejecuta reiterativamente una instrucción dada una variación del o los parámetros de los cuáles la instrucción depende. Su sintaxis se describe a continuación:
  Do [instrucciones, {parámetro, inicio, final}];
- Cuando se ejecuta el valor del parámetro en "final" se sale del ciclo
- El siguiente ejemplo, despliega la gráfica de la función Sen[n*x] cuando el parámetro x varía de de 0 a dos pi y el parámetro n de 1 a tres con incrementos de 0.25

```
Do[Print[Plot[Sin[n x], {x, 0, 2 π}]], {n, 1, 3, 0.5}]
```

Instrucción For

La sintaxis de esta estructura de control de repetición en Mathematica es:

- For [instrucciones 1, expresión; instrucciones 2; instrucciones 3;]
  Donde:
  - instrucciones 1: se ejecutará una sola vez al inicio del ciclo, generalmente se realizan inicializaciones y declaraciones de variables.
  - expresión: es evaluada en cada ciclo y dependiendo del valor que devuelva, el bucle continúa ejecutándose (valor de la evaluación True o False)
  - instrucciones 2: es ejecutado siempre en cada ciclo al terminar de ejecutar todas las instrucciones 2 que pertenecen al bucle For en cuestión. Por lo general puede contener alguna actualización para las variables de control
  - instrucciones 3: grupo de instrucciones que se requiere se ejecuten repetidamente

Ejemplo 1

Diseñe un algoritmo que calcule el producto de dos números enteros A y B

```
a = Input["Digite el valor de a:"];
While[IntegerQ[a] == False, a = Input["Digite el valor de a:"]]
b = Input["Digite el valor de b:"];
While[IntegerQ[b] == False, b = Input["Digite el valor de b:"]]
proAB  = a * b;
Print[proAB];
```

Ejemplo 2

Diseñe un algoritmo que resuelva el siguiente problema: suponga que un individuo quiere invertir su capital en un banco y desea saber cuanto dinero ganará después de un mes si el banco paga a razón de 2% mensual

```
cap = Input["¿Cuál es el capital?"];
ganancia = Abs[cap] * 0.02; (* Abs calcula el valor absoluto *)
Print[ganancia];
```

Ejemplo 3

Diseñe un algoritmo que resuelva el siguiente problema: un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones

```
suelB = Input["Digite el sueldo base"];
venta1 = Input["Digite el valor de la primera venta"];
venta2 = Input["Digite el valor de la segunda venta"];
venta3 = Input["Digite el valor de la tercera venta"];
totalventa = Abs[venta1] + Abs[venta2] + Abs[venta3];
comision = totalventa * 0.10;
sueldoR = Abs[suelB] + comision;
Print["El sueldo es de: ", sueldoR, ", la comisión fue de: ", comision];
```

Ejemplo 4

Diseñe un método que reciba un número y devuelva true si el número es positivo o false sino

```
Positivo[num_] := If[num ≥ 0, Return[True], Return[False]]
Positivo[-5]
```

Ejemplo 5

Diseñe un método que determine las soluciones a una ecuación de primer grado a X + b = 0

```
Solun[a_, b_] := If[a != 0, Print[-b / a],
  If[b == 0, Print["La solución es IR"], Print["La solución en vacía"]]]
Solun[
 1,
 2]
```

Tercera sesión

Ejemplo 6

Diseñe un método que reciba las longitudes de los tres lados de un triángulo y determine si es equilátero, isósceles o escaleno

```
TipoTrian[a_, b_, c_] :=
 If [a == b && b == c, Print["El triángulo es equilátero"], If [a == b || b == c || c == a,
Print["El triángulo es isósceles"], Print["El triángulo es escaleno"]]]
TipoTrian[1, 1, 1]
```

Mejorar este código para verificar si las medidas corresponden o no a un triángulo

```
TipoTrian[a_, b_, c_] := If[a + b > c && a + c > b && b + c > a,
  If [a == b && b == c, Print["El triángulo es equilátero"], If [a == b || b == c || c == a,
Print["El triángulo es isósceles"], Print["El triángulo es escaleno"]]],
  Print["Las longitudes no satisfacen la desigualdad triangular"]]
TipoTrian[
 1,
 1,
 4]
```

Ejemplo 7

Diseñe un método que reciba tres números enteros y devuelva el mayor de ellos

```
EncuentraMayor[ a_, b_, c_] :=
 If[IntegerQ[a] == True && IntegerQ[b] == True && IntegerQ[c] == True,
  If [a > b, If [a > c, Print[a], Print [c] ], If [b > c, Print[b],
      Print[c]]], Print["Hay un dato que no es un entero"]]
EncuentraMayor[656, 10 000, 400 000]
```

Mathematica cuenta con los comandos Max y Min

```
Mayor[ a_, b_, c_] = Max[a, b, c];
Menor[ a_, b_, c_] = Min[a, b, c];
Mayor[656, 10 000, 400 000]
Menor[656, 10 000, 400 000]
```

Ejemplo 8

Diseñe un método que calcule la cantidad de días de un mes

```
bisiesto[ a_] := If [Mod[a, 4] == 0 && Mod[a , 100] != 0, v = True,
  If [Mod[a , 100] == 0 && Mod[a, 400] == 0, v = True, v = False]]
diasMes[mes_, a_] := Switch [mes, 4 , Print[30], 6 , Print[30], 9 , Print[30],
  11, Print[30], 2, If[bisiesto[a] == True, Print[29], Print[28]],
  1, Print[31], 3, Print[31], 5, Print[31], 7,
  Print[31], 8, Print[31], 10, Print[31], 12, Print[31]]
diasMes[
 2,
 2011]
```

Ejemplo 9

Diseñe un método “Cal” a través del cual se implementen las cuatro operaciones aritméticas básicas: suma, resta, multiplicación y división entera. Los datos de entrada para el método “Cal” serán los dos operandos seguidos del operador, este último representado por un valor de tipo char (‘+’, ‘-’, ‘*’ y ‘/’ respectivamente)

```
Cal[o1_, o2_, o_] := Switch[o, "+", Print[o1 + o2], "-", Print[o1 - o2], "*",
  Print[o1 * o2], "/", If[o2 ≠ 0, Print[o1 / o2], Print["Indefinido"]]]
Cal[4, 1, "+"]
Cal[4, 1, "-"]
Cal[4, 2, "*"]
Cal[4, 0, "/"]
```

Ejemplo 10

Diseñe un método que calcule la media aritmética de una lista de N números positivos

```
con = 1;
suma = 0;
Prom[ n_] := While [con <= n, num = Input["Digite el valor"];
  While[num ≤ 0, num = Input["Digite el valor"]];
  suma = suma + num;
  If[con == n, Print[N[suma / n]]];
  con = con + 1]
Prom[3]
```

La lista se puede pasar como parámetro del método

```
con = 1;
suma = 0;
Prom[ L_List] := While [con <= Length[L], suma = suma + L[[con]];
  If[con == Length[L], Print[N[suma / Length[L]]]]; con = con + 1]
L = {1, 2, 3, 4, 5, 6, 7, 8};
Prom[L]
```

Las inicializaciones pueden quedar dentro de la función por medio de Module

```
Prom[ L_List] =
  Module[{con = 1, suma = 0}, While [con <= Length[L], suma = suma + L[[con]];
    If[con == Length[L], Print[N[suma / Length[L]]]]; con = con + 1]];
L = {1, 2, 3, 4, 5, 6, 7, 8};
Prom[L]
```

Ejemplo 11

Escriba un método que reciba dos valores m y n e imprima todos los dígitos pares mayores o iguales que n y menores que m

```
ParesN[n_, m_] := If[Floor[n] < Floor[m], num = Floor[n];
  While[num <  Floor[m], If[Mod[num , 2] == 0, Print[num]];
   num = num + 1]]
ParesN[2.5, 20.9] (* Floor calcula la parte entera de un número real *)
```

Ejemplo 12

Escriba un método DivN que reciba un número entero n y escriba todos sus divisores

```
Divi[n_] := For[i = 1, i ≤ Floor[n], If[Mod[Floor[n], i] == 0, Print[i]];
  i++]
Divi[
 10.6]
```

O bien:

```
Div[n_] := Divisors[Floor[n]]
Div[10.6]
```

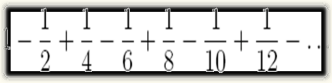
```
{1, 2, 5, 10}
```

O bien, con el uso del Module y generando un vector de divisores:

```
Di[n_] := Module[{L = {}},
  For[i = 1, i ≤ Floor[n], If[Mod[Floor[n], i] == 0, L = Append[L, i]];
   If[i == Floor[n], Print[L]]; i++]]
Di[10.6]
```

