Programación con *Mathematica 5.0*

Máster Enrique Vílchez Quesada
Universidad Nacional
Escuela de Informática
evilchez@una.ac.cr

Resumen: *Mathematica 5.0* fue desarrollado por la compañía Wolfram Research, Inc constituye una aplicación de cálculo simbólico con su propio lenguaje de programación de alto nivel. Una de las ventajas ofrecidas por el entorno de programación de este software, radica en la reutilización de múltiples funciones para el tratamiento de diversas áreas en matemática, tales como: cálculo diferencial e integral, álgebra lineal, teoría de números y estadística. En este documento se exponen las técnicas de programación y los comandos más importantes, para la creación de pequeñas aplicaciones con fines didácticos en la enseñanza y el aprendizaje de la programación estructurada.

Palabras clave: enseñanza, aprendizaje, programación, estructurada, *Mathematica*.

1. INTRODUCCIÓN

La programación estructurada es un tema de vital importancia para futuros docentes en el área de la enseñanza de la matemática y desde luego, futuros profesionales de la Informática.

En particular, en el contexto del curso Fundamentos de Informática impartido a estudiantes de primer ingreso de las carreras Ingeniería en Sistemas de Información e Informática Educativa de la Universidad Nacional de Costa Rica, el reto que impone la enseñanza de la programación estructurada exige metodologías donde el alumno pueda aprender los fundamentos básicos de la lógica que se utiliza para resolver un problema de forma algorítmica, además de todos los recursos que tiene a su disposición para ello, en un lenguaje de programación.

En este sentido, *Mathematica 5.0* ofrece la posibilidad de simular la resolución de problemas de forma estructurada utilizando el lenguaje de programación que integra, constituyéndose éste en un recurso fundamental para la comprensión de la funcionalidad de cualquier lenguaje de programación y posteriormente de la lógica aplicada en el paradigma orientado a objetos.

2. OBJETIVOS

2.1 Objetivo general

Desarrollar experiencias de enseñanza y aprendizaje para el curso Fundamentos de Informática, utilizando el lenguaje de programación del software *Mathematica 5.0* como herramienta de mediación para el aprendizaje de la programación estructurada.

2.2 Objetivos específicos

- Explicar teóricamente elementos básicos de la programación estructurada.
- Explicar la estructura y sintaxis de las expresiones que se utilizan para programar mediante el software *Mathematica 5.0.*
- Explicar el uso de operadores.
- Resolver y ejemplificar a través del uso del software *Mathematica 5.0* ejercicios típicos de la programación estructurada.

3. MARCO TÉORICO

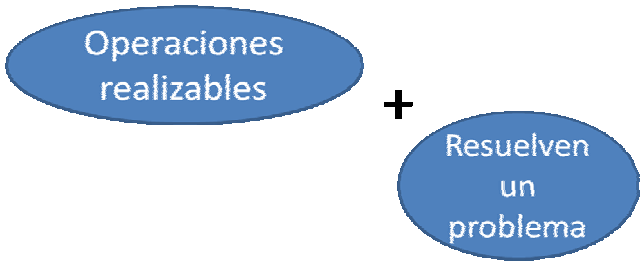
A continuación se presenta algunos elementos básicos fundamentales dentro de la programación estructurada. La teoría abarca desde las nociones primarias del concepto de algoritmo hasta el uso de estructuras de control secuencial, condicional y de bucle.

3.1 Introducción a los algoritmos

Un programador debe aprender cómo utilizar una computadora. Debe aprender a resolver problemas teniendo la claridad sobre los pasos a seguir. Esto implica:

- Dividir el problema en otros más simples.
- Identificar los pasos a seguir para resolver cada subproblema.
- Unificar el esquema de solución para resolver el problema original.

