# PRÁCTICA Nº8

## Números Naturales y Enteros (Parte I):
## *Cálculos elementales con números enteros, Algoritmo de Euclides e Identidad de Bezout.*

### 1.- CÁLCULOS ELEMENTALES CON NÚMEROS ENTEROS.

Estas son algunas de las funciones más usuales utilizadas para los cálculos con números enteros:

| *Función:* | |
|---|---|
| **Mod[m, n]** | Calcula el resto que resulta de dividir m entre n. |
| **Quotient[n, m]** | Calcula el cociente que resulta de dividir n entre m. |
| **Prime[n]** | Nos devuelve el n-ésimo número primo. |
| **FactorInteger[n]** | Calcula una factorización de n. |
| **PrimeQ[expr]** | Comprueba si expr es un número primo. |
| **Divisors[n]** | Nos da una lista de divisores de n. |
| **PrimePi[x]** | Nos dice el número de primos menores o iguales que x. |
| **IntegerDigits[n]** | Nos da una lista de los dígitos decimales de n. |
| **IntegerQ[expr]** | Comprueba si expr es un número entero. |
| **LCM[n1,n2, ... ]** | Calcula el mínimo común múltiplo. |
| **GCD[n1,n2, ... ]** | Calcular el máximo común divisor. |
| **Abs[n]** | Calcula el valor absoluto. |
| **Sign[n]** | Devuelve -1, 0 o 1 dependiendo de si n es negativo, cero o positivo. |
| **Negative[x]** | Devuelve True si x es negativo. |
| **NonNegative[x]** | Devuelve True si x no es un número negativo. |
| **Positive[x]** | Devuelve True si x es positivo. |
| **Floor[x]** | Calcula el mayor entero menor que x. |
| **Ceiling[x]** | Calcula el menor entero mayor que x. |
| **IntegerPart[x]** | Calcula la parte entera de x. |
| **Round[x]** | Calcula el entero más cercano a x. |
| **FractionalPart[x]** | Calcula la parte fraccionaria de x. |

Ejercicio 1:

- Realizar un programa que determine si un número es entero o no, par o impar y positivo o negativo. Aplicarlo a los siguientes números: -1, 0, 3, 3.12, 1/4.
- Utilizando un bucle, escribir un programa que haga lo mismo que **PrimeQ[ - ]** (usar la definición de número primo).

**<u>Solución:</u>**

## 2.- ALGORITMO DE EUCLIDES.

Si queremos calcular el máximo común divisor y mínimo común múltiplo de dos números enteros, realmente utilizaríamos las funciones de las que dispone el Mathematica para tal efecto (véase el apartado anterior).

Sin embargo para realizar posteriores cálculos, nos será útil tener implementado el algoritmo de Euclides. Para ello resolvemos el siguiente ejemplo.

*Ejemplo 1:*

Calcular usando el algoritmo de Euclides el máximo común divisor y mínimo común múltiplo de 270 y 1500.

SOLUCIÓN:

| PROGRAMA | COMENTARIOS |
|---|---|
| **n1=270;**<br>**n2=1500;** | Valores a los que aplicamos el algoritmo de Euclides. |
| **a=Abs[n1];**<br>**b=Abs[n2];** | Utilizaremos sus valores absolutos. |
| **If [a<b,a=b;b=Abs[n1];];**<br>**m=1;** | Distinguimos el mayor. |
| **While[m>0,**<br>**m=Mod[a,b];**<br>**a=b;**<br>**b=m;**<br>**];** | Algoritmo de Euclides. |
| **Print["m.c.d.(",n1,",",n2,")=",a]**<br>**Print["m.c.m.(",n1,",",n2,")=",Abs[(n1*n2)/a]]** | Salida de datos. |

Ejercicio 2:

Calcular el máximo común divisor y el mínimo común múltiplo de 5005 y 12345.

**Solución:**

## 3.- IDENTIDAD DE BEZOUT.

Como ya es sabido para calcular la identidad de Bezout es necesario aplicar el algoritmo de Euclides. Utilizándolo podemos escribir un programa que la calcule:

*Ejemplo 2:*

Calcular la identidad de Bezout para 270 y -3120.

<u>SOLUCIÓN:</u>

| PROGRAMA | COMENTARIOS |
|---|---|
| Clear[valor1,valor2]; | Necesario para poder utilizar el programa varias veces. |
| n1=270;<br>n2=-3120; | In troducimos los datos. |
| If[Abs[n1]>Abs[n2], temp=n1; n1=n2; n2=temp]; | |
| Signo1=n1/Abs[n1];<br>Signo2=n2/Abs[n2]; | Comprobamos los signos. |
| a=Abs[n1];<br>b=Abs[n2]; | Calculamos los valores absolutos. |
| If [a<b,a=b;b=Abs[n1];n3=a;n4=b;]; | Distinguimos el mayor de los dos. |
| r=1; | |
| Cocientes={}; | Creamos una lista para los cocientes. |
| s=0; | |
| While[r>0,<br>q=Quotient[a,b];<br>r=Mod[a,b];<br>a=b;<br>b=r;<br>s=s+1;<br>AppendTo[cocientes,q];<br>]; | Algoritmo de Euclides. |
| Listam=Table[0,{i,s}];<br>Listam[[1]]=valor1;<br>Listam[[2]]=valor2;<br>For [f=3,f<s+1,f++,<br>Listam[[f]]=listam[[f-2]]-(listam[[f-1]]*cocientes[[f-2]])<br>];<br>Bezout:=Simplify[listam[[s-1]]-(listam[[s]]*cocientes[[s-1]])]; | Volvemos hacia atrás para obtener la identidad de Bezout. |
| Valor1=1;<br>Valor2=0;<br>Valoru=Bezout; | Calcula 'u', |
| Valor1=0;<br>Valor2=1;<br>Valordev=Bezout; | Calcula 'v'. |
| Valoru=Valoru*Signo2; | Cambia el signo de 'u'. |
| Valordev=Valordev*Signo1; | Cambia el signo de 'v'. |
| Print["m.c.d.{",n1,",",n2,"}=",a]<br>Print["m.c.m.{",n1,",",n2,"}=",(n3*n4)/a]<br>Print["Identidad de Bezout: ",a," = ",n2,"·(",Valoru,") + ",n1,"·(",Valordev,")."] | Muestra en pantalla el resultado. |

Ejercicio 3:

a) Calcular la identidad de Bezout para 1560 y 781250.
b) Resolver utilizando la identidad de Bezout la siguiente ecuación:

$$1000x + 33y = 7,$$

sabiendo que x e y son números enteros.

**Solución:**