Breve Introducción a *Mathematica*

El entorno escogido para las prácticas es *Mathematica*. Este software permite la manipulación sencilla de los conceptos abstractos que se expondrán en clase de teoría. En ese sentido, el interés principal de las prácticas no estriba en conseguir implementaciones eficientes, sino en utilizar un lenguaje que facilite, en el tiempo limitado del que se dispone, la construcción de estos objetos abstractos y el manejo de los mismos.

Teniendo en cuenta todos estos aspectos, a continuación se exponen brevemente las características de *Mathematica* imprescindibles para la realización de las prácticas. Todas las expresiones ejemplo son susceptibles de ser modificadas para que el alumno pueda comprobar las diferencias de comportamiento de los comandos y estructuras presentadas.

Este fichero *Mathematica* permite el despliegue de las distintas secciones haciendo click sobre los triangulos

Sintaxis

Características generales de *Mathematica*

Antes de empezar es importante tener en cuenta lo siguiente:

- **Mathematica diferencia entre mayúsculas y minúsculas**
- *Mathematica* es un entorno semi interpretado, esto significa que las expresiones han de evaluarse después de ser introducidas. **La evaluación de una expresión se realiza por la combinación shift+intro**. Por ejemplo, tras la evaluación de la siguiente expresión:

```
In[6]:= numero = 23;

In[5]:= lista = {0, 1, 2, 3, 4}
Out[5]= {0, 1, 2, 3, 4}
```

  después de la evaluación de estas expresiones, el identificador **numero** toma el valor 23 y al identificador **lista** se le asigna {0,1,2,3,4}.
- Las expresiones se van numerando de manera automática y se disponen en celdas. Cada expresión lleva asociada una **celda de entrada** (In[..]) y otra de **salida** (Out[..]) que es resultado de su evaluación.
- Una de las estructuras de datos más importantes en *Mathematica* (sin duda la que nosotros emplearemos más) es la **lista**. Una lista es toda secuencia de elementos separados por comas y encerrados entre llaves, por ejemplo:

In[7]:=
```
unalistamas = {a, b, 10, {7.5, "b", b}, {a}}
```
Out[7]=
```
{a, b, 10, {7.5, b, b}, {a}}
```

Como puede verse, **los elementos de una determinada lista no han de ser homogéneos**. El cuarto y quinto elementos elementos de la lista anterior son a su vez listas y se mezclan distintos tipos de valores e identificadores.

- Como se ha comentado, en la mayoría de los casos una expresión de entrada genera una expresión de salida después de evaluarse. Un punto y coma al final de una expresión evita que se muestre la expresión de salida. Por ejemplo:

In[8]:=
```
lista = {0, 2, 4, 6, 8};
```

la ausencia de salida no implica que la asignación no se haya realizado.

- Una expresión no tiene efecto alguno si no ha sido evaluada. Como ejemplo, cualquier uso de un identificador previo a la asignación de un valor devuelve exclusivamente el nombre del identificador. Como ejemplo, evalua las siguientes expresiones:

In[9]:=
```
otralista
```
Out[9]=
```
otralista
```

In[10]:=
```
otralista = {a, b, b, a}
```
Out[10]=
```
{a, b, b, a}
```

In[11]:=
```
otralista
```
Out[11]=
```
{a, b, b, a}
```

- El resultado de las sucesivas evaluaciones se almacena en el ***kernel*** de *Mathematica.* A menudo, después de la evaluación de una expresión incorrecta, el kernel almacena un valor erroneo. El 'borrado' de esta asignación puede realizarse mediante el comando:

In[12]:=
```
Clear[otralista]
```
borra

Si se ha perdido el rastro del error, es posible 'anular' el Kernel, olvidando toda expresión evaluada hasta el momento. Esto se hace mediante la opción **Quit Kernel** del menú **Evaluation**

- En todo momento se puede pedir ayuda interactiva sobre la sintaxis de un comando escribiendo el signo **?** seguido del nombre del comando (o parte de él, utilizando el comodín ***). Por ejemplo:

In[13]:=
```
?Appen*
```
▼ System`

Append	AppendTo

In[14]:=
```
?AppendTo
```
AppendTo[*s*, *elem*] appends *elem* to the value of *s*, and resets *s* to the result. ≫

La primera expresión devuelve una lista con los comandos que empiezan con **Appen**. La segunda devuelve una ayuda breve acerca de la sintaxis de la instrucción **AppendTo**. Siempre es posible obtener más información en la opción **Find Selected Function** o **Virtual Book** del menú **Help**.