El procedimiento anterior estructura una serie de pasos cuyo objetivo principal es resolver de manera sistemática el problema original y este es precisamente el concepto de algoritmo. ¿Qué es un algoritmo?, es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema, es decir;



Un algoritmo presenta las siguientes características:

- Definido: el mismo proceso ejecutado varias veces nos conduce al mismo resultado.

- Finito: tiene un determinado número de pasos.
- Preciso: los pasos tienen un orden.

Siglo IX La palabra algoritmo deriva de la traducción al latín de la palabra árabe Al Khowrizmi. Nombre del matemático árabe que enunció reglas paso a paso para efectuar operaciones aritméticas con números decimales.

¿Cómo se crea un algoritmo?

- Se determina el problema.
- Se identifican las entradas y las salidas.
- Se identifican los procesos donde dadas las entradas producen las salidas.

Los algoritmos utilizan procesos y los procesos se conceptualizan como:

- Son acciones que se pueden descomponer en otras más simples, es decir, un conjunto de acciones o pasos.
- Una acción tiene una duración limitada y produce un resultado bien definido y concreto.
- Los procesos pueden ser secuenciales o paralelos.

Al resolver un problema de forma algorítmica es necesario utilizar un lenguaje preciso que no produzca ambigüedad. El lenguaje natural con el que nos comunicamos normalmente es muy impreciso. Para ello se utiliza:

- Pseudocódigo: lenguaje natural sin imprecisiones.
- Diagramas de flujo de datos (Dfd) o organigramas: se usan símbolos para describir el flujo lógico del algoritmo.
- Diagramas de Nassi-Schneiderman o Diagramas N-S: los pasos sucesivos se escriben en cajas con distintas formas.

Los diagramas de flujo se caracterizan por ser imágenes gráficas de la secuencia de acciones requeridas para resolver un problema. Tienen la desventaja de funcionar para programas cortos y son difíciles de modificar. En la siguiente imagen se muestran los imágenes comúnmente utilizadas para su diseño.

El pseudocódigo permite describir los algoritmos en un lenguaje que se parece más al lenguaje utilizado para escribir programas de computadora, es decir, un lenguaje de pseudoprogramación; una imitación del código de las computadoras. Es fácil de codificar a cualquier lenguaje de programación. Especifica los pasos lógicos requeridos para resolver un problema dado, sin seguir reglas precisas de ningún lenguaje de programación. Es una técnica muy popular que no tiene los inconvenientes de tener que dibujar símbolos y agrupa mucha información en poco espacio.}

Cuando se programa es necesario almacenar la información que se procesa en la memoria, esto se hace a través del uso de variables y constantes.

Las variables constituyen un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso. Las variables se declaran en la mayoría de los lenguajes (*Mathematica 5.0* es una excepción a esta regla), la declaración de una variable consiste en asignarle nombre (identificador) y tipo.

Tipos de variables:

- 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).

Las variables se clasifican de la siguiente forma:

- **Por su contenido**
  - Variables numéricas

    a = 0.15;

    pi = 3.1416;

    costo = 2500;
  - Variables lógicas
  - Variables alfanuméricas

    letra = “a”;

    apellido = “lopez”;

    direccion = “Av. Segunda”;

- **Por su uso**
  - Variables de trabajo
  - Contadores
  - Acumuladores

De acuerdo con su contenido existen tipos básicos de variables:

- Tipo bool
  - Almacena datos lógicos: verdadero o falso
  - Utiliza un byte
- Tipo float
  - Con punto decimal
  - float utiliza cuatro bytes con precisión simple
  - double, long double utiliza ocho bytes con precisión double

Para la declaración de variables es necesario caracterizarlas por un tipo (por ejemplo: **int**) y un nombre simbólico denominado “identificador” (por ejemplo: añoNacimiento).

- Ejemplos:

  string nombre;

  string apellidos;

  int añoNacimiento;

  int unAño;

El tipo determina:

- La clase de valores que puede tomar una variable (valores enteros en el caso de añoNacimiento).
- Las operaciones en las que puede participar (aritméticas en añoNacimiento).

