Conceptos básicos de programación en Mathematica

Práctica 2 (Común para las asignaturas de Álgebra Lineal y Cálculo Matemático en E.U.A.T.)

Variables y Funciones

```
Clear["Global`*"];
```

■ Variables

■ Ejemplo 1

Observa la diferencia entre las dos formas de asignación de datos a variables en los dos siguientes bloques de órdenes:

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

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

En la primera asignación, **a=x**, el valor de **a** se asigna inmediatamente, es decir, el valor que se asigna a la variable **a** es el valor que tiene **x** en ese momento. Sin embargo en la asignación **a:=x** el proceso es diferido, es decir, tras la asignación, cada vez que aparece la variable **a**, se evalua lo que aparece a la derecha de "**:=**" (en este caso **x**) y se le asigna a la variable **a**; por tanto, el valor de **a** cambia cada vez que cambia el de **x**.

■ Ejemplo 2

Las asignaciones permiten modificar "sobre la marcha" una variable

```
b = 1;
x = b;
x
```

```
x = x + 1;
x

b = x;
x
b
```

Observa que el símbolo "=" no es el "igual" que se usa habitualmente (la segunda celda no tendría sentido en general). Más adelante volveremos sobre este "igual".

■ Ejercicio 1

a) Asígnale a la variable **q** el valor 37.
b) Calcula **q^2-q+1**, el logaritmo neperiano de **q^3 - 3** y e elevado al coseno de **q^4-2**.

■ Ejercicio 2

¿Qué valor tendrá la variable **c** en la primera celda? ¿Y en la segunda? ¿Cuál es la diferencia?

```
y = 3;
y = y – 2;
c := y;
y = 2 * c;
c

Clear[y, c];
y = 3;
y = y – 2;
c = y;
y = 2 * c;
c
```

■ Funciones en una variable

Veamos cómo se define una función con *Mathematica* mediante dos ejemplos.

```
f[x_] := x^3
f[2]

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

■ Ejercicio 3

1) Define la función g(x)=tg(x/4).
2) Calcula g(π).

■ Observaciones

1) Hay que tener en cuenta el uso de "**_**" después de la variable independiente **x** y de "**:=**". ¿Qué pasa si no se utilizan?

```
g[x] := x - 3;
g[6]

h[x_] = x + 1;
x = 2;
h[-1]

h[x_] := x + 1;
x = 2;
h[-1]
```

2) Para que la definición sea correcta, el símbolo **x** debe ser una variable, es decir, no debe tener asignado valor alguno antes de definir la función. Observa la salidas del siguiente ejemplo.

```
x = 3
f[x_] := x^2
f[x]
f'[x] (*f' indica la función derivada de f*)
```

Y ahora las de éste.

```
Clear[x]
f[x]
f'[x]
```

¿Cuáles son las diferencias?

■ Funciones en varias variables

Se definen y evaluan de manera análoga al caso de una variable.

```
f[x_, y_] := x^2 + y^2
f[2, 3]
```

Expresiones lógicas

Mathematica permite trabajar con expresiones lógicas, las cuales se construyen a partir de los conectivos y operadores lógicos que este programa manipula.

Los conectivos lógicos son

==	igualdad
!=	desigualdad

<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que

Los operadores lógicos son

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

Así, si **p** y **q** son dos expresiones lógicas que pueden tomar el valor True (verdadero) o False (falso), se tiene que

- El operador **p && q** (o **And[p,q]**) es True si los dos argumentos (**p** y **q**) son True; es False si cualquiera de los argumentos es False.

- El operador **p || q** (u **Or[p,q]**) es True si cualquiera de los argumentos es True; es False si los dos argumentos son False.

- El operador **Xor[p,q]** es True si uno de los dos argumentos es True; es False si los dos argumentos son True o los dos son False.

- El operador **!p** (o **Not[p]**) es False si **p** es True; es True si **p** es False.

La siguiente instrucción (que podrás entender al final de la práctica) muestra en pantalla como actúan los operadores que hemos visto.

```
TablaVerdad =
 TableForm[ { {"p", "q", "And[p,q]", "Or[p,q]", "Xor[p,q]", "Not[p]"},
   {True, True, And[True, True], Or[True, True], Xor[True, True], Not[True]},
   {True, False, And[True, False], Or[True, False], Xor[True, False]},
   {False, True, And[False, True],
    Or[False, True], Xor[False, True], Not[False]},
   {False, False, And[False, False], Or[False, False], Xor[False, False]} } ]
```

■ Ejemplo 3

Para ensayar con expresiones lógicas, demos a la variable **a** el valor 3.

```
a=3
```

Comprobemos el valor de varias expresiones lógicas.

1) ¿Es la variable **a** igual a 5? (Observa los dos iguales)

```
a==5
```

2) ¿Es **a** distinta de 5?

```
a != 5
```

3) ¿Es **a** menor que 5?

```
a < 5
```

4) ¿Es **a** menor o igual que 5?

```
a <= 5
```

5) ¿Es **a** mayor que 5?

