Conceptos básicos de programación

Variables y Funciones

La asignación de datos a variables se lleva a cabo en el entorno de trabajo de *Mathematica* de una manera natural, utilizando el operador de igualdad. Observe y ejecute las siguientes asignaciones de datos.

```
a = Cos[π]
```

```
a = 3 Sin[π/2] - 2;
```

```
a
```

```
a = 7 3 + 5
```

```
a = Log[2]
```

```
a = 4 - b
```

```
a = 3 < 1
```

```
a = "Esto es un texto"
```

```
a = Plot[Sin[x], {x, -π, π}];
```

Habrá observado cómo la variable **a** va siendo modificada a través de la asignación de contenidos que van cambiando, que pueden ser números, expresiones alfanuméricas, valores lógicos o gráficos. ¿Cuál es el contenido actual de la variable **a**?Tal y como se ha ido definiendo, será el último.

```
a
```

La salida obtenida indica que a la variable **a** se le ha asociado una gráfica, que podrá verse utilizando la orden ya conocida, **Show**.

```
Show[a]
```

Pero hay otra forma de asignar datos. Observe los siguientes conjuntos de órdenes de asignación.

```
x = 2;
a = x;
x = 3;
a
```

```
x = 2;
a := x;
x = 3;
a
```

No se obtiene el mismo valor para la variable **a**. En el primer caso, en una de los pasos se definió **a** como **x**, por lo que a la variable **a** se le asignó el contenido que en ese momento tenía **x**, es decir, 2. Posteriormente se modifica la variable **x**, pero la variable a no resulta modificada.

En el segundo grupo de órdenes, la variable **a** se define mediante asignación diferida, lo que se nota en que no se emplea el operador de igualdad sino := El efecto es que la modificación de la variable mediante la que se define produce la modificación automática de **a**.

La definición de funciones es otra tarea elemental, que requiere la especificación de un nombre para la función, otro para la variable, que debe ser especificada entre corchetes y no entre paréntesis, y la utilización del operador de asignación. A continuación se muestra cómo se define una función con *Mathematica*.

```
f[x_] := x^3
```

Se evalúa de la manera natural.

```
f[2]
```

Podemos redefinirla.

```
f[x_] := Cos[x]
```

Y evaluar la nueva función resultante.

```
f[π]
```

Hay que tener cuidado a la hora de definir funciones. Observe las salidas del siguiente ejemplo:

```
x = 3
f[x_] := x^2
f[x]
f′[x]
```

La función f se ha definido en base al valor que en ese momento tiene la variable x, lo que afecta a su expresión y a la de su derivada. Para que no suceda esto hay que limpiar la variable **x**, lo que se consigue con la orden **Clear**. Vea el resultado.

```
Clear[x]
f[x]
f′[x]
```

Análogamente se definen y evalúan funciones de varias varibles como se muestra a continuación:

```
f[x_, y_] := x^2 + y^2
```

```
f[2, 3]
```

Expresiones lógicas. Órdenes condicionales.

Las siguientes son expresiones lógicas construidas con los conectivos y operadores lógicos que *Mathematica* manipula. Los conectivos lógicos son:

==	igualdad
! =	desigualdad
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que

y los operadores lógicos son:

&& o And[,]	conjunción " y " ,
|| u Or[,]	disyunción " o " ,
Xor[,]	disyunción exclusiva ,
! o Not[ ]	negación.

```
a = 3
```

```
a == 5
```

```
a ≠ 5
```

```
a < 5
```

```
a ≤ 5
```

```
a > 5
```

```
a ≥ 5
```

```
b = 7
```

```
a ≤ 5 && b ≥ 8
```

```
a ≤ 5 || b ≥ 8
```

```
Xor[a ≤ 5, b ≥ 5]
```

Observemos ahora cómo actúan las órdenes condicionales

If[*condición*, *proceso1*, *proceso2*]

y

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

Para ello, ejecute los siguientes bloques de órdenes:

```
a = 2;
b = 3;
If[a < 3 && b == 4,
        Print[a]; Print[b],
     Print["Falso"]]
```

```
If[a ≤ 4 || b ≥ 5,
      Print["a=", a]; Print["b=", b],
      Print["Falso"]]
```

Compruebe que la orden **If** actúa del siguiente modo: si la condición es cierta se realiza el *proceso1* y, en caso contrario, se realiza el *proceso2.* Asimismo, ejecute y observe este otro grupo de órdenes:

```
f[x_] := Which[0 ≤ x ≤ 1, x^2, 1 ≤ x ≤ 2, 2 - x^2]

f[0.5]

f[1.5]

f[3]
```

En este último caso no se ha generado salida ya que f[x] no está definida para x=3. Pero la siguiente función está definida en todo el conjunto de los números reales.