En la mayoría de los lenguajes las variables tienen que ser definidas antes de ser utilizadas. Toda variable tiene que ser definida una única vez en un mismo bloque o segmento de código y a partir de esta definición puede ser utilizada. Las variables son declaradas escribiendo su tipo, seguido del identificador de la misma terminando con un punto y coma.

Por otra parte, una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa. Es necesario inicializar las constantes al principio del algoritmo. Esto determina el tipo de la constante. Por ejemplo:

Los identificadores son una secuencia de caracteres que permiten identificar de forma única a cada elemento/objeto de un algoritmo (variables o constantes). 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.
- Algunos lenguajes no distinguen entre caracteres en mayúsculas y minúsculas (Pascal) y otros sí (C, Java).
- Un identificador no puede contener : * , : /. no son válidos.

Algunos lenguajes son *case sensitive* (*Mathematica 5.0* lo es), es decir, sensibles al uso de las mayúsculas o minúsculas, por ejemplo: Color - color – coLor, en *Mathematica 5.0* representarían identificadores distintos.

Cuando se resuelve un problema de forma algorítmica se divide el problema en otros más simples, es decir, se comienzan a crear pequeños programas que dan solución al problema macro, a estos pequeños segmentos de código se les denomina métodos. Un método consiste en una serie de sentencias para llevar a cabo una acción, a través de un conjunto de parámetros de entrada (variables) que devolverán un valor de salida (o valor de retorno) de algún tipo. Los resultados de los métodos se muestran de dos formas:

- Si el algoritmo devuelve un único valor: return o retorne.
- Si el algoritmo no devuelve ningún valor: en este caso la descripción del método empieza con la palabra void (nulo), son utilizados para realizar operaciones, por lo tanto reciben el nombre de procedimientos.

Un algoritmo puede ser escrito con tres estructuras de control básicas:

- Secuenciales
- Selectivas o condicionales
- Repetitivas o de ciclo

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;
…….
}
```

Cuando se escribe un programa existe una semántica y una sintaxis que el programador debe respetar, pues esto permite una única interpretación de las instrucciones sea para otro programador o bien para que la computadora ejecute lo que se requiere.

3.2 Operadores

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

- La operación de asignación: la instrucción se indica por un "=", o bien una flecha "←"**.** Por ejemplo: x=3; x←3; *Mathematica 5.0* no utiliza el operador "←"**.**
- 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 5.0*: nombreCiudad=input["Digite el nombre de la ciudad"]; edad=input["Digite la edad"]; salario=input["Digite el salario"];

En *Mathematica 5.0* esta operación se ejecuta por medio de la instrucción Print.

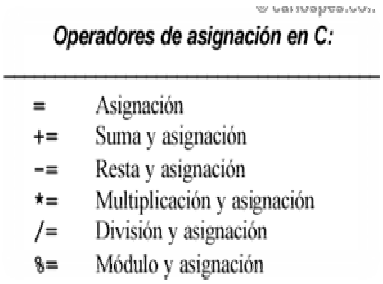
Existen diferentes tipos de asignaciones:

- Simples: consiste en pasar un valor constante a una variable, por ejemplo "a=3;".
- Contador: consiste en usar la variable como un verificador del número de veces de realización de un proceso, por ejemplo "a++; o a = a+1;".

- Acumulador: consiste en usar la variable como un sumador en un proceso, por ejemplo “s=s+5”.
- De trabajo: puede recibir el resultado de una operación matemática que involucre muchas variables, por ejemplo “a = c + b*2/4;”.

Mathematica 5.0 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;



Operadores de incremento y decremento

- ++ : incrementa en una unidad el contenido numérico de una variable
  - a=++b; | Se incrementa “b” y luego se asigna a “a”
  - a=b++; | Se asigna a “a” el valor de “b” y luego se incrementa “b”
- -- : decrementa en una unidad del contenido numérico de una variable
  - a=--b; | Se decrementa “b” y luego se asigna a “a”
  - a=b--; | Se asigna a “a” el valor de “b” y luego se decrementa “b”

Ejemplos de asignaciones. Determine el valor de “b” en cada una de las siguientes asignaciones: a = 3; b = (a++)*5; a = 3; b = (++a)*5; Solución:

- b = 3*5 = 15, a = 4
- b = 4*5 = 20, a = 4

En programación estructurada se utilizan “expresiones”. Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Cada expresión toma un valor que se determina tomando los valores de las variables y

constantes implicadas y la ejecución de las operaciones. Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican en:

- Aritméticas
- Relacionales
- Lógicas

Operadores aritméticos: permiten la realización de operaciones matemáticas con los valores de las variables y las constantes. En *Mathematica 5.0* tenemos:

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

Estos operadores respetan el siguiente orden de prioridad**:**

- Primera + o – unitario y ^
- Segunda *, /, %
- Tercera +, -
- Los paréntesis alteran estas prioridades:

Operadores relacionales: se utilizan para establecer una relación entre dos valores, devolviendo un valor lógico. Si la relación es verdadera “True”, o si es falsa “False”. Comparan valores del mismo tipo (numérico o alfanumérico). Tienen el mismo nivel de prioridad. Son los siguientes:

$>$	Mayor que
$<$	Menor que
$\geq$, >=	Mayor o igual que
$\leq$, <=	Menor o igual que
$\neq$, <>, !=	Diferente, Distinto
==	Igual de comparación

Por ejemplo, suponga que a = 10, b = 20 y c = 30. Veamos el valor de verdad de algunas expresiones. En *Mathematica 5.0*:

```
In[23]:= a = 10; b = 20; c = 30;

In[24]:= a + b > c
         a - b < c
         a - b == c
         a + b == c

Out[24]= False

Out[25]= True

Out[26]= False

Out[27]= True
```

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	$A \wedge \neg A = \square$ $A \vee \neg A = \blacksquare$

En *Mathematica 5.0*, por ejemplo:

```
In[91]:= H = 6; J = 12;

In[110]:= 2 * H ≤ J
          2 * H - 1 < J
          (H > 0) && (J > 10)
          (H > 25) || ((H < 50) && (J < 50))
          (H < 4) || (J > 5)
          Not[H > 6]

Out[110]= True

Out[111]= True

Out[112]= True

Out[113]= True

Out[114]= True

Out[115]= True
```

3.3 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

Las condiciones se especifican usando expresiones lógicas.

3.3.1 Estructura condicional 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:

```
In[119]:= If[J > 3, J = J*0; Print[J];]

          0
```

3.3.2 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 = 12;
If[J > 12, J = J*0; Print[J],
 Print[J+3];]

15
```

3.3.3 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:

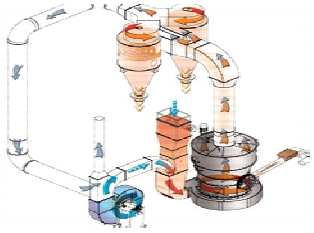
```
Switch[Mod[x, 3], 0, a, 1, b, 2, c]

30
```

3.4 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. Las estructuras iterativas por tanto, permiten ejecutar una o más acciones un número determinado de veces. Son denominadas también *bucles, lazos o ciclos*. Por ejemplo, al generar un fractal es necesario ejecutar un número determinado de iteraciones:

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 5.0 posee tres tipos de estructuras repetitivas que estudiaremos: While, Do y For.