```
a > 5
```

6) ¿Es **a** mayor o igual que 5?

```
a >= 5
```

Asignamos ahora el valor 7 a la variable **b**.

```
b = 7
```

7) ¿Es **a** mayor o igual que 5 y **b** mayor o igual que 8?

```
a <= 5 && b >= 8
```

8) ¿Es **a** menor o igual que 5 o **b** mayor o igual que 8?

```
a <= 5 || b >= 8
```

9) ¿O bien es **a** menor o igual que 5 o bien es **b** mayor o igual que 5?

```
Xor[a <= 5, b >= 5]
```

10) ¿Qué es lo contrario de "¿Es **a** menor o igual que 5?" ?

```
Not[a <= 5]
```

■ Ejercicio 4

Dale a la variable **a** el valor de la primera cifra de tu DNI o pasaporte, a la variable **b** el valor de la segunda cifra y a la variable **c** el valor de la tercera cifra. Comprueba si **b^2-4ac** es mayor que cero, si es menor que cero o si es igual que cero.

Órdenes condicionales

Las órdenes condiconales emplean expresiones lógicas y permiten que se efectúe un proceso u otro según se verifique cierta condición. Tales órdenes son:

If[condición, proceso1, proceso2]

y

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

■ If

La orden **If** actúa del siguiente modo: si la **condición** es cierta se realiza el **proceso1** y, en caso contrario, se realiza el **proceso2**. Comprueba esto ejecutando las dos siguientes celdas. Observe que un proceso puede constar de varias órdenes siempre que se separen con punto y coma ;)

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

If[(a <= 4) || (b >= 5), Print["a=", a]; Print["b=", b], Print["Falso"]]
```

Observa que un proceso puede constar de varias órdenes siempre que éstas se separen con un punto y coma (**;**).

■ Ejercicio 5

Define los valores **a**, **b** y **c** como en el Ejercicio 4. Escribe una instrucción que imprima el texto "Hay al menos una solución" si **b^2-4ac** es mayor o igual que cero, y que imprima "No hay ninguna solución" en caso contrario.

■ Which

Para ver el funcionamiento de la orden Which, ejecuta y observa las dos siguientes celdas.

```
f[x_] := Which[0 <= x <= 1, x^2, 1 <= x <= 2, 2 - x^2]
f[0.5]
f[1.5]

f[3]
```

En la segunda celda no se ha generado salida ya que **f[x]** no está definida para **x=3**. La siguiente función sí está definida en todo R.

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

f[3]
```

¿Cuál es la función de "True" como quinto argumento?

■ Ejercicio 6

Define la función valor absoluto como función definida a trozos.

Bucles

Las órdenes más importantes que definen bucles, o procesos iterativos (es decir, repetitivos), son **Do**, **For** y **While**:

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

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

While[condición, proceso]

La orden **Do** realiza el **proceso** para cada uno de los valores que toma la variable **contador** desde el valor **inicio** hasta el valor **fin**. Cada vez que se ejecuta el **proceso**, la variable **contador** se incrementa en una cantidad determinada por el **paso**.

La orden **For** realiza el **proceso** para cada uno de los valores que toma la variable **contador** desde el valor **inicio** y mientras **condición** sea cierta. Nuevamente cada vez que se ejecuta el **proceso**, la variable **contador** se incrementa en una cantidad determinada por el **paso**.

La orden **While** ejecuta el proceso indicado mientras **condición** sea cierta.

Como ejemplo en el que se muestra la diferencia de estas órdenes, nos planteamos el problema de escribir los 9 primeros números naturales y sus cubos mediante cada una de estas sentencias.

```
Do[Print[i, "  ", i^3], {i, 1, 9}]

For[i = 1, i < 10, i = i + 1, Print[i, "  ", i^3]]

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

■ Ejemplo 4

En este ejemplo observa cómo actuan las variables **i** (**contador**) y **s** (**acumulador**). Se calcula la suma de los 20 primeros números naturales pares, del 2 al 40.

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

■ Observación: ¡MUY IMPORTANTE!

El bucle **Do** es un proceso que termina siempre ya que tiene un **contador** con **inicio** y **final** bien determinados. Por contra los bucles **For** y **While** son más peligroso si no se controlan bien ya que su finalización depende de una **condición** que ha de estar perfectamente definida. La siguiente celda es un ejemplo de esto. ¡Es recomendable (muy recomendable) que no la ejecutes!

```
i = 1;
While[i >= 1,
  Print["¡Cuidado con los bucles infinitooooooooooos!!!!!"]; i = i + 1];
```

■ **Ejercicio 7**

a) Calcula la suma de los 20 primeros números naturales usando **Do**, **For** y **While**.
b) Calcula la multiplicación **1*3*5*...*1001** usando **Do**, **For** y **While**.

