PRÁCTICA Nº3

Programación en Mathematica

Mathematica no sólo es un paquete de cálculo simbólico en el que se introduce un problema y se obtiene un resultado, sino que también incorpora un lenguaje de programación, que nos permite implementar algoritmos numéricos, algebraicos o gráficos.

En programas escritos en este lenguaje podemos utilizar todos los comandos de Mathematica, lo cual supone un gran enriquecimiento de tal lenguaje.

Como en cualquier otro lenguaje de programación posee órdenes destinadas a los siguientes fines: definición de constantes y variables, asignación de datos, definición de funciones, toma de decisiones, repetición de acciones a modo de bucles, construcción de paquetes de funciones u ordenes, etc.

Mathematica es un programa desarrollado en C; por lo tanto, el propio lenguaje de programación de Mathematica hereda gran parte de la estructura de éste.

1. EXPRESIONES LÓGICAS.

Una expresión lógica es aquella que, al ser evaluada, puede tomar los valores verdadero (TRUE) y falso (FALSE). Las más simples constan de dos miembros unidos por un conectivo lógico:

==	Igualdad
!=	Desigualdad
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

Se puede combinar dos o más expresiones lógicas para lograr otra más compleja mediante los operadores lógicos:

&& ó And[,]	Conjunción "y"

|| u Or[,]	Disyunción "o"
Xor[,]	Disyunción exclusiva
! ó Not[ ]	Negación

2. ORDENES CONDICIONALES.

Son aquellas en las que es posible decidir si se realiza una u otra acción según la veracidad o falsedad de una expresión lógica:

If[condición, proceso1, proceso2]

Si "condición" es verdadera se ejecutarán "proceso1", en caso contrario se ejecutarán "proceso2". La expresión "condición" es de tipo lógico y "proceso1" y "proceso2" son secuencias de ordenes separadas por el carácter ";".

```
a=2;
b=2;
If[(a<3)||(b==4),Print[a];Print[b],Print["falso"]]
```

En los casos en los que se haga necesario controlar el programa no a través de una sola condición, sino de varias, se dispone del comando:

Which[condición1,proceso1,condición2,proceso2,...]

Evalúa "condición1", si es verdadera se ejecuta "proceso1" y finaliza la orden, y si es falsa, evalúa "condición2",... hasta que encuentra una cierta y devuelva el valor de la expresión asociada.

Si ponemos True como última condición, se consigue evaluar la última expresión si ninguna de las condiciones anteriores han resultado ciertas. Se utiliza por ejemplo a la hora de definir funciones a trozos:

```
f[x_]:= Which[0<=x<=1,x^2,1<=x<=2,2-x^2,True,0]
f[0.5]
```

Define una función a trozos dada por x^2 sobre [0, 1], por $2-x^2$ sobre [1, 2] y 0 en el resto de los valores reales.

Ejercicio 1:

Definir la función valor absoluto usando la orden Which.

Solución:

3. BUCLES.

Un bucle es un proceso que se realiza cierto número de veces. En tales procesos aparecen dos tipos de variables:

a) Un contador es una variable que toma valores numéricos distanciados entre sí por una misma cantidad, que llamaremos paso. Al primer valor del contador lo llamaremos inicio. Es usual utilizar las letras i, j, k,... Si i es el nombre de un contador el aumento o disminución se indica mediante:

i=i+paso	**i=i-paso**
i++ (equival. a i=i+1)	**i-- (equival. a i=i-1)**

Las estructuras que definen bucles son:

4.1 El bucle FOR

Es conocido por el lector de otros lenguajes de programación el funcionamiento de los bucles FOR, en Mathematica este bucle se programa mediante la siguiente instrucción:

For[comienzo, test, incremento, cuerpo]

Donde:

a) *comienzo,* marca el inicio del bucle, por ejemplo 'j=1', indicará que el bucle comienza asignando a la variable j el valor 1.
b) *test*, indicará la condición de parada del bucle, es decir, cuando debe de terminar el bucle, por ejemplo 'j<8', hará que el bucle esté funcionando siempre que j sea menor estricto que 8.
c) *incremento,* indicará la variación en cada ciclo del bucle que sufre la variable definida en *comienzo,* por ejemplo, 'j++' hará que j se incremente en una unidad a cada ciclo del bucle.
d) *cuerpo*, en esta parte pondremos todas las instrucciones que queremos que se ejecuten en cada ciclo del bucle.