3.4.1 Instrucción While

La estructura de control While posee la siguiente sintáxis en *Mathematica 5.0*:

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 ++]
2
3
5
7
11
13
17
19
23
29
```

Imprime los primeros diez números primos.

3.4.2 Instrucción Do

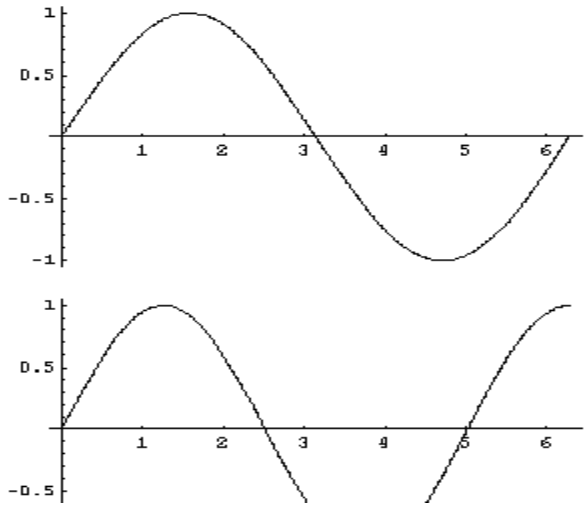
Esta instrucción iterativa ejecuta reiterativamente una instrucción dada una variación del o los parámetros de los cuáles la instrucción depende. Su sintaxis se describe a continuación:

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.

```
In[130]:= Do[Plot[Sin[n x], {x, 0, 2 π}], {n, 1, 3, 0.25}];
```



3.4.3 Instrucción For

La sintaxis de esta estructura de control de repetición en *Mathematica 5.0* 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.

En la siguiente sección se resuelven ejemplos típicos de los procesos de enseñanza y aprendizaje de programación estructurada del curso Fundamentos de Informática, que se imparte a los estudiantes de las carreras Ingeniería en Sistema de Información e Informática Educativa de la Escuela de Informática de la Universidad Nacional de Costa Rica.

4. MARCO METODOLÓGICO

Se abordan ejemplos específicos para la enseñanza y el aprendizaje de la programación estructurada utilizando como herramienta de mediación del software *Mathematica 5.0.*

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

```
a=Input["a"];
b=Input["b"];
proAB  = a * b;
Print[proAB];
```

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=cap * 0.02;
Print[ganancia];
```

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=venta1+venta2+venta3;
comision=totalventa*0.10;
sueldoR=suelB+comision;
Print[sueldoR,comision];
```

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

```
Positivo[num_]:= If[num>0,
Print["Verdadero"],Print["Falso"]];
```

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 R"], Print["La solución en vacía"]]];
```

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

```
TipoTrian[a_,b_,c_]:=If [a==b&&b==c, Print["Equilátero"],If
[a==b||b==c||c==a,Print["Isósceles"], Print["Escaleno"]];]
```

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

```
EncuentraMayor[ a_, b_, c_]:= If [a > b, If [a > c, Print[a],Print
[c] ], If [b > c, Print[b], Print[c]]]
```

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]]
```

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], "/", Print[o1/o2]]
```

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

```
con=1;
suma=0;
Prom[ No_]:=While [con<= No, num=Input["Digite el
valor"];suma = suma + num; If[con == No,
Print[N[suma/No]]]; con= con + 1;];
```

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[n<m, num=n; While [num < m,
    If[Mod[num ,2] == 0, Print[num]];num = num +1 ;];]
```

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

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

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

$$1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{6} + \frac{1}{8} - \frac{1}{10} + \frac{1}{12} - \ldots$$

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

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]];]
SonAmigos[n_,m_]:=If[Sumdiv[n]==Sumdiv[m], Print["Son
amigos"], Print["No son amigos"]];
```

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.

```
S=0; i=1;
TipNum[ N_] :=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]]];]]
```

16. Los polinomios de Legendre se pueden calcular mediante las fórmulas:

- $P_0 = 1$
- $P_1 = x$
- $P_n = ((2*n-1)/n) * x * P_{n-1} - ((n-1) / n) * P_{n-2}$

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 $\mathbf{P_n}$ correspondiente, no se ocupe de validar x.

```
i=2;
Legendr[x_, n_]:=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];]]
```

