# Práctica 2 (común)

## Conceptos básicos de programación

---

## Variables y Funciones

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

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

```
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 tenía **x** en ese momento. Sin embargo en la asignación **a:=x** se asigna la variable de forma diferida, es decir, tras la asignación, cada vez que aparece la variable **a**, se evalúa lo que aparece a la derecha de **:=** (en este caso **x**) y se le asigna a la variable **a** , con lo que el valor de **a** cambia cada vez que cambia el de **x**.

**Ejercicio:** Asigne a la variable **q** el valor 37, y luego calcule q^2-q+1.

A continuación se muestra cómo se define una función con Mathematica mediante algunos ejemplos:

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

```
f[2]
```

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

```
f[Pi]
```

**Ejercicio:** Defina la función g(x)=x^2+2. Calcule g(2).

Para que la definición sea correcta, el símbolo x debe ser una variable y por tanto no debe tener asignado valor alguno. Obsérvense la salidas del siguiente ejemplo:

```
x=3
```

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

```
f[x]
```

```
f'[x]
```

y ahora las de este otro:

```
Clear[x]
```

```
f[x]
```

```
f'[x]
```

¿cuál es la diferencia?

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. |

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, y False si cualquiera de sus argumentos es falso.
- El operador p || q u Or[p,q] es True si cualquiera de sus argumentos es True, y es False si sus dos argumentos son False.
- El operador Xor[p,q] es True si uno de los dos argumentos es True, y devuelve False si los dos argumentos son True o los dos argumentos son False.
- El operador !p o Not[p] devuelve False si p es True, y devuelve True si p es False.

La siguiente instrucción (que podrá entender al final de la práctica) muestra en pantalla cómo 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]}
  } ]
```

Para ensayar estas expresiones lógicas, vamos a darle a la variable **a** el valor 3.

```
a=3
```

Y vamos a comprobar el valor de las siguientes expresiones lógicas.

Respondemos a la pregunta ¿**a** es igual a 5? (Observe los dos iguales)

```
a==5
```

¿**a** es distinto a 5?

```
a!=5
```

¿**a** es menor que 5?

```
a<5
```

¿**a** es menor o igual que 5?

```
a<=5
```

¿**a** es mayor que 5?

```
a>5
```

¿**a** es mayor o igual que 5?

```
a>=5
```

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

```
b=7
```

¿**a** es menor o igual que 5, y **b** mayor o igual que 8?

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

¿**a** es menor o igual que 5, o **b** mayor o igual que 8?

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

¿O bien **a** es menor o igual que 5, o bien **b** es mayor o igual que 5 (pero no ambos)?

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

Lo contrario de ¿**a** es menor o igual que 5?

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

**Ejercicio:** Dele a la variable **a** el valor de la primera cifra de su DNI o pasaporte, a la variable **b** el valor de la segunda cifra, y a la variable **c** el valor de la tercera cifra. Compruebe si b^2-4ac es mayor que cero, si es menor que cero o si es igual que cero.

Vemos ahora órdenes que emplean expresiones lógicas, y que permiten que se efectúe un proceso u otro según se verifique cierta condición. Estas órdenes son:

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 en las órdenes que acaba de ejecutar, 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. (Observe que un proceso puede constar de varias órdenes siempre que se separen con punto y coma ; )

**Ejercicio:** Defina los valores a, b y c como en el ejercicio anterior. Escriba 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.

Para ver el funcionamiento de la orden Which, 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 R.

```
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 Do y While, cuyas estructuras son del tipo:

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

While[ condición, proceso]

La orden Do realiza el proceso proceso, para cada uno de los valores que tome la variable contador, comenzando la variable por el valor inicio, hasta el valor fin. Cada vez que se ejecuta el proceso proceso, la variable se incrementa cuanto diga paso. Si el valor de paso se omite, se entiende que el paso es uno, es decir la variable se incrementa de uno en uno.

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

Como ejemplo en el que se muestre 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}]

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

En el siguiente siguiente ejemplo observe cómo actuan las variables i, llamada contador, y s, llamada 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
```

# 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 por ejemplo en:

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

es decir

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

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

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

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

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

o sea

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

Análogamente, 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:** Calcule la suma de los 20 primeros números naturales (1+2+...+19+20).

# 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.

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

```
a^2
```

```
b={-1,0,1}
```

```
a+b
```

```
Sin[a]
```

```
%//N
```

```
E^b
```

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

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}}
```

```
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 visualizarse en esta forma con la orden

TableForm[a]

```
TableForm[a]
```

Estas listas de listas se emplearán más adelante para definir matrices, donde cada una de las sublistas es una fila de la matriz.

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]]
```

Y el primer elemento de **b** se escribe:

```
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]

que añade el elemento como última componente a 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]
```

**Ejercicio:** Añada al final de la lista **lista0** el elemento 7.

Para asignar el valor **x** a la componente i-ésima del vector a, se usa la orden

a[[ i ]] = x

Por ejemplo, vamos a generar una lista **a** con tres componentes nulas, y luego damos distintos valores a las componentes de **a**.

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

**Ejercicio:** Genere una lista con diez elementos. Usando una orden Do, sustituya el valor del elemento i-ésimo por 2^ i.

---

# Ejercicios

Para hacer los ejercicios, defina d1 con el valor de la primera cifra de su D.N.I., d2 con el valor de la segunda cifra, y ds la suma de todas las cifras.

1.- Usando la orden Do, muestre por pantalla, los múltiplos de 7 comprendidos entre 33 y 128. (Observe que ni 33 ni 128 son múltiplos de 7.)

2.- Calcule, usando la orden Do, la suma de los números comprendidos entre ds y 1890 que sean múltiplos de 5. Resolver el mismo problema usando la orden Sum. (Observe que 1890 es múltiplo de 5, pero que ds puede que no lo sea.)

3.- Genere, y guarde en la variable pol, una lista de 10 polinomios tales que el que ocupa la posición i-ésima sea (x^i)+(d1 x)+d2. Efectúe su suma y su producto con las órdenes Sum y Product.

4.- Sea v un vector de n componentes. Se define la longitud de v como la norma euclídea del mismo, es decir, la raíz cuadrada de la suma de los cuadrados de sus componentes. Escriba el vector cuyas componentes son los dígitos de su D.N.I. y calcule su longitud.

5.- Defina el vector **v** que tiene por componentes las cifras de su DNI. Use 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 w[[n]]=v[[1]].