Bucles predefinidos: iteradores Sum y Product

La suma propuesta en el Ejercicio 7a se puede calcular directamente por medio de la orden **Sum**,

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

Para el ejercicio propuesto es

Sum[2*i, {i, 1, 20, 1}]

Comprébalo en la celda siguiente.

```
Sum[2 * i, {i, 20}]
```

Si en lugar de una suma se desea calcular un producto, la orden es **Product**,

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

Así el producto de los 20 primeros números naturales pares es

Product[2*i, {i, 1, 20, 1}]

```
Product[2 * i, {i, 20}]
```

■ **Ejemplo 5**

La suma y el producto de los cuadrados de los múltiplos de 3 entre 6 y 27 es:

```
Sum[i^2, {i, 6, 27, 3}]
```

```
Product[i^2, {i, 6, 17, 3}]
```

■ **Ejercicio 8**

a) Calcula la suma de los 20 primeros números naturales usando **Sum**.
b) Calcula la multiplicación **1*3*5*...*1001** usando **Product**.

Listas

Una lista es un conjunto de datos cualesquiera. Los siguientes ejemplos asignan listas a variables, efectúan operaciones con listas o evalúan funciones en listas.

■ Ejemplo 6

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

b = {-1, 0, 1}
a + b

Sin[a]
% // N

E^b
```

■ Ejercicio 9

Define una lista, con el nombre **lista0**, que contenga los elementos 0, 1, 2 y 3.

■ Length[lista]

La orden **Length[lista]** calcula el número de componentes de la lista.

```
Length[a]
```

■ Listas de listas

Una lista puede tener como componentes otras listas.

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

■ TableForm[lista]

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 visualizarse en esta forma con la orden **TableForm[.]**.

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

Estas listas de listas se emplearán más adelante para definir matrices. En una matriz cada una de las sublistas se corresponde con una fila de dicha matriz.

■ Table[lista]

La orden **Table** genera una lista de elementos. Observa 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}]
```

■ Dimensions[lista]

La orden **Dimensions[lista]** genera una lista con las dimensiones de la lista dada y, si existen, de las sublistas que la componen.

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

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

■ Componentes (elementos) particulares de una lista

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 una lista **a** se escribe **a[[2]]** y el primer elemento de una lista **b** se escribe **b[[1]]**.

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

b = {-1, 0, 1};
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 = {{1, 2}, {3, 5}, {-1, 1}};
a[[2, 2]]
a[[3, 1]]
```

■ Observación: ¡MUY IMPORTANTE!

Es habitual olvidar los dobles corchetes y poner tan sólo uno al iniciar y otro al terminar. Este no es el tratamiento correcto.

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

■ Cómo añadir componentes a una lista

Si queremos añadir una nueva componente a una lista utilizaremos la orden **AppendTo[lista, elemento]** (añade la componente al final de la lista) o la orden PrependTo[lista, elemento] (añade la componente al principio de la lista). Veamos algunos 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]

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

PrependTo[m, v]

PrependTo[v, 0]
```

■ Ejercicio 10

Añade al final de la lista **lista0** (del Ejercicio 9) el elemento 7 tanto al principio como al final.

■ Asignación de valores a las componentes a una lista

Para asignar el valor **x** a la componente i-ésima del vector **a** (esto es, **a** es una lista formada sólo por números) se usa la orden **a[[i]] = x**. Por ejemplo, definamos una lista **a** con tres componentes nulas y luego demos distintos valores a las componentes de **a**.

```
a = Table[0, {i, 1, 3}]
a[[1]] = -1
a[[2]] = 2
a[[3]] = 100
a
```

■ Ejercicio 11

a) Genera una lista con diez elementos. Usando una orden **Do**, sustituye el valor del elemento i-ésimo por **2^i**.

b) Genere la matriz de orden 3x3 $\begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{pmatrix}$ usando la asignación directa **Table** y algún bucle (**Do**, **For** o **While**).

Ejercicios

1.- Usando la orden **Do**, muestra por pantalla los múltiplos de 7 comprendidos entre 33 y 128. Ten en cuenta que ni 33 ni 128 son múltiplos de 7.

2.- Calcula, usando la orden **Do**, la suma de los números comprendidos entre 1890 y 18 que sean múltiplos de 5. Resolver el mismo problema usando la orden Sum. Ten en cuenta que 18 no es múltiplo de 5.

3.- Genera y guarda en la variable **pol** una lista de 10 polinomios tales que el que ocupa la posición i-ésima sea **x^i+7x+5**. Efectúa la suma y el producto de los 10 polinomios con las órdenes **Sum** y **Product**.

4.- Sea **v** un vector de **n** componentes. Definiremos la longitud de **v** como la norma euclídea del mismo, esto es, como la raíz cuadrada de la suma de los cuadrados de sus componentes. Escribe el vector cuyas componentes son los dígitos de tu DNI y calcula su longitud.

5.- Define el vector **v** que tiene por componentes los dígitos de tu DNI. Usa un bucle para crear un vector **w** que contenga las mismas cifras que **v** en orden inverso, es decir, que si **n** es el número de componentes de la lista, se tiene que **w[[1]]=v[[n]]**, **w[[2]]=v[[n-1]]** y así hasta llegar a **w[[n]]=v[[1]]**.