Para cada una de estas constantes podemos hallar valores aproximados con cualquier precisión. Por ejemplo:

Ejemplo 2.13. Calcular una aproximación decimal del número π con 39 cifras decimales.

In[]:= **N[Pi, 40]**

Out[]= `3.14159265358979323846264338327950288419 7`

Obtenemos el valor aproximado del número π con 40 cifras significativas y por tanto con 39 cifras decimales.

■

Mathematica tiene definidas la mayor parte de las funciones matemáticas elementales y otras que se utilizan con frecuencia en la práctica para la resolución de problemas técnicos de todo tipo. Generalmente, dichas funciones se nombran abreviando su denominación en inglés. Para obtener el resultado, tenemos que escribirlas teniendo en cuenta su sintaxis (véase capítulo 1).

En la siguiente tabla presentamos algunas de estas funciones sin analizarlas con detenimiento. Una descripción detallada de cada una de ellas, podemos encontrarla en la ayuda del programa:

Funciones elementales	
Sin[x], Cos[x], Tan[x], **Csc[x], Sec[x], Cot[x],** **ArcSin[x], ArcCos[x], ArcTan[x] …**	Funciones trigonométricas
Exp[x] **Log[x]** **Log[b, x]** **Sqrt[x]** **Surd[x, n]**	Función exponencial Logaritmo de “x” en base *e* Logaritmo de “x” en base “b” Raíz cuadrada de “x” Raíz “n”-ésima de “x”
Round[x] **Floor[x]** **Ceiling[x]** **Mod[a, b]** **Quotient[a, b]** **Sign[x]** **Abs[x]** **Max[x,y,z,...]** **Min[x,y,z,...]** **n!**	Redondea “x” a un entero Mayor entero menor que “x” Menor entero mayor que “x” Resto de dividir “a” entre “b” (con b > 0) Cociente de dividir “a” entre “b” (con b > 0) Devuelve el signo de “x” Valor absoluto de “x” Máximo valor de “x, y, z, ...” Mínimo valor de “x, y, z, ...” Factorial de “n”
IntegerQ[x] **PrimeQ[x]**	Devuelve si “x” es o no entero Devuelve si “x” es o no primo

Tabla 2.4. Funciones elementales.

Ejemplo 2.14. Veamos cómo actúan alguna de estas funciones. Por ejemplo, calculemos el seno de 30º:

Ejemplo 2.22. Definir la función de Fibonacci y evaluarla en 10 (aunque ya se encuentra implementada en Mathematica con el nombre **Fibonacci**).

La función de Fibonacci, f: $\mathbb{Z} \to \mathbb{Z}$ se define de forma recursiva mediante la expresión $f(n) = f(n-2) + f(n-1)$ siendo $f(1) = f(2) = 1$. Con Mathematica se puede definir de la siguiente manera:

```
In[]:=   fib[1]=1; fib[2]=1;
         fib[n_] : = fib[n – 2]+fib[n – 1]

In[]:=   fib[10]

Out[]=   55
```

Vemos que la función anterior sólo tiene sentido para números naturales; si se intenta calcular el valor de dicha función para un número decimal, usando la definición anterior, caeríamos en un ciclo recursivo sin final y Mathematica lo indica con un mensaje de error.

```
In[]:=   fib[3.5]

         $RecursionLimit::reclim : Recursion depth of 1024 exceeded. >>

Out[]=   Hold[fib[-2040.5-2]]
```

Para evitar este problema, podría indicarse en la definición de la función que su argumento sólo puede tomar valores enteros:

```
In[]:=   fib2[1]=1; fib2[2]=1;
         fib2[n_Integer]:=fib2[n – 2] + fib2[n – 1]

In[]:=   fib2[3.5]

Out[]=   fib2[3.5]
```

Ahora, el programa no intenta calcular el número de Fibonacci de 3.5 y devuelve la función sin evaluar. Mathematica tiene también esta función implementada: **Fibonacci[n]**.

■

5.3. FUNCIONES Y PROGRAMACIÓN. VARIABLES LOCALES

Frecuentemente necesitaremos que una función realice varios cálculos o ejecute varias instrucciones de código teniendo en cuenta los valores que le pasemos. Dicho código puede ser más o menos complejo y también puede contener variables que únicamente se usan dentro de esta función, a tales variables las denominaremos locales o de ámbito local, porque sólo tienen repercusión dentro de la función, en contraposición con las globales, que tendrán

sentido en cualquier parte del código que ejecutemos. Entonces, en estos casos, las funciones o procedimientos las definiremos de otra forma, para ello usaremos la función Module[]:

nombre[var1_, var2_,…] := Module[{var1local, var2local,…}, cuerpo]

Como vemos Module[] tiene dos argumentos o entradas:

1) El primero esta formado por todas las variables locales separadas por comas y entre llaves.
2) El segundo, el código que queramos que se ejecute, compuesto por cuantas instrucciones deseemos, todas ellas separadas entre sí por punto y coma, y que denominaremos 'cuerpo' de la función.

Para usar variables locales en nuestro código también podremos utilizar Block[], que funciona de manera similar y tiene los mismos argumentos:

Block[{var1local, var2local,…}, cuerpo]

Las funciones, procedimientos de programación o programas los mostraremos y maquetaremos así:

PROGRAMA	COMENTARIOS
Definimos variables y funciones previas;	Comentarios.
CÓDIGO	Comentarios y explicaciones.

Programa *.*. Nombre.

Ilustración 2.2. Esquema de un programa, función o procedimiento.

Ejemplo 2.23. En este ejemplo practicaremos el uso de variables locales.

In[]:=

```
a=0;
c="Global";
Block[{a,b},
  a=1;
  b="Varible local";
  Print["El valor de la variable local a es ", a];
  Print["El valor de la variable local b es ", b];
  Print["El valor de la variable global c es ", c];
];
Print["El valor de la variable global a es ",a]
Print ["La variable b es local, fuera de Block[] no está
  definida: ",b]
```

Out[]=

```
El valor de la variable local a es 1
El valor de la variable local b es Varible local
El valor de la variable global c es Global
```

A partir de la definición de una lista, podemos calcular con ella cualquier operación, como si se tratara de otra variable, efectuando dicha operación en cada elemento de la misma. En Mathematica podemos escribir:

In[]:= **l={2, x, Pi/2, 3+I};**

In[]:= **h=2*l**

Out[]= `{4,2x,π,6+2i}`

In[]:= **Sin[h]//N**

Out[]= `{-0.756802,Sin[2.x],0.,-1.05122+3.4824i}`

Las listas pueden estar formadas por elementos de distintos tipos, por ejemplo:

In[]:= **l2={{}, {1, 2}, x, 4, {4}, {{4}}, rojo};**

Esta lista está formada por 7 elementos, entre los que encontramos un número, 4, dos variables, 'x' y 'rojo', una lista vacía, {}, otra con un elemento, {4}, y otra más que contiene a su vez otra lista, {{4}}.

■

En la siguiente tabla podemos ver las funciones más habituales que podemos emplear para manejar listas:

Funciones para manipular listas		
Referirse a un elemento de la lista	**lista[[i]]**	El *i*-ésimo elemento de "lista".
	First[lista]	Primer elemento de "lista".
	Last[lista]	Último elemento de "lista".
Añadir elementos a la lista	**Append[lista, elemento]**	Devuelve una lista resultado de añadir "elemento" al final de "lista", pero no modifica el contenido de "lista".
	AppendTo[lista,elemento]	Modifica "lista" añadiendo "elemento" en última posición.
	Prepend[lista,elemento]	Devuelve una lista resultado de añadir "elemento" al principio de "lista", pero no modifica el contenido de "lista".
	PrependTo[lista,elemento]	Modifica "lista" añadiendo "elemento" en primera posición.
	Insert[lista,elemento,n]	Devuelve una nueva lista resultado de añadir "elemento" en la posición "n" de "lista".
Eliminar elementos	**Delete[lista,n]**	Devuelve una nueva lista resultado de eliminar el elemento de la posición "n" de "lista".

	Drop[lista,n]	Devuelve la lista después de quitarle los primeros "n" términos.
Pegar listas	**Join[lista1,lista2,...]**	Devuelve una nueva lista resultado de pegar "lista1", "lista2" ...
Seleccionar parte de una lista	**Take[lista,{m,n}]**	Devuelve una lista compuesta por los elementos que van desde el "m"-ésimo al "n"-ésimo de "lista.
	TakeWhile[lista,test]	Devuelve los elementos de la lista "lista" desde el primero en adelante hasta encontrar uno que no verifique "test".
	Select[lista,test]	Devuelve los elementos de la lista "lista" que verifique "test".
Ordenar listas	**Sort[lista]**	Ordena los elementos de "lista".
	Ordering[lista]	Devuelve las posiciones que ocuparían los elementos de "lista" tras ordenarlos con Sort[].
Otras	**Length[lista]**	Longitud o número de elemetos de "lista"
	Position[lista, elemento]	Nos devuelve las posiciones en la lista donde se encuentra el elemento.

Tabla 3.1. Funciones para listas.

Ejemplo 3.2. Consideramos un par de listas y probamos con ellas las distintas funciones que aparecen en el cuadro anterior:

In[]:=
```
lista1={0,2, -7, y, 4, 3x^2 +1, E, 5+2I};
lista2={a,l,g,e,b,r,a};
```

Para calcular la longitud de las listas:

In[]:=
```
Length[lista1]
Length[lista2]
```

Out[]=
```
8
7
```

Podemos juntarlas y obtener una nueva lista:

In[]:=
```
lista3=Join[lista1,lista2]
Length[lista3]
```

Out[]=
```
{0,2,-7,b,4,1+3x^2,e,5+2i,a,l,g,e,b,r,a}
15
```

Así vemos que la nueva lista tiene 7 + 8 = 15 elementos. Para obtener el elemento que ocupa el cuarto lugar de "lista1":

```
f[3]
```

Out[]=
```
0.25
0
```

■

2.2.2 SWITCH

Switch[expresión,*forma1*,valor1,forma2,valor2,...]

Evalúa "expresión", comprueba si el resultado coincide para algún *i* con "forma*i*" y devuelve "valor*i*".

Ejemplo 4.7. La siguiente función sustituye letras por la posición que tienen en el abecedario:

In[]:=
```
f[variable_] := Switch[variable,
  a, 1, b, 2, c, 3, d, 4, e, 5, f, 6, g, 7, h, 8, i, 9, j, 10, k, 11, l, 12, m,
  13, n, 14, ñ, 15, o, 16, p, 17, q, 18, r, 19, s, 20, t, 21, u, 22, v,
  23, w, 24, x, 25, y, 26, z, 27];
lista = {a, l, g, e, b, r, a};
Table[f[lista[[i]]], {i, Length[lista]}]
```

Out[]=
```
{1,12,7,5,2,19,1}
```

■

2.2.3 PIECEWISE

Piecewise[{{valor1,*condición1*},{valor2,condición2},...},valor]

Representa una función a trozos que alcanza el valor "valor*i*" en la región especificada por "condición*i*", si no se encuentra en ninguna de las regiones entonces devuelve "valor".

Ejemplo 4.8. Definimos la misma función del ejemplo 4.6.:

In[]:=
```
g[x_] := Piecewise[{{x^2, 0<=x<= 1}, {2-x^2, 1<=x<=2}}, 0]
g[0.5]
g[3]
```

Out[]=
```
0.25
0
```

■

3. EJERCICIOS

Ejercicio 5.1. Determinar según tu DNI el valor de verdad de la siguiente forma enunciativa:

Mi DNI es par o termina en 1, 3, 7; y es múltiplo de 3 o de 5.

■

Ejercicio 5.2. Evaluar las siguientes formas enunciativas en las combinaciones de valores de verdad indicadas.

a) $\sim x_1$; $x_1 = V$.
b) $x_1 \leftrightarrow x_2$; $x_1 = V$, $x_2 = F$.
c) $((\sim x_1) \wedge x_2) \rightarrow x_3$; $x_1 = V$, $x_2 = F$, $x_3 = V$.
d) $(x_1 \vee x_1) \leftrightarrow (\sim x_2)$; $x_1 = V$, $x_2 = V$.
e) $[(x_1 \downarrow x_2) \uparrow (\sim x_3)] \rightarrow (\sim x_3)$; $x_1 = V$, $x_2 = F$, $x_3 = V$.
f) $(\sim((\sim(x_1 \wedge x_2)) \rightarrow x_3)) \uparrow (x_4 \vee x_5)$; $x_5 = F$ y tomando el resto de variables cualesquiera valores.

■

Ejercicio 5.3. Traducir a forma simbólica utilizando lógica proposicional y evaluar en la combinación de valores de verdad que elijas las formas enunciativas que las representan.

a) Si un alumno se matricula en Informática entonces estudiará Álgebra y asistirá a prácticas.
b) La suma de números enteros es natural si ambos son positivos.
c) La suma de dos números enteros es par si y sólo si ambos son pares o impares.
d) Si U es un subespacio entonces U no es el vacío supuesto que U contenga al 0.
e) Muchos estudiantes estudian Lógica y Álgebra en el primer curso de la carrera.
f) $(x + y)z = xz + yz$.
g) Si $ab = 0$ entonces $a = 0$ o $b = 0$.

■

Ejercicio 5.4. Calcular las tablas de verdad de las formas enunciativas del ejercicio anterior.

■

Ejercicio 5.5. Definir una conectiva cuya tabla de verdad sea:

p	q	$p \leftarrow q$
V	V	V
V	F	V
F	V	F
F	F	V

■

Ejercicio 5.6. Calcular las tablas de verdad de las siguientes formas enunciativas:

a) $(\sim r \vee p) \rightarrow q$.
b) $(p \rightarrow q) \wedge r$.
c) $[(p \rightarrow q) \wedge r] \rightarrow [(\sim r \vee p) \rightarrow q]$.
d) $(p \rightarrow (q \vee p))$.
e) $((q \vee r) \rightarrow ((\sim r) \rightarrow q))$.
f) $(((\sim p) \rightarrow q) \rightarrow (((\sim s) \rightarrow (\sim q)) \rightarrow p))$.
g) $(p \leftrightarrow (q \leftarrow p))$.
h) $((q \rightarrow r) \wedge (s \rightarrow (\sim q)))$.
i) $(p \leftarrow q) \leftrightarrow (\sim((q \wedge (\sim r)) \leftarrow (s \leftarrow (\sim p))))$.
j) $\sim(\sim((p \rightarrow \sim(q \rightarrow r)) \rightarrow \sim((p \uparrow q) \downarrow r)))$.
k) $(p \leftarrow q) \rightarrow (q \leftrightarrow r)$.
l) $(\sim(q_1 \wedge q_2) \leftrightarrow (q_3 \vee q_4)) \rightarrow (\sim(q_5 \downarrow q_6))$.

■

Ejercicio 5.7. ***Leyes de De Morgan.*** Para cualesquiera formas enunciativas $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$,

a) $\sim(\bigwedge_{i=1}^{n} \mathcal{A}_i) \Leftrightarrow \bigvee_{i=1}^{n} (\sim \mathcal{A}_i)$,
b) $\sim(\bigvee_{i=1}^{n} \mathcal{A}) \Leftrightarrow \bigwedge_{i=1}^{n} (\sim \mathcal{A}_i)$.

Comprobar con Mathematica las Leyes de De Morgan para dos y tres formas enunciativas. (Usar el principio de sustitución y cambiar las formas enunciativas por variables de enunciado).

■

Ejercicio 5.8. Definir una conectiva cuya tabla de verdad sea:

p	q	$p \oplus q$
V	V	F
V	F	V
F	V	V
F	F	F

La llamaremos XOR es decir, 'o exclusivo'. Comprueba que su tabla de verdad coincide con la de $(p \vee q) \wedge (\sim(p \wedge q))$.

I. Calcular las tablas de verdad de las siguientes formas enunciativas y comentar los resultados obtenidos.

a) $\sim(p \leftrightarrow q)$.
b) $(p \vee q) \wedge ((\sim p) \vee (\sim q))$.
c) $(p \wedge (\sim q)) \vee ((\sim p) \wedge q)$.

II. Calcular las tablas de verdad de las siguientes formas enunciativas $p_1 \oplus p_2$, $p_1 \oplus p_2 \oplus p_3$, $p_1 \oplus p_2 \oplus p_3 \oplus p_4$. Deducir a partir de lo anterior la tabla de verdad de la forma enunciativa $p_1 \oplus p_2 \oplus \ldots \oplus p_n$ para cualquier $n > 4$.