- Lo mismo que en el resto de entornos de programación, es posible incluir comentarios en *Mathematica* encerrando el contenido del comentario entre los pares de símbolos **(*** y ***)**.

```
(* esto es un comentario *) Print["y esto no lo es..."]
                                escribe
```

- *Mathematica* considera los literales **True** y **False** como elementos del **tipo booleano**.

Listas

- Es posible indexar los elementos de una lista del mismo modo que se indexan los elementos de un vector en otros lenguajes de programación. Es importante tener en cuenta que la sintaxis utiliza dobles corchetes y que el primer elemento de la lista tiene índice uno. Por ejemplo:

In[24]:=
```
unalistamas[[1]]
```
Out[24]=
```
a
```

In[25]:=
```
unalistamas[[5]]
```
Out[25]=
```
{a}
```

También es posible acceder a los elementos de listas anidadas, por ejemplo, véase que las siguientes expresiones son equivalentes:

In[26]:=
```
unalistamas[[4]][[2]]
```
Out[26]=
```
b
```

In[27]:=
```
unalistamas[[4, 2]]
```
Out[27]=
```
b
```

en ambos casos se accede al segundo elemento de la lista que ocupa el cuarto lugar en la lista **unalistamas**.

Comandos predefinidos

Es importante tener en cuenta que, a no ser que se indique lo contrario, El resultado de un comando no actualiza los parámetros con los que se invoca. *Mathematica* dispone de multitud de comandos útiles para casi todas, si no todas, las áreas de la Ingeniería o las Matemáticas.

Comandos sobre listas

Entre los comandos prededinidos de *Mathematica*, hay algunos que consideran las listas como tales. En este conjunto, de los más útiles para las prácticas de la asignatura se incluyen:

- Join: Concatena dos listas, por ejemplo:

In[16]:=
```
l1 = {a, b};
l2 = {0, 1, 2};
Join[l1, l2]
junta
```

Out[18]=
```
{a, b, 0, 1, 2}
```

- Append: Añade el segundo parámetro como un elemento al final de la lista que se indica como primer parámetro. Por ejemplo:

In[19]:=
```
Append[l1, {0, 1, 2}]
añade
```

Out[19]=
```
{a, b, {0, 1, 2}}
```

- AppendTo: Equivalente a Append pero actualizando la lista recibida como primer parámetro

In[20]:=
```
AppendTo[l1, {0, 1, 2}]
añade al final
(**
   es totalmente equivalente a
   l1=Append[l1,{0,1,2}]
      añade
**)
```

Out[20]=
```
{a, b, {0, 1, 2}}
```

- Prepend/PrependTo: Equivalente a Append/AppendTo pero añadiendo el elemento al principio de la lista en lugar de al final
- First: Devuelve el primer elemento de la lista

In[21]:=
```
First[{a, b, c, d, e}]
primero
```

Out[21]=
```
a
```

- Rest: Devuelve la lista resultante de eliminar de la recibida el primer elemento

In[22]:=
```
Rest[{a, b, c, d, e}]
todos excepto el primero
```

Out[22]=
```
{b, c, d, e}
```

- Take: Devuelve los primeros elemento de la lista, tantos como se indica en el segundo parámetro

In[23]:=
```
Take[{a, b, c, d, e}, 3]
toma
```

Out[23]=
```
{a, b, c}
```

El comando Take tiene varias funciones que varían dependiendo del segundo parámetro. Un resumen de todas ellas se obtiene evaluando la ayuda que ofrece *Mathematica*.

In[28]:=
```
? Take
```

Take[*list*, *n*] gives the first *n* elements of *list*.

Take[*list*, –*n*] gives the last *n* elements of *list*.

Take[*list*, {*m*, *n*}] gives elements *m* through *n* of *list*.

Take[*list*, seq_1, seq_2, …] gives a nested list in which elements specified by seq_i are taken at level *i* in *list*. ≫

Comandos sobre conjuntos

Otro conjunto de comandos de *Mathematica* tratan las listas como conjuntos de elementos. Entre estos destacamos:

- Union: Crea una lista con los elementos que aparecen en las listas que se reciben como parámetro. La lista de salida no contiene elementos repetidos y está ordenada. Por ejemplo:

In[29]:=
```
Union[{a, b, d, e}, {a, f, d, e}, {a, f}, {c}]
```
unión

Out[29]=
```
{a, b, c, d, e, f}
```

In[30]:=
```
(* Union puede recibir una única lista... *)Union[{b, a, c, a, a, b, b, c}]
```
unión unión

Out[30]=
```
{a, b, c}
```

- Intersection: Crea una lista con los elementos que aparecen en todas las listas que se reciben como parámetro. La lista de salida no contiene elementos repetidos y está ordenada. Por ejemplo:

In[31]:=
```
Intersection[{a, b, d, e}, {a, f, d, e}]
```
intersección

Out[31]=
```
{a, d, e}
```

- Complement: Crea una lista con los elementos que aparecen en la lista indicada como primer parámetro y que no aparecen en ninguna de las restantes listas que se reciben. Por ejemplo:

In[32]:=
```
Complement[{a, b, d, e}, {a, f, d, e}, {c}]
```
complemento

Out[32]=
```
{b}
```

- MemberQ: Devuelve **True** si el segundo parámetro es un elemento del primero.

In[33]:=
```
MemberQ[{a, {a}, b, d, e}, {a}]
```
¿contenido en?

Out[33]=
```
True
```

In[34]:=
```
MemberQ[{{a, b}, d, {e, a, f}, d, e}, a]
¿contenido en?
```

Out[34]=
```
False
```

Miscelanea

- A menudo es muy útil imprimir mensajes con el valor de determinados identificadores. El comando Print permite construir estos mensajes. Por ejemplo:

In[35]:=
```
unalista = {0, 1, 3, 5, 7, 9};
Print["el valor de unalistamas es: ", unalista];
escribe
```

```
el valor de unalistamas es: {0, 1, 3, 5, 7, 9}
```

- Cases[lista,patrón]: Devuelve una lista con los elementos del primer parámetro que concuerdan con patrón indicado en el segundo elemento. El patrón permite fijar la longitud o estructura de los elementos que interesa seleccionar y puede contener el símbolo "_", que hace la función de comodín.

In[37]:=
```
lista = {{a, a, b}, {b, b, a}, {b, b}, {a, b}};
(* elementos de lista que son listas de longitud tres... *)
Cases[lista, {_, _, _}]
casos
(* elementos que son listas de longitud dos que empiezan por a... *)
Cases[lista, {a, _}]
casos
```

Out[38]=
```
{{a, a, b}, {b, b, a}}
```

Out[39]=
```
{{a, b}}
```

Programación

Operadores lógicos y relacionales

Como en todo entorno de programación, *Mathematica* dispone de operadores lógicos y de comparación. Si bien su sintaxis es muy similar a la de otros lenguajes, es importante tener presente las diferencias.

- Negación: !

In[40]:=
```
valor = True;
        verdade
```

```
! valor
```

- Conjunción: &&

```
valor && False
```
falso

- Disyunción: ||

```
valor || False
```
falso

- Igualdad: ==

```
valor = 5;
```

```
valor == 5
```

- Desigualdad: =!=

```
valor =!= 7
```

- < , > , >= , <=

```
valor < 10
```

Estructuras de Control

Mathematica permite la evaluación y ejecución de estructuras de control de forma individual, esto es, sin necesidad de incorporarlas a una determinada función. Utilizando de forma conveniente esta característica, es fácil desarrollar y comprobar las distintas partes de un programa y, posteriormente, fusionar todas ellas para construir una función.

Al igual que con los operadores lógicos, es importante tener presente la sintaxis de las estructuras de control para evitar errores.

Aspectos generales a tener en cuenta:

- Como en todo módulo predefinido en *Mathematica*, no importa de que estructura de control estemos hablando, **la primera letra de la palabra reservada es mayúscula**
- Toda instrucción relacionada con la instrucción está encerrada entre **corchetes**
- Si hay más de una instrucción a ejecutar, estas se separan por **punto y coma**
- Como en toda instrucción en *Mathematica,* hay que separar la “instrucción de control” de la siguiente instrucción con **punto y coma**

Selección condicional

La estructura de selección sigue el mismo esquema de todo lenguaje, siendo importante tener presente:

- **Los tres bloques de la selección** (condición, conjunto de instrucciones a ejecutar si la condición es cierta y conjunto de instrucciones a ejecutar si la condición es falsa) **están separados por comas**
- El conjunto de instrucciones a ejecutar si la condición es falsa es opcional

```
clave = 23;
If[clave > 0,
 si
   Print["menor"];
   escribe
   clave = 0,
   clave = clave * 100;
 ];(* if *)
```

```
clave = 23;
If[clave > 0,
 si
   clave = 0
 ];(* if *)
```

Iteración

Nos centraremos en el uso de dos tipos de bucle:

Bucle While

Es importante tener presente que **los dos bloques de la iteración** (condición y conjunto de instrucciones a ejecutar) **están separados por comas**

```
clave = False;
        falso
i = 0;
While[! clave,
 mientras
   i++;
   If[i > 10, clave = True];
   si                    verdadero
   Print[i];
   escribe
 ];(* while *)
```

Bucle For

Al igual que en el bucle while, **los cuatro bloques de la iteración** (inicialización, condición, paso y conjunto de instrucciones a ejecutar) **están separados por comas**

```
For[i = 0, i ≤ 10, i++,
 para cada
   Print["valor de i:", i];
   escribe
 ];(* for *)
```

Módulos

Mathematica lleva incorporado un lenguaje de programación propio que permite incorporar funciones para realizar tareas específicas al mismo nivel que las funciones predefinidas. Para resolver los ejercicios de las prácticas nos centraremos en el diseño de **módulos**.

El objetivo de las prácticas es implementar algoritmos relacionados con la Teoría de Lenguajes y su aplicación al problema del *string matching*, dejando de lado otros aspectos, por lo que **asumiremos que las llamadas a los módulos son correctas**, con el número apropiado de parámetros y del tipo correcto.

El concepto de módulo es totalmente parecido al de procedimiento en otros lenguajes de programación. Su esquema genérico es:

```
NombreDelModulo[ListadeParametros] := Module[{listadevariableslocales},
                                      módulo
     instruccion;
     instruccion;
     ...
    Return[valordesalida];
    retorna
 ](* fin de la definición del modulo *)
```

Para evitar errores difíciles de depurar es importante tener presente:

- Las versiones actuales de *Mathematica* sangran el código y lo colorean en función de la sintaxis. Un sangrado o coloreado no esperado, puede indicar la presencia de un error.
- *Mathematica* no permite la modificación de los parámetros
- *Mathematica* permite indicar el tipo de cada parámetro. Sin embargo esto es opcional.
- El guión de subrayado tiene una semántica asociada en *Mathematica* y no debe usarse como parte de un identificador

In[41]:=
```
Potencia[base_, exponente_] := Module[{i, resultado},
                               módulo
  resultado = 1;
  For[i = 1, i ≤ exponente, i++,
  para cada
   resultado = resultado * base;
  ]; (* for *)
  Return[resultado];
  retorna
 ](* Potencia *)
```

Una vez definido y evaluado un módulo, pueden incluirse llamadas a él en otros módulos o bien ejecutarse de forma aislada

In[42]:=
```
Potencia[3, 1000]
```

Out[42]=
```
1 322 070 819 480 806 636 890 455 259 752 144 365 965 422 032 752 148 167 664 920 368 226 ↘
 828 597 346 704 899 540 778 313 850 608 061 963 909 777 696 872 582 355 950 954 582 100 ↘
 618 911 865 342 725 257 953 674 027 620 225 198 320 803 878 014 774 228 964 841 274 390 ↘
 400 117 588 618 041 128 947 815 623 094 438 061 566 173 054 086 674 490 506 178 125 480 ↘
 344 405 547 054 397 038 895 817 465 368 254 916 136 220 830 268 563 778 582 290 228 416 ↘
 398 307 887 896 918 556 404 084 898 937 609 373 242 171 846 359 938 695 516 765 018 940 ↘
 588 109 060 426 089 671 438 864 102 814 350 385 648 747 165 832 010 614 366 132 173 102 ↘
 768 902 855 220 001
```

Ejercicios

En lo que sigue, una palabra sobre un alfabeto A, se representará mediante una lista de símbolos de A. Por ejemplo, la palabra *ababb* se representa mediante la lista {a,b,a,b,b}. La palabra vacía se representa mediante la lista {}.

1. Escriba un módulo *Mathematica* que, dados una lista y dos enteros i y j, devuelva la lista con los elementos de las posiciones i y j intercambiados.

Pista: Recuerde que se puede acceder directamente a la componente i de la lista l de la forma l[[i]].

2. Escriba un módulo *Mathematica* que con entrada una lista y un elemento, devuelva el número de veces que el elemento aparece en la lista

Algoritmo: Inicializar un contador a cero. Recorrer la lista comparando cada elemento con el elemento de entrada. Aumentar el contador en caso de que ambos coincidan.

3. Escriba un módulo *Mathematica* que devuelva el conjunto de prefijos de una palabra x recibida como parámetro

Algoritmo: Inicializar una lista de salida con la lista vacía. Para i igual a cero, hasta la longitud de la lista, obtener el segmento de longitud i utilizando la función Take. Añadir la lista obtenida en la lista de salida.

4. Escriba un módulo *Mathematica* que, teniendo como entrada una palabra x, y un entero k, devuelva el conjunto de segmentos de x de longitud k

Pista: La última posición en la que puede empezar un segmento de longitud k es la posición n-k+1, siendo n la longitud de la palabra.