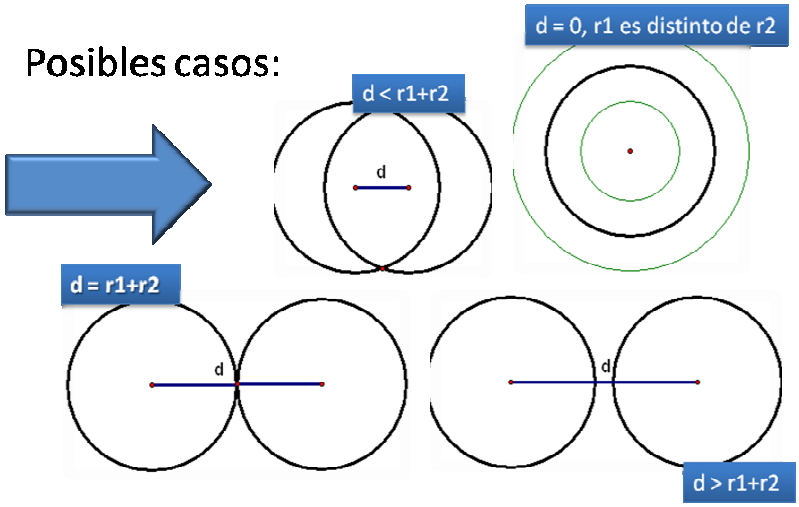
17. Diseñe un método que dadas dos circunferencias (con centros en coordenadas “x”, “y” y su radio) calcule cuántos puntos en común tienen dichas circunferencias (uno, ninguno, dos o infinitos puntos)

Para realizar este programa, se debe calcular la suma de los radios *r*1 y *r*2; y la distancia entre los dos centros (*x*1, *y*1) y (*x*2, *y*2) de las circunferencias utilizando las siguientes fórmulas:

$$Suma\_radios = r_1 + r_2$$

$$Distancia = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Si la distancia es cero y los radios son iguales se trata de la misma circunferencia y todos sus puntos coinciden (infinitos puntos de corte). Si la distancia es cero y los radios no son iguales entonces no existe ningún punto de corte. En otro caso, si la distancia es menor que la suma de los radios, entonces existen dos puntos de corte. Si la distancia es igual a la suma de los radios, entonces sólo existe un punto de corte. Por último, si la distancia es mayor que la suma de los radios, entonces no existe ningún punto de corte. Observe la siguiente figura:




```
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"];];];];
```

5. CONCLUSIONES

Mathematica 5.0 es un software versátil para la enseñanza y el aprendizaje de la programación estructurada, brinda herramientas de programación de fácil uso para introducir a los estudiantes neófitos; en la lógica de la resolución de problemas de forma algorítmica.

7. REFERENCIAS BIBLIOGRÁFICAS

[1]. Alcalde, E. y García M. (1989). Metodología de la programación. Mc. Graw Hill. México.

[2]. Ávila, J. (2008). Crestomatía de temas para EIF200 Fundamentos de Informática. Universidad Nacional.

[3]. Carrillo, A. (2005). Mathematica 5, Aplicaciones para PC. Alfaomega. México.

[4]. Joyanes, L. (1990). Problemas de la metodología de la programación. McGraw Hill. España.

[5]. Korfhage, R. (1970). Lógica y algoritmos. Limusa-Wesley. México.

[6]. Levine G. (2001). Computación y programación moderna. Prentice Hall. Primera Edición.

[7]. Long, L. (1990). Introducción a las computadoras y al procesamiento de información. Prentice Hall. México.

[8]. López, R. (2006). Metodología de la programación orientada a objetos. Alfaomega. México.

[9]. Well, P., Gaylord, R. y Kamin, S. (2005). An Introduction to Programming with Mathematica. Third Edition. CAMBRIDGE UNIVERSITY PRESS.