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

f[3]
```

Bucles

Las órdenes más importantes que definen bucles, o procesos repetitivos, son **For**, **Do** y **While**, cuyas estructuras son del tipo:

For[*contador=inicio, condición, paso, proceso*]

Do[*proceso*, {*contador*, *inicio*, *fin*, *paso*}]

While[*condición*, *proceso*]

Si no se especifica en **Do** el valor de *inicio*, se considera que es 1; igual ocurre si no es indica el valor del paso.

Como ejemplo en el que se muestre la diferencia de estructuras, nos planteamos el problema de escribir los 9 primeros números naturales y sus cubos mediante cada una de estas sentencias. La orden que produce la salida indicada es **Print**. Entre corchetes se especifica el contenido que se quiere mostrar.

```
For[i = 1, i ≤ 9, i++, Print[i, "   ", i^3]]

Do[Print[i, "    ", i^3], {i, 9}]

i = 1;
While[i ≤ 9, Print[i, "    ", i^3]; i = i + 1]
```

En el siguiente siguiente ejemplo observe cómo actúan las variables **i**, llamada contador, y **s,** llamada acumulador. Se trata del cálculo de la suma de los 20 primeros números naturales pares, del 2 al 40.

```
s = 0;
Do[s = s + 2 i, {i, 1, 20}]
s
```

Iteradores, Sum y Product

La anterior suma se podría haber calculado directamente utilizando la orden

Sum[*expresión*, {*contador*, *inicio*, *fin*, *paso*}]

que en nuestro caso se traduce en

$$\sum_{i=1}^{20} 2\,i$$

Si en lugar de la suma se deseara el producto se utilizaría la sentencia

Product[*expresión*, {*contador*, *inicio*, *fin*, *paso*}]

es decir, el producto de los 20 primeros números naturales pares es

$$\prod_{i=1}^{20} 2\,i$$

Análogamente, la suma y el producto de los cuadrados de los múltiplos de 3 entre 6 y 27 es

$$\sum_{\substack{i=6 \\ \Delta i=3}}^{27} i^2$$

$$\prod_{\substack{i=6 \\ \Delta i=3}}^{17} i^2$$

Listas

Una lista es un conjunto de datos cualesquiera. En los siguientes ejemplos se asignan listas a variables, se efectúan operaciones con listas y se evalúan funciones en listas.

```
a = {1, 2, 3}
```

a^2

```
b = {-1, 0, 1}
a + b
Sin[a]
N[%]
```

E^b

La orden

Length[*lista*]

calcula el número de componentes de la lista.

```
Length[a]
```

Una lista puede tener como componentes otras listas. Por ejemplo,

```
a = {{1, 2, 3}, {-1, -2, -3}, {2, 3, 1}}
```

Se puede operar sobre ella.

```
a^2
```

En el caso en el que todas las sublistas de una lista sean de la misma longitud, se puede considerar la lista como una tabla en la que cada fila está formada por las componentes de cada sublista y puede verse en esta forma con las órdenes

TableForm[*a*]

o

MatrixForm[*a*]

```
TableForm[a]
```

La orden **Table** genera una lista de elementos. Observe su estructura en los siguientes ejemplos:

```
Table[i, {i, 1, 3}]

Table[i j, {i, -1, 2}, {j, 2}]

Table[i + j, {i, 1, 3}, {j, 2, 6, 2}]
```

La orden

Dimensions[*lista*]

genera una lista con las dimensiones de la lista dada y las sublistas que la componen, en el caso de que existan.

```
a = {-1, 3, 5};
Dimensions[a]

a = {{1, 2}, {3, 5}, {-1, 1}};
Dimensions[a]
```

Un elemento de una lista se indica con el nombre de la lista y la posición que ocupa indicada entre dobles corchetes. Por ejemplo el segundo elemento de la lista a se escribe

```
a[[2]]

b[[1]]
```

Si los elementos de una lista son sublistas, la expresión

a[[i,j]]

indica la componente j-ésima de la sublista i-ésima.

```
a[[2, 2]]
a[[3, 1]]
```

Por último, si se desea añadir una nueva componente a una lista se escribirá la orden

AppendTo[*lista*, *elemento*]

y el elemento indicado será la última componente de la lista. Observe esta orden en los siguientes ejemplos:

```
m = Table[i j, {j, 3}, {i, 1, 7, 2}]
v = Table[i^3, {i, -2, 8, 3}]
AppendTo[m, v]
AppendTo[v, 1000]
```

Ejercicios

1.- Calcule, usando la orden **Do**, la suma de los números comprendidos entre 1890 y ds que sean múltiplos de 5. Resuelva el mismo problema usando las órdenes **For** y **Sum**.

2. -Tome las listas m={d1,d2,d3,d4,d5,d6,d7,d8} y n={d2,d4,d6,d8,d1,d3,d5,d7}. Calcule su producto escalar por medio de la sentencia **For**.

3.- Sea v un vector de n componentes. Escriba el vector cuyas componentes son los dígitos de su D.N.I. y calcule la suma de los cuadrados de sus componentes.

4.- Use las órdenes **Do** e **If** para programar el siguiente problema: se desea calcular el producto de las componentes del vector del problema anterior que no sean nulas. Compruebe que se obtiene el mismo resultado cuando en el vector v se suprimen las componentes nulas y se calcula el producto de las restantes con la orden **Product**. Aplíquelo todo de nuevo al vector v1={1,0,1,0,1,0,dm,0}.