Ejemplo 13

Diseñar un método para calcular la suma de los primeros n términos de la serie:



```
SumaSerie[n_] := For[i = 1, i < Floor[n], i++, If[i == 1, suma = 1; signo = -1;];
   suma = suma + signo / (2 * i);
   signo = -signo; If[i == Floor[n] - 1, Return[suma]]];
SumaSerie[5]
```

Ejemplo 14

Diseñe un método que reciba dos números y determine si son “amigos”. Dos números son amigos si cada uno de ellos es igual a la suma de los divisores del otro, excluyendo al número

```
Sumdiv[n_] := For[j = 1, j <= n, j++, If[j == 1, suma = 0];
   If[Mod[n, j] == 0, suma = suma + j];
   If[j == n, Return[suma - n]]]
SonAmigos[n_, m_] :=
 If[Sumdiv[n] == m && Sumdiv[m] == n, Print["Son amigos"], Print["No son amigos"]]
SonAmigos[284, 220]
```

Ejemplo 15

Considerando la suma de los divisores propios existen varios tipos de números:

- Números defectivos (1): la suma de los divisores propios es menor que el número
- Números abundantes (2): la suma de los divisores propios es mayor que el número
- Números perfectos (3): la suma de los divisores propios es igual a sí mismo

Diseñe un método que reciba un número N y devuelva un 1 si es defectivo, un 2 si es abundante y un 3 si es perfecto

```
TipNum[ N_] :=
 Module[{S = 0, i = 1}, While [i <= N / 2, If [ Mod[N , i] == 0, S = S + i;] ;
   i ++;
   If[i > N / 2, If[S > N, Return[2], If[S < N, Return[1], Return[3]]]]]]
TipNum[10]
```

Cuarta sesión

Ejemplo 16

Los polinomios de Legendre se pueden calcular mediante las fórmulas:

- P0 = 1
- P1 = x
- Pn = ((2*n-1)/n) * x * Pn-1 - ((n-1) / n) * Pn-2

Donde n = 2,3,4,.... y x es un número real, que se encuentra entre -1 y 1. Escribir un método que reciba como parámetros de entrada n y x genere el Pn correspondiente, no se ocupe de validar x

```
Legendr[x_, n_] := Module[{i = 2}, While [i <= n, If[i == 2, pn2 = 1; pn1 = x;] ;
   pn = ((2 * i - 1) / i) * x * pn1 - ((i - 1) / i) * pn2; pn2 = pn1; pn1 = pn;
   i++;
   If[i == n, Return[pn]]]]
Legendr[2, 5]
```

$$\frac{443}{8}$$

Ejemplo 17

Diseñe un método que dadas dos circunferencias (con centros en coordenadas “x”, “y” y su radio) calcule cuántos puntos en común tienen dichas circunferencias (uno, ninguno, dos o infinitos puntos)

```
PuntosDeCorte[x1_, y1_, r1_, x2_, y2_, r2_] :=
 If[Sqrt[(x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)] == 0, If[r1 == r2,
   Print["Existen infinitos puntos de corte"], Print["No hay puntos de corte"]],
  If[r1 + r2 > Sqrt[(x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)] ,
   Print["Existen dos puntos de corte"],
   If[r1 + r2 == Sqrt[(x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)] ,
    Print["Existe un punto de corte"], Print["No hay puntos de corte"]]]]
PuntosDeCorte[
 1,
 1,
 1,
 1,
 1,
 1]
```

Ejemplo 18

Conjeture el valor de la traza de la matriz $n \times n$ *:*

$$A = \begin{pmatrix} 2 & 1 & 1 & \cdots & 1 \\ 0 & 4 & 1 & \cdots & 1 \\ 0 & 0 & 6 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2n \end{pmatrix}$$

En las versiones inferiores a la 7, el comando Join debe reemplazarse por AppendColumns

```
M = {{}};
H = {{}};
n = 2;
While[n ≤ 10, For[i = 1, i <= n, For[j = 1, j <= n,
    If[i == j, H = Insert[H, 2 i, {1, j}]]; If[i < j, H = Insert[H, 1, {1, j}]];
    If[i > j, H = Insert[H, 0, {1, j}]];
    j++]; If[i == 1, M = H];
  If [i != 1, M = Join[M, H]];
  H = {{}}; i++];
 Print[Tr[M]]; n++]
```

$Tr(A) = n(n+1) \;\; \forall n, \; n \in \mathbb{N}, \; n \geq 2$

Ejemplo 19

Para la siguiente matriz, calcule su inversa usando:

a) Operaciones elementales por filas

b) El método por cofactores

$$T = \begin{pmatrix} 2 & 1 & 3 & 4 & 2 \\ 1 & 2 & 3 & -1 & 4 \\ 2 & 3 & 2 & 1 & 4 \\ 5 & -1 & 3 & 2 & 6 \\ 3 & 1 & 0 & -5 & 2 \end{pmatrix}$$

Jordan-Gauss:

```
T = {{2, 1, 3, 4, 2, 1, 0, 0, 0, 0},
     {1, 2, 3, -1, 4, 0, 1, 0, 0, 0},
     {2, 3, 2, 1, 4, 0, 0, 1, 0, 0},
     {5, -1, 3, 2, 6, 0, 0, 0, 1, 0},
     {3, 1, 0, -5, 2, 0, 0, 0, 0, 1}};

For[j = 1, j ≤ 5, For[i = 1, i ≤ 5, If[i == j, T[[i]] = 1/T[[i, j]] * T[[i]]];
  If[i ≠ j, T[[i]] = T[[i]] - T[[i, j]] * 1/T[[j, j]] * T[[j]]];
  Print[MatrixForm[T]]; i++];
 j++; If [j == 6, Print[MatrixForm[T]]]]
```

Por cofactores:

$$T = \begin{pmatrix} 2 & 1 & 3 & 4 & 2 \\ 1 & 2 & 3 & -1 & 4 \\ 2 & 3 & 2 & 1 & 4 \\ 5 & -1 & 3 & 2 & 6 \\ 3 & 1 & 0 & -5 & 2 \end{pmatrix};$$

$$NADJ = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix};$$

```
u = 1;
l = 1;
For[i = 1, i ≤ 5, For[j = 1, j ≤ 5, For[h = 1, h ≤ 5,
   For[k = 1, k ≤ 5, If [h ≠ i && k ≠ j, M = ReplacePart[M, T[[h, k]], {u, l}];
     If [u ≤ 4, l++]; If[l == 5, u++; l = 1]]; k++]; h++];
  u = 1; l = 1;
  NADJ = ReplacePart[NADJ, (-1)^(i+j) Det[M], {i, j}]; j++];
 i++; If [i == 6, Print[MatrixForm[1/Det[T] Transpose[NADJ]]]]]
```

Ejemplo 20

Si $M = (a_{ij}) \in M_n(\mathbb{R})$ tal que $a_{ij} = \begin{cases} 1 \text{ si } i+j = n+1 \\ 0 \text{ si } i+j \neq n+1 \end{cases}$ conjeture una fórmula para calcular $\det(M)$.

```
M = {{}};
H = {{}};
n = 2;
While[n ≤ 10,
 For[i = 1, i <= n, For[j = 1, j <= n, If[i + j == n + 1, H = Insert[H, 1, {1, j}]];
   If[i + j != n + 1, H = Insert[H, 0, {1, j}]];
   j++]; Print[H];
  If[i == 1, M = H];
  If [i != 1, M = Join[M, H]];
  H = {{}}; i++];
 Print[Det[M], " +++++ ", M // MatrixForm]; n++]
```

$$\det(M) = (-1)^{\frac{n(n-1)}{2}}$$

Ejemplo 21

Si se generaliza la matriz definida en el ejercicio anterior como $M = (a_{ij}) \in M_n(\mathbb{R})$ *con n par, tal que:*

$$a_{ij} = \begin{cases} \alpha \text{ si } i = j \\ \beta \text{ si } i + j = n + 1 \\ 0 \text{ en las demás entradas} \end{cases}$$

conjeture si es posible obtener una relación que exprese el $\det(M)$ *en términos de* α, β *y* n.

```
M = {{}};
H = {{}};
n = 2;
While[n ≤ 10, If[Mod[n, 2] == 0,
  For[i = 1, i <= n, For[j = 1, j <= n, If[i == j, H = Insert[H, α, {1, j}]];
    If[i + j == n + 1, H = Insert[H, β, {1, j}]];
    If[i + j != n + 1 && i ≠ j, H = Insert[H, 0, {1, j}]]; j++];
   If[i == 1, M = H];
   If [i != 1, M = Join[M, H]];
   H = {{}}; i++];
  Print[Factor[Det[M]]]]; n++]
```

$$\det(M) = (\alpha - \beta)^{\frac{n}{2}} (\alpha + \beta)^{\frac{n}{2}}$$