Ejemplo:

In[1]:=

t=0;
For[j=1,j<8,j++,t++;Print[t]];

Out[1]=

1
2
3
4
5
6
7

Este bucle comienza con j=1 para cuando j sea mayor o igual a 8 e incrementa j una unidad en cada ciclo, luego repetirá la ejecución del cuerpo 7 veces y en cada ciclo suma a t una unidad y muestra en pantalla el valor de t.

4.2 El bucle DO

En Mathematica este bucle se programa mediante la siguiente instrucción:

Do[cuerpo,{contador,inicio,fin,incremento}]

Donde:

a) *cuerpo,* en esta parte pondremos todas las instrucciones que queremos que se ejecuten en cada ciclo del bucle.
b) *contador,* indica el nombre que se le asigna al contador.
c) *inicio,* valor de inicio del bucle.
d) *fin*, indicará el valor de parada del bucle, es decir, cuando debe de terminar el bucle.
e) *incremento,* indicará la variación en cada ciclo del bucle que sufre la variable definida en *comienzo*.

Por ejemplo, {j, 1, 7,1}, indica que el bucle comienza asignando a la variable j el valor 1, hará que esté funcionando hasta que j tome el valor 7 y que j se incremente en una unidad a cada ciclo del bucle.

Ejemplo:

In[1]:=

Do[Print[j^2],{j,1,7,2}]

Out[1]=

1
9
25
49

Este bucle comienza con j=1 para cuando j sea 7 e incrementa j en dos unidades en cada ciclo, por tanto dará 4 ciclos y en cada uno de ello calcula e imprime el valor de j^2.

4.3 El bucle WHILE

Es conocido por el lector de otros lenguajes de programación el funcionamiento de los bucles WHILE, en Mathematica este bucle se programa mediante la siguiente instrucción:

While[test, cuerpo]

Donde:

a) *test*, indicará la condición de parada del bucle, es decir, cuando debe de terminar el bucle, por ejemplo 'PrimeQ[n]==False' hará que el bucle repita la ejecución del cuerpo mientras que n no sea primo.
b) *cuerpo*, en esta parte pondremos todas las instrucciones que queremos que se ejecuten en cada ciclo del bucle.

Ejemplo:

In[1]:=

```
n=500;
While[n<1000,n=n+50;Print[n]]
```

Out[1]=

```
550
600
650
700
750
800
850
900
950
1000
```

Este bucle comienza con n=500 para cuando n sea mayor o igual a 1000, el bucle parará, mientras tanto ejecuta el cuerpo reiteradas veces, como n se incrementa 50 unidades en cada ciclo, el bucle no parará hasta que n alcance 1000.

La orden **Break[]** que no lleva argumento, es útil en un proceso definido por For, Do, While, para interrumpir la acción del mismo. El uso de ésta necesita de una estructura If entre los comandos que componen el proceso. Por ejemplo

```
For[i=30, i<=40,i++,If[i^2>1500,Break[],Print[i^2]]]

Do[If[i^2>1500,Break[],Print[i^2]],{i,30,40}]

i=30;
While[i<=40,If[i^2>1500,Break[],Print[i^2]];i=i+1]
```

Escribe el cuadrado de todos los números comprendidos entre 30 y 40 siempre que éste no supere el valor de 1500.

b) Otro tipo de variables que aparecen en los bucles son los acumuladores, variables que, a partir de un valor inicial, mediante un proceso repetitivo, va modificando su valor en función de otra variable (contador):

De tipo aditivo:

acumulador=acumulador+expresión

Los conmutadores de las órdenes For y Do cuando crecen, es decir, con paso positivo, son un ejemplo (su valor inicial corresponde al valor inicio). Para muchos contadores aditivos el valor inicial normalmente es cero. Por ejemplo, la suma de los 100 primeros números pares pueden programarse utilizando un acumulador aditivo en la forma siguiente:

```
s=0;
Do[s=s+2*i,{i,1,100}]
Print[s"es la suma de los 100 primeros números pares"]
```

De tipo multiplicativo:

acumulador=acumulador*expresión

y su valor inicial, usualmente, es 1. Por ejemplo, el producto de los cuadrados de los 10 primeros números impares se puede calcular mediante un pequeño programa como el siguiente:

```
p=1;
Do[p=p*(2n-1)^2,{n,1,10}]
Print[p"es el producto de los 10 primeros números impares"]
```

Ejercicio 2:

a) Calcular usando los distintos bucles la suma de los números menores que 100 y multiplos de 7.
b) Calcular usando los distintos bucles el producto de los números impares comprendidos entre 10 y 20.

Solución: