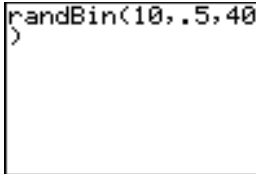# *Chapter 2:*
# *Math, Angle, and Test Operations*

## Getting Started: Coin Flip

**Getting Started is a fast-paced introduction. Read the chapter for details.**

**Suppose you want to model flipping a fair coin 10 times. You want to track how many of those 10 coin flips result in heads. You want to perform this simulation 40 times. With a fair coin, the probability of a coin flip resulting in heads is 0.5 and the probability of a coin flip resulting in tails is 0.5.**

1. Begin on the home screen. Press MATH ◄ to display the **MATH PRB** menu. Press **7** to select **7:randBin(** (random Binomial). **randBin(** is pasted to the home screen. Press **10** to enter the number of coin flips. Press ⎡,⎤. Press ⎡.⎤ **5** to enter the probability of heads. Press ⎡,⎤. Press **40** to enter the number of simulations. Press ⎡)⎤.

randBin(10,.5,40
)

2. Press [ENTER] to evaluate the expression. A list of 40 elements is generated with the first 7 displayed. The list contains the count of heads resulting from each set of 10 coin flips. The list has 40 elements because this simulation was performed 40 times. In this example, the coin came up heads five times in the first set of 10 coin flips, five times in the second set of 10 coin flips, and so on.

```
randBin(10,.5,40
)
{5 5 7 4 6 6 3 …
```

3. Press [▶] or [◀] to view the additional counts in the list. Ellipses (**...**) indicate that the list continues beyond the screen.

```
randBin(10,.5,40
)
{5 5 7 4 6 6 3 …
Ans→L₁
{5 5 7 4 6 6 3 …
```

4. Press [STO▶] [2nd] [L1] [ENTER] to store the data to the list name **L1**. You then can use the data for another activity, such as plotting a histogram (Chapter 12).

   **Note:** Since **randBin(** generates random numbers, your list elements may differ from those in the example.

```
randBin(10,.5,40
)
{5 5 7 4 6 6 3 …
Ans→L₁
…2 5 3 6 5 7 5 …
```

# Keyboard Math Operations

## Using Lists with Math Operations

Math operations that are valid for lists return a list calculated element by element. If you use two lists in the same expression, they must be the same length.

```
{1,2}+{3,4}+5
         {9 11}
```

## + (Addition), – (Subtraction), ∗ (Multiplication), ⁄ (Division)

You can use **+** (addition, $\boxed{+}$), – (subtraction, $\boxed{-}$), ∗ (multiplication, $\boxed{\times}$), and ⁄ (division, $\boxed{\div}$) with real and complex numbers, expressions, lists, and matrices. You cannot use ⁄ with matrices.

*valueA***+***valueB*          *valueA* – *valueB*
*valueA***∗***valueB*          *valueA* ⁄ *valueB*

## Trigonometric Functions

You can use the trigonometric (trig) functions (sine, $\boxed{\text{SIN}}$; cosine, $\boxed{\text{COS}}$; and tangent, $\boxed{\text{TAN}}$) with real numbers, expressions, and lists. The current angle mode setting affects interpretation. For example, **sin(30)** in **Radian** mode returns -**.9880316241**; in **Degree** mode it returns **.5**.

**sin(***value***)**          **cos(***value***)**          **tan(***value***)**

You can use the inverse trig functions (arcsine, [2nd] [SIN⁻¹]; arccosine, [2nd] [COS⁻¹]; and arctangent, [2nd] [TAN⁻¹]) with real numbers, expressions, and lists. The current angle mode setting affects interpretation.

**sin⁻¹(***value***)**        **cos⁻¹(***value***)**        **tan⁻¹(***value***)**

**Note:** The trig functions do not operate on complex numbers.

## ^ (Power), ² (Square), √( (Square Root)

You can use **^** (power, [^]), **²** (square, [x²]), and **√(** (square root, [2nd] [√]) with real and complex numbers, expressions, lists, and matrices. You cannot use **√(** with matrices.

*value***^***power*          *value***²**          **√(***value***)**

## ⁻¹ (Inverse)

You can use **⁻¹** (inverse, [x⁻¹]) with real and complex numbers, expressions, lists, and matrices. The multiplicative inverse is equivalent to the reciprocal, $1/x$.

*value*⁻¹

```
5⁻¹
              .2
```

## log(, 10^(, ln(

You can use **log(** (logarithm, $\boxed{\text{LOG}}$), **10^(** (power of 10, $\boxed{\text{2nd}}$ [$10^x$]), and **ln(** (natural log, $\boxed{\text{LN}}$) with real or complex numbers, expressions, and lists.

**log(***value***)**          **10^(***power***)**          **ln(***value***)**

## e^( (Exponential)

**e^(** (exponential, $\boxed{\text{2nd}}$ [$e^x$]) returns the constant **e** raised to a power. You can use **e^(** with real or complex numbers, expressions, and lists.

**e^(***power***)**

```
e^(5)
            148.4131591
```

## e (Constant)

**e** (constant, $\boxed{\text{2nd}}$ [e]) is stored as a constant on the TI-83 Plus. Press $\boxed{\text{2nd}}$ [e] to copy **e** to the cursor location. In calculations, the TI-83 Plus uses 2.718281828459 for **e**.

```
e
            2.718281828
```

## − (Negation)

- (negation, [(-)]) returns the negative of *value*. You can use - with real or complex numbers, expressions, lists, and matrices.

-*value*

EOS™ rules (Chapter 1) determine when negation is evaluated. For example, -**A²** returns a negative number, because squaring is evaluated before negation. Use parentheses to square a negated number, as in **(-A)²**.

```
2→A:{-A²,(-A)²,-
2²,(-2)²}
        {-4 4 -4 4}
```

**Note:** On the TI-83 Plus, the negation symbol (-) is shorter and higher than the subtraction sign (–), which is displayed when you press [-].

## π (Pi)

π (Pi, [2nd] [π]) is stored as a constant in the TI-83 Plus. In calculations, the TI-83 Plus uses 3.1415926535898 for π.

```
π
        3.141592654
```

# MATH Operations

## MATH Menu

To display the **MATH** menu, press [MATH].

| MATH | NUM CPX PRB | |
|---|---|---|
| 1: ▶Frac | Displays the answer as a fraction. |
| 2: ▶Dec | Displays the answer as a decimal. |
| 3: $^3$ | Calculates the cube. |
| 4: $^3\sqrt{(}$ | Calculates the cube root. |
| 5: $^x\sqrt{}$ | Calculates the $x^{th}$ root. |
| 6: fMin( | Finds the minimum of a function. |
| 7: fMax( | Finds the maximum of a function. |
| 8: nDeriv( | Computes the numerical derivative. |
| 9: fnInt( | Computes the function integral. |
| 0: Solver... | Displays the equation solver. |

## ▶Frac, ▶Dec

▶**Frac** (display as a fraction) displays an answer as its rational equivalent. You can use ▶**Frac** with real or complex numbers, expressions, lists, and matrices. If the answer cannot be simplified or the resulting denominator is more than three digits, the decimal equivalent is returned. You can only use ▶**Frac** following *value*.

*value* ►**Frac**

►**Dec** (display as a decimal) displays an answer in decimal form. You can use ►**Dec** with real or complex numbers, expressions, lists, and matrices. You can only use ►**Dec** following *value*.

*value* ►**Dec**

```
1/2+1/3►Frac
              5/6
Ans►Dec
      .8333333333
```

## ³(Cube), ³√( (Cube Root)

**³** (cube) returns the cube of *value*. You can use **³** with real or complex numbers, expressions, lists, and square matrices.

*value*³

**³√(** (cube root) returns the cube root of *value*. You can use **³√(** with real or complex numbers, expressions, and lists.

**³√(***value***)**

```
{2,3,4,5}³
       {8 27 64 125}
³√(Ans)
            {2 3 4 5}
```

## ˣ√ (Root)

ˣ√ (*xth root*) returns the *xth root* of *value*. You can use ˣ√ with real or complex numbers, expressions, and lists.

*xthroot*ˣ√*value*

```
5 ˣ√32
            2
```

## fMin(, fMax(

**fMin(** (function minimum) and **fMax(** (function maximum) return the value at which the local minimum or local maximum value of *expression* with respect to *variable* occurs, between *lower* and *upper* values for *variable*. **fMin(** and **fMax(** are not valid in *expression*. The accuracy is controlled by *tolerance* (if not specified, the default is 1E-5).

**fMin(***expression***,***variable***,***lower***,***upper*[**,***tolerance*]**)**
**fMax(***expression***,***variable***,***lower***,***upper*[**,***tolerance*]**)**

**Note:** In this guidebook, optional arguments and the commas that accompany them are enclosed in brackets ([ ]).

```
fMin(sin(A),A,-π
,π)
        -1.570797171
fMax(sin(A),A,-π
,π)
         1.570797171
```

## nDeriv(

**nDeriv(** (numerical derivative) returns an approximate derivative of *expression* with respect to *variable*, given the *value* at which to calculate the derivative and ε (if not specified, the default is 1E⁻3). **nDeriv(** is valid only for real numbers.

**nDeriv(***expression*,*variable*,*value*[,ε]**)**

**nDeriv(** uses the symmetric difference quotient method, which approximates the numerical derivative value as the slope of the secant line through these points.

$$f'(x) = \frac{f(x+\varepsilon) - (f(x-\varepsilon)}{2\varepsilon}$$

As ε becomes smaller, the approximation usually becomes more accurate.

```
nDeriv(A^3,A,5,.
01)
          75.0001
nDeriv(A^3,A,5,.
0001)
              75
```

You can use **nDeriv(** once in *expression*. Because of the method used to calculate **nDeriv(**, the TI-83 Plus can return a false derivative value at a nondifferentiable point.

## fnInt(

**fnInt(** (function integral) returns the numerical integral (Gauss-Kronrod method) of *expression* with respect to *variable*, given *lower* limit, *upper* limit, and a *tolerance* (if not specified, the default is 1E-5). **fnInt(** is valid only for real numbers.

**fnInt(***expression***,***variable***,***lower***,***upper*[**,***tolerance*]**)**

```
fnInt(A²,A,0,1)
       .3333333333
```

**Tip:** To speed the drawing of integration graphs (when **fnInt(** is used in a Y= equation), increase the value of the **Xres** window variable before you press GRAPH.

# Using the Equation Solver

## Solver

**Solver** displays the equation solver, in which you can solve for any variable in an equation. The equation is assumed to be equal to zero. **Solver** is valid only for real numbers.
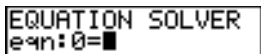
When you select **Solver**, one of two screens is displayed.

- The equation editor (see step 1 picture below) is displayed when the equation variable **eqn** is empty.

- The interactive solver editor is displayed when an equation is stored in **eqn**.

## Entering an Expression in the Equation Solver

To enter an expression in the equation solver, assuming that the variable **eqn** is empty, follow these steps.

1. Select **0:Solver** from the **MATH** menu to display the equation editor.

```
EQUATION SOLVER
eqn:0=■
```

2. Enter the expression in any of three ways.

- Enter the expression directly into the equation solver.

- Paste a **Y=** variable name from the **VARS Y-VARS** menu to the equation solver.

- Press [2nd] [RCL], paste a **Y=** variable name from the **VARS Y-VARS** menu, and press [ENTER]. The expression is pasted to the equation solver.

The expression is stored to the variable **eqn** as you enter it.

```
EQUATION SOLVER
eqn:0=Q^3+P²-125
■
```

3. Press [ENTER] or [▼]. The interactive solver editor is displayed.

```
Q^3+P²-125=0
 Q=0
 P=0
 bound={-1ε99,1...
```

- The equation stored in **eqn** is set equal to zero and displayed on the top line.

- Variables in the equation are listed in the order in which they appear in the equation. Any values stored to the listed variables also are displayed.

- The default lower and upper bounds appear in the last line of the editor (**bound={-1ε99,1ε99}**).

- A ↓ is displayed in the first column of the bottom line if the editor continues beyond the screen.

**Tip:** To use the solver to solve an equation such as $K=.5MV^2$, enter **eqn:0=K−.5MV²** in the equation editor.

## Entering and Editing Variable Values

When you enter or edit a value for a variable in the interactive solver editor, the new value is stored in memory to that variable.

You can enter an expression for a variable value. It is evaluated when you move to the next variable. Expressions must resolve to real numbers at each step during the iteration.

You can store equations to any **VARS Y-VARS** variables, such as **Y1** or **r6**, and then reference the variables in the equation. The interactive solver editor displays all variables of all **Y=** functions referenced in the equation.

```
\Y9∎X²−4AC
\Y0=
```

```
EQUATION SOLVER
eqn:0=Y9+7
```

```
Y9+7=0
 X=0
 A=0
 C=0
 bound={-1E99,1…
```

## Solving for a Variable in the Equation Solver

To solve for a variable using the equation solver after an equation has been stored to **eqn**, follow these steps.

1. Select **0:Solver** from the **MATH** menu to display the interactive solver editor, if not already displayed.

```
Q^3+P²-125=0
 Q=0
 P=0
 bound={-1E99,1…
```

2. Enter or edit the value of each known variable. All variables, except the unknown variable, must contain a value. To move the cursor to the next variable, press ENTER or ⊡.
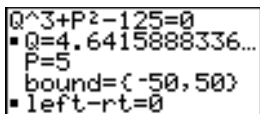
```
Q^3+P²-125=0
 Q=0
 P=5■
 bound={-1E99,1…
```

3. Enter an initial guess for the variable for which you are solving. This is optional, but it may help find the solution more quickly. Also, for equations with multiple roots, the TI-83 Plus will attempt to display the solution that is closest to your guess.

```
Q^3+P²-125=0
 Q=4■
 P=5
 bound={-1E99,1…
```

The default guess is calculated as $\frac{(upper+lower)}{2}$.

4. Edit **bound={***lower*,*upper***}**. *lower* and *upper* are the bounds between which the TI-83 Plus searches for a solution. This is optional, but it may help find the solution more quickly. The default is **bound={⁻1ᴇ99,1ᴇ99}**.

5. Move the cursor to the variable for which you want to solve and press ALPHA [SOLVE] (above the ENTER key).

```
Q^3+P²-125=0
∎Q=4.6415888336…
 P=5
 bound={-50,50}
∎left-rt=0
```

- The solution is displayed next to the variable for which you solved. A solid square in the first column marks the variable for which you solved and indicates that the equation is balanced. An ellipsis shows that the value continues beyond the screen.

  **Note:** When a number continues beyond the screen, be sure to press ▶ to scroll to the end of the number to see whether it ends with a negative or positive exponent. A very small number may appear to be a large number until you scroll right to see the exponent.

- The values of the variables are updated in memory.

- **left**-**rt=***diff* is displayed in the last line of the editor. *diff* is the difference between the left and right sides of the equation. A solid square in the first column next to **left**-**rt=** indicates that the equation has been evaluated at the new value of the variable for which you solved.

### Editing an Equation Stored to eqn

To edit or replace an equation stored to **eqn** when the interactive equation solver is displayed, press $\blacktriangle$ until the equation editor is displayed. Then edit the equation.

### Equations with Multiple Roots

Some equations have more than one solution. You can enter a new initial guess or new bounds to look for additional solutions.

### Further Solutions

After you solve for a variable, you can continue to explore solutions from the interactive solver editor. Edit the values of one or more variables. When you edit any variable value, the solid squares next to the previous solution and **left**-**rt=***diff* disappear. Move the cursor to the variable for which you now want to solve and press [ALPHA] [SOLVE].
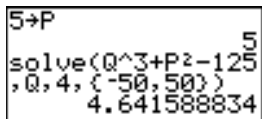
## Controlling the Solution for Solver or solve(

The TI-83 Plus solves equations through an iterative process. To control that process, enter bounds that are relatively close to the solution and enter an initial guess within those bounds. This will help to find a solution more quickly. Also, it will define which solution you want for equations with multiple solutions.

## Using solve( on the Home Screen or from a Program

The function **solve(** is available only from **CATALOG** or from within a program. It returns a solution (root) of *expression* for *variable*, given an initial *guess*, and *lower* and *upper* bounds within which the solution is sought. The default for *lower* is -1E99. The default for *upper* is 1E99. **solve(** is valid only for real numbers.

**solve(***expression***,***variable***,***guess***[,{***lower***,***upper***}])**

*expression* is assumed equal to zero. The value of *variable* will not be updated in memory. *guess* may be a value or a list of two values. Values must be stored for every variable in *expression*, except *variable*, before *expression* is evaluated. *lower* and *upper* must be entered in list format.

```
5→P
              5
solve(Q^3+P²-125
,Q,4,{-50,50})
    4.641588834
```

# MATH NUM (Number) Operations

## MATH NUM Menu

To display the **MATH NUM** menu, press MATH ▶.

| MATH **NUM** CPX PRB | |
|---|---|
| **1:** abs( | Absolute value |
| 2: round( | Round |
| 3: iPart( | Integer part |
| 4: fPart( | Fractional part |
| 5: int( | Greatest integer |
| 6: min( | Minimum value |
| 7: max( | Maximum value |
| 8: lcm( | Least common multiple |
| 9: gcd( | Greatest common divisor |

## abs(

**abs(** (absolute value) returns the absolute value of real or complex (modulus) numbers, expressions, lists, and matrices.

**abs(**_value_**)**

```
abs(-256)
              256
abs((1.25,-5.67)
)
      (1.25 5.67)
```

**Note: abs(** is also available on the **MATH CPX** menu.

## round(

**round(** returns a number, expression, list, or matrix rounded to *#decimals* (≤9). If *#decimals* is omitted, *value* is rounded to the digits that are displayed, up to 10 digits.

**round(***value*[**,***#decimals*]**)**

```
round(π,4)
            3.1416
```

```
123456789012→C
        1.23456789E11
C-round(C)
                  12
123456789012-123
456789000
                  12
```

## iPart(, fPart(

**iPart(** (integer part) returns the integer part or parts of real or complex numbers, expressions, lists, and matrices.

**iPart(***value***)**

**fPart(** (fractional part) returns the fractional part or parts of real or complex numbers, expressions, lists, and matrices.

**fPart(***value***)**

```
iPart(-23.45)
                -23
fPart(-23.45)
               -.45
```

## int(

**int(** (greatest integer) returns the largest integer $\leq$ real or complex numbers, expressions, lists, and matrices.

**int(***value***)**

```
int(-23.45)
            -24
```

**Note:** For a given *value*, the result of **int(** is the same as the result of **iPart(** for nonnegative numbers and negative integers, but one integer less than the result of **iPart(** for negative noninteger numbers.

## min(, max(

**min(** (minimum value) returns the smaller of *valueA* and *valueB* or the smallest element in *list*. If *listA* and *listB* are compared, **min(** returns a list of the smaller of each pair of elements. If *list* and *value* are compared, **min(** compares each element in *list* with *value*.

**max(** (maximum value) returns the larger of *valueA* and *valueB* or the largest element in *list*. If *listA* and *listB* are compared, **max(** returns a list of the larger of each pair of elements. If *list* and *value* are compared, **max(** compares each element in *list* with *value*.

**min(***valueA*,*valueB***)**              **max(***valueA*,*valueB***)**
**min(***list***)**                          **max(***list***)**
**min(***listA*,*listB***)**                 **max(***listA*,*listB***)**
**min(***list*,*value***)**                  **max(***list*,*value***)**

```
min(3,2+2)
              3
min({3,4,5},4)
          {3 4 4}
max({4,5,6})
              6
```

**Note: min(** and **max(** also are available on the **LIST MATH** menu.

## lcm(, gcd(

**lcm(** returns the least common multiple of *valueA* and *valueB*, both of which must be nonnegative integers. When *listA* and *listB* are specified, **lcm(** returns a list of the lcm of each pair of elements. If *list* and *value* are specified, **lcm(** finds the lcm of each element in *list* and *value*.

**gcd(** returns the greatest common divisor of *valueA* and *valueB*, both of which must be nonnegative integers. When *listA* and *listB* are specified, **gcd(** returns a list of the gcd of each pair of elements. If *list* and *value* are specified, **gcd(** finds the gcd of each element in *list* and *value*.

**lcm(***valueA***,***valueB***)**        **gcd(***valueA***,***valueB***)**
**lcm(***listA***,***listB***)**          **gcd(***listA***,***listB***)**
**lcm(***list***,***value***)**           **gcd(***list***,***value***)**
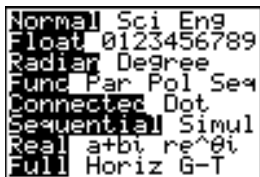
```
lcm(2,5)
              10
gcd({48,66},{64,
122})
          {16 2}
```

# Entering and Using Complex Numbers

## Complex-Number Modes

The TI-83 Plus displays complex numbers in rectangular form and polar
form. To select a complex-number mode, press MODE, and then select
either of the two modes.

- **a+b*i*** (rectangular-complex mode)
- **re^θ*i*** (polar-complex mode)



On the TI-83 Plus, complex numbers can be stored to variables. Also,
complex numbers are valid list elements.

In **Real** mode, complex-number results return an error, unless you
entered a complex number as input. For example, in **Real** mode **ln(-1)**
returns an error; in **a+b*i*** mode **ln(-1)** returns an answer.

| **Real** mode | **a+b***i* mode |
|---|---|
| `ln(-1)█` | `ln(-1)█` |
| ↓ | ↓ |
| `ERR:NONREAL ANS`<br>`1█Quit`<br>`2:Goto` | `ln(-1)`<br>`        3.141592654i` |

## Entering Complex Numbers

Complex numbers are stored in rectangular form, but you can enter a complex number in rectangular form or polar form, regardless of the mode setting. The components of complex numbers can be real numbers or expressions that evaluate to real numbers; expressions are evaluated when the command is executed.

## Note about Radian Versus Degree Mode

Radian mode is recommended for complex number calculations. Internally, the TI-83 Plus converts all entered trigonometric values to radians, but it does not convert values for exponential, logarithmic, or hyperbolic functions.

In degree mode, complex identities such as $e^\wedge(i\theta) = \cos(\theta) + i\sin(\theta)$ are not generally true because the values for cos and sin are converted to radians, while those for e^() are not. For example, $e^\wedge(i45) = \cos(45) + i\sin(45)$ is treated internally as $e^\wedge(i45) = \cos(\pi/4) + i\sin(\pi/4)$. Complex identities are always true in radian mode.

## Interpreting Complex Results

Complex numbers in results, including list elements, are displayed in either rectangular or polar form, as specified by the mode setting or by a display conversion instruction. In the example below, polar-complex (**re^**θ*i*) and **Radian** modes are set.

```
(2+i)-(1e^(π/4i)
>
1.325654296e^(...
```

## Rectangular-Complex Mode

Rectangular-complex mode recognizes and displays a complex number in the form **a+b***i*, where **a** is the real component, **b** is the imaginary component, and *i* is a constant equal to $\sqrt{-1}$.

```
ln(-1)
      3.141592654i
```

To enter a complex number in rectangular form, enter the value of *a* (*real component*), press $\boxed{+}$ or $\boxed{-}$, enter the value of *b* (*imaginary component*), and press $\boxed{2nd}$ $[i]$ (constant).

*real component*(**+** or **-**)*imaginary component **i***

```
4+2i
           4+2i
```

## Polar-Complex Mode

Polar-complex mode recognizes and displays a complex number in the
form *re^θi*, where *r* is the magnitude, *e* is the base of the natural log, *θ* is the
angle, and *i* is a constant equal to $\sqrt{-1}$.

```
ln(-1)
3.141592654e^(1...
```

To enter a complex number in polar form, enter the value of *r* (*magnitude*),
press [2nd] [e^x] (exponential function), enter the value of *θ* (*angle*), press
[2nd] [*i*] (constant), and then press [)].

*magnitude***e^(***angle**i***)**

```
10e^(π/3i)
10e^(1.04719755...
```

# MATH CPX (Complex) Operations

## MATH CPX Menu

To display the **MATH CPX** menu, press $\boxed{\text{MATH}}$ $\boxed{\blacktriangleright}$ $\boxed{\blacktriangleright}$.

| MATH NUM CPX PRB | |
|---|---|
| **1:** conj( | Returns the complex conjugate. |
| 2: real( | Returns the real part. |
| 3: imag( | Returns the imaginary part. |
| 4: angle( | Returns the polar angle. |
| 5: abs( | Returns the magnitude (modulus). |
| 6: ▶Rect | Displays the result in rectangular form. |
| 7: ▶Polar | Displays the result in polar form. |

## conj(

**conj(** (conjugate) returns the complex conjugate of a complex number or list of complex numbers.

**conj(***a***+***b**i***)** returns $a-bi$ in **a+b***i* mode.
**conj(***r***e^(***θi***))** returns $r$**e^(**-$θi$**)** in **re^**θ*i* mode.

```
conj(3+4i)
            3-4i
```

```
conj(3e^(4i))
3e^(2.283185307…
```

## real(

**real(** (real part) returns the real part of a complex number or list of complex numbers.

**real(***a***+***b***i)** returns *a*.
**real(***r***e^(***θ***i))** returns *r**cos*(*θ*).

```
real(3+4i)
              3
```
```
real(3e^(4i))
     -1.960930863
```

## imag(

**imag(** (imaginary part) returns the imaginary (nonreal) part of a complex number or list of complex numbers.

**imag(***a***+***b***i)** returns *b*.
**imag(***r***e^(***θ***i))** returns *r**sin*(*θ*).

```
imag(3+4i)
              4
```
```
imag(3e^(4i))
     -2.270407486
```

## angle(

**angle(** returns the polar angle of a complex number or list of complex numbers, calculated as $\tan^{-1}(b/a)$, where b is the imaginary part and a is the real part. The calculation is adjusted by $+\pi$ in the second quadrant or $-\pi$ in the third quadrant.

**angle(**$a$**+**$bi$**)** returns $tan^{-1}(b/a)$.
**angle(**$r$**e^(**$\theta i$**))** returns $\theta$, where $-\pi < \theta < \pi$.

```
angle(3+4i)
        .927295218
```
```
angle(3e^(4i))
        -2.283185307
```

## abs(

**abs(** (absolute value) returns the magnitude (modulus), $\sqrt{(real2+imag2)}$, of a complex number or list of complex numbers.

**abs(**$a$**+**$bi$**)** returns $\sqrt{(a2+b2)}$.
**abs(**$r$**e^(**$\theta i$**))** returns $r$ (magnitude).

```
abs(3+4i)
             5
```
```
abs(3e^(4i))
             3
```

### ▶Rect

▶**Rect** (display as rectangular) displays a complex result in rectangular form. It is valid only at the end of an expression. It is not valid if the result is real.

*complex result*▶**Rect** returns *a*+*bi*.

```
√(-2)▶Rect
        1.414213562i
```

### ▶Polar

▶**Polar** (display as polar) displays a complex result in polar form. It is valid only at the end of an expression. It is not valid if the result is real.

*complex result*▶**Polar** returns *r***e^(***θi***)**.

```
√(-2)▶Polar
1.414213562e^(1…
```

# MATH PRB (Probability) Operations

## MATH PRB Menu

To display the **MATH PRB** menu, press MATH ◄.

| MATH NUM CPX PRB | |
|---|---|
| 1: rand | Random-number generator |
| 2: nPr | Number of permutations |
| 3: nCr | Number of combinations |
| 4: ! | Factorial |
| 5: randInt( | Random-integer generator |
| 6: randNorm( | Random # from Normal distribution |
| 7: randBin( | Random # from Binomial distribution |

## rand

**rand** (random number) generates and returns one or more random numbers > 0 and < 1. To generate a list of random-numbers, specify an integer > 1 for *numtrials* (number of trials). The default for *numtrials* is 1.

**rand[(***numtrials***)]**

**Tip:** To generate random numbers beyond the range of 0 to 1, you can include **rand** in an expression. For example, **rand5** generates a random number > 0 and < 5.

With each **rand** execution, the TI-83 Plus generates the same random-number sequence for a given seed value. The TI-83 Plus factory-set seed value for **rand** is **0**. To generate a different random-number sequence, store any nonzero seed value to **rand**. To restore the factory-set seed value, store **0** to **rand** or reset the defaults (Chapter 18).

**Note:** The seed value also affects **randInt(**, **randNorm(**, and **randBin(** instructions.

```
rand
        .1272157551
        .2646513087
1→rand
                    1
rand(3)
{.7455607728 .8…
```

### nPr, nCr

**nPr** (number of permutations) returns the number of permutations of *items* taken *number* at a time. *items* and *number* must be nonnegative integers. Both *items* and *number* can be lists.

*items* **nPr** *number*

**nCr** (number of combinations) returns the number of combinations of *items* taken *number* at a time. *items* and *number* must be nonnegative integers. Both *items* and *number* can be lists.

*items* **nCr** *number*

```
5 nPr 2
              20
5 nCr 2
              10
{2,3} nPr {2,2}
              {2 6}
```

## ! (Factorial)

**!** (factorial) returns the factorial of either an integer or a multiple of .5. For a list, it returns factorials for each integer or multiple of .5. *value* must be ≥ˉ.5 and ≤69.

*value***!**

```
6!
              720
{5,4,6}!
     {120 24 720}
```

**Note:** The factorial is computed recursively using the relationship (n+1)! = n**∗**n!, until n is reduced to either 0 or ˉ1/2. At that point, the definition 0!=1 or the definition (ˉ1/2)!=√π is used to complete the calculation. Hence:

n!=n**∗**(n−1)**∗**(n−2)**∗** ... **∗**2**∗**1, if n is an integer ≥0
n!= n**∗**(n−1)**∗**(n−2)**∗** ... **∗**1/2**∗**√π, if n+1/2 is an integer ≥0
n! is an error, if neither n nor n+1/2 is an integer ≥0.

(The variable n equals *value* in the syntax description above.)

## randInt(

**randInt(** (random integer) generates and displays a random integer within a range specified by *lower* and *upper* integer bounds. To generate a list of random numbers, specify an integer >1 for *numtrials* (number of trials); if not specified, the default is 1.

**randInt(***lower***,***upper*[**,***numtrials*]**)**

```
randInt(1,6)+ran
dInt(1,6)
                6
randInt(1,6,3)
          {2 1 5}
```

## randNorm(

**randNorm(** (random Normal) generates and displays a random real number from a specified Normal distribution. Each generated value could be any real number, but most will be within the interval [$\mu-3(\sigma)$, $\mu+3(\sigma)$]. To generate a list of random numbers, specify an integer > 1 for *numtrials* (number of trials); if not specified, the default is 1.

**randNorm(***μ***,***σ*[**,***numtrials*]**)**

```
randNorm(0,1)
      .0772076175
randNorm(35,2,10
0)
{34.02701938 37…
```

## randBin(

**randBin(** (random Binomial) generates and displays a random integer from a specified Binomial distribution. *numtrials* (number of trials) must be ≥ 1. *prob* (probability of success) must be ≥ 0 and ≤ 1. To generate a list of random numbers, specify an integer > 1 for *numsimulations* (number of simulations); if not specified, the default is 1.

**randBin(***numtrials***,***prob*[**,***numsimulations*]**)**

```
randBin(5,.2)
                    3
randBin(7,.4,10)
{3 3 2 5 1 2 2 …
```

**Note:** The seed value stored to **rand** also affects **randInt(**, **randNorm(**, and **randBin(** instructions.

# ANGLE Operations

## ANGLE Menu

To display the **ANGLE** menu, press [2nd] [ANGLE]. The **ANGLE** menu displays angle indicators and instructions. The **Radian/Degree** mode setting affects the TI-83 Plus's interpretation of **ANGLE** menu entries.

| ANGLE | |
|---|---|
| 1: ° | Degree notation |
| 2: ' | DMS minute notation |
| 3: ʳ | Radian notation |
| 4: ►DMS | Displays as degree/minute/second |
| 5: R►Pr( | Returns **r**, given **X** and **Y** |
| 6: R►Pθ( | Returns θ, given **X** and **Y** |
| 7: P►Rx( | Returns **x**, given **R** and θ |
| 8: P►Ry( | Returns **y**, given **R** and θ |

## Entry Notation

DMS (degrees/minutes/seconds) entry notation comprises the degree symbol (°), the minute symbol (**'**), and the second symbol (**"**). *degrees* must be a real number; *minutes* and *seconds* must be real numbers ≥ 0.

*degrees*°*minutes*'*seconds*"

For example, enter for 30 degrees, 1 minute, 23 seconds. If the angle mode is not set to **Degree**, you must use ° so that the TI-83 Plus can interpret the argument as degrees, minutes, and seconds.

**Degree** mode

```
sin(30°1'23")
        .5003484441
```

**Radian** mode

```
sin(30°1'23")
        -.9842129995
sin(30°1'23"°)
        .5003484441
```

## ° (Degree)

° (degree) designates an angle or list of angles as degrees, regardless of the current angle mode setting. In **Radian** mode, you can use ° to convert degrees to radians.

*value*°
**{**value1**,**value2**,**value3**,**value4**,…,**value n**}**°

° also designates *degrees* (D) in DMS format.
**'** (minutes) designates *minutes* (M) in DMS format.
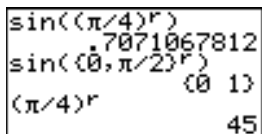**"** (seconds) designates *seconds* (S) in DMS format.

**Note:** " is not on the **ANGLE** menu. To enter ", press [ALPHA] [''].

## ʳ (Radians)

ʳ (radians) designates an angle or list of angles as radians, regardless of the current angle mode setting. In **Degree** mode, you can use ʳ to convert radians to degrees.
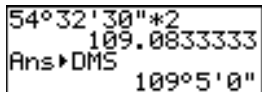
*value*ʳ

**Degree** mode

```
sin((π/4)ʳ)
          .7071067812
sin({0,π/2}ʳ)
               {0 1}
(π/4)ʳ
                  45
```

## ▶DMS

▶**DMS** (degree/minute/second) displays *answer* in DMS format. The mode setting must be **Degree** for *answer* to be interpreted as degrees, minutes, and seconds. ▶**DMS** is valid only at the end of a line.

*answer*▶**DMS**

```
54°32'30"*2
          109.0833333
Ans▶DMS
          109°5'0"
```

## R►Pr(, R►Pθ(, P►Rx(, P►Ry(

**R►Pr(** converts rectangular coordinates to polar coordinates and returns **r**. **R►Pθ(** converts rectangular coordinates to polar coordinates and returns θ. *x* and *y* can be lists.

**R►Pr(***x*,*y***), R►Pθ(***x*,*y***)**

```
R►Pr(-1,0)
            1
R►Pθ(-1,0)
   3.141592654
```

**Note: Radian** mode is set.

**P►Rx(** converts polar coordinates to rectangular coordinates and returns **x**. **P►Ry(** converts polar coordinates to rectangular coordinates and returns **y**. *r* and *θ* can be lists.

**P►Rx(***r*,*θ***), P►Ry(***r*,*θ***)**

```
P►Rx(1,π)
           -1
P►Ry(1,π)
            0
```

**Note: Radian** mode is set.

# TEST (Relational) Operations

## TEST Menu

To display the **TEST** menu, press [2nd] [TEST].

| This operator... | Returns 1 (true) if... |
|---|---|
| TEST    LOGIC | |
| 1: **=** | Equal |
| 2: ≠ | Not equal to |
| 3: **>** | Greater than |
| 4: ≥ | Greater than or equal to |
| 5: **<** | Less than |
| 6: ≤ | Less than or equal to |

### **=,** ≠**, >,** ≥**, <,** ≤

Relational operators compare *valueA* and *valueB* and return **1** if the test is true or **0** if the test is false. *valueA* and *valueB* can be real numbers, expressions, or lists. For **=** and ≠ only, *valueA* and *valueB* also can be matrices or complex numbers. If *valueA* and *valueB* are matrices, both must have the same dimensions.

Relational operators are often used in programs to control program flow and in graphing to control the graph of a function over specific values.

| | |
|---|---|
| *valueA***=***valueB* | *valueA*≠*valueB* |
| *valueA***>***valueB* | *valueA*≥*valueB* |
| *valueA***<***valueB* | *valueA*≤*valueB* |

```
25=26
              0
{1,2,3}<3
          {1 1 0}
{1,2,3}≠{3,2,1}
          {1 0 1}
```

## Using Tests

Relational operators are evaluated after mathematical functions according to EOS rules (Chapter 1).

- The expression **2+2=2+3** returns **0**. The TI-83 Plus performs the addition first because of EOS rules, and then it compares 4 to 5.

- The expression **2+(2=2)+3** returns **6**. The TI-83 Plus performs the relational test first because it is in parentheses, and then it adds 2, 1, and 3.

# TEST LOGIC (Boolean) Operations

## TEST LOGIC Menu

To display the **TEST LOGIC** menu, press [2nd] [TEST] [▶].

| This operator... | Returns a 1 (true) if... |
| --- | --- |
| TEST **LOGIC** | |
| **1:** and | Both values are nonzero (true). |
| 2: or | At least one value is nonzero (true). |
| 3: xor | Only one value is zero (false). |
| 4: not( | The value is zero (false). |

## Boolean Operators

Boolean operators are often used in programs to control program flow and in graphing to control the graph of the function over specific values. Values are interpreted as zero (false) or nonzero (true).

## and, or, xor

**and**, **or**, and **xor** (exclusive or) return a value of **1** if an expression is true or **0** if an expression is false, according to the table below. *valueA* and *valueB* can be real numbers, expressions, or lists.

*valueA* **and** *valueB*
*valueA* **or** *valueB*
*valueA* **xor** *valueB*

| valueA | valueB | | and | or | xor |
|--------|--------|---------|-----|-----|-----|
| ≠0 | ≠0 | returns | 1 | 1 | 0 |
| ≠0 | 0 | returns | 0 | 1 | 1 |
| 0 | ≠0 | returns | 0 | 1 | 1 |
| 0 | 0 | returns | 0 | 0 | 0 |

## not(

**not(** returns **1** if *value* (which can be an expression) is **0**.

**not(***value***)**

## Using Boolean Operations

Boolean logic is often used with relational tests. In the following program, the instructions store **4** into **C**.

```
PROGRAM:BOOLEAN
:2→A:3→B
:If A=2 and B=3
:Then:4→C
:Else:5→C
:End
```