By default, Scilab accepts complex numbers only in rectangular form. Use \%i to represent the complex number \(i\).

\[
\text{-->5+4*\%i}\\
\text{ans =}\\
5. + 4.\%i
\]

You can define two additional functions, \text{p2z()} and \text{z2p()}, that make it easy to enter and display numbers in polar form. The code for these functions is included on the last page of this tutorial. You can also find the code online at http://csserver.evansville.edu/~richardson/. Follow the installation instructions in the README.TXT file.

To enter the complex number \((1\angle45^\circ)\) using the \text{p2z()} function, just enter:

\[
\text{-->} \quad x = \text{p2z}(1, 45)\\
x = 0.7071068 + 0.7071068\%i
\]

To display a complex number in polar form use the \text{z2p()} function:

\[
\text{-->} z2p(x)\\
\text{! 1. 45. !}\\
\text{ans =}\\
0.7071068 + 0.7071068\%i
\]

Note that the magnitude is displayed first and that the phase angle is in degrees. The \text{z2p()} function just displays the number in polar form. It returns the complex number in standard rectangular form.

You can use the \text{p2z()} function to enter a complex number in polar form at any point you might use a number in rectangular form. Here are some examples:

\[
\text{-->} z = (7+8*\%i + \text{p2z}(20, -30))/\text{p2z}(5, 45)\\
z = 3.1565965 - 3.7222819\%i
\]

\[
\text{-->} z2p(z);\\
\text{! 4.8805209 - 49.701147 !}
\]

\[
\text{-->} \text{volts} = [\text{p2z}(10, 45); \text{p2z}(20, -30); \text{p2z}(100, 0)]\\
\text{volts =}\\
\text{! 7.0710678 + 7.0710678\%i !}\\
\text{! 17.320508 -10.\%i !}\\
\text{! 100. !}
\]

The first two results indicate that \(z\) is equal to \(3.1566 - 3.72228\%i\) or \((4.8805 \angle -49.70^\circ)\). The last example illustrates how a voltage column array can be defined. \text{p2z()} will also accept matrix arguments. The first argument should contain the
magnitudes and the second argument should contain the phase angles, the volts array in the previous example could be defined as:

```scilab
--> volts = p2z([10 20 100]',[45 -30 0]')
volts =
!  7.0710678 + 7.0710678i !
! 17.320508 -10.i        !
!  100.                   !
```

`z2p()` will accept an array of complex numbers. It displays an array of magnitude and phase pairs:

```scilab
--> z2p(volts);
! 10.     45. !
! 20.   -30. !
! 100.    0. !
```

By default, `z2p()` returns the original complex matrix in standard rectangular form. It only displays the values in polar form. However, if two return arguments are used the magnitude and phase angles are returned in separate matrices:

```scilab
--> [m a] = z2p(volts)
a =
! 45. !
! -30. !
!  0. !
m =
! 10. !
! 20. !
! 100. !
```

Be careful when using the apostrophe (') operator with complex arrays and matrices. It returns the complex-conjugate transpose of a complex array instead of just the transpose of the array or matrix. If you want the transpose be sure to use the dot-apostrophe ('). operator. For a real array the apostrophe and dot-apostrophe operators give identical results. When applied to a single complex number (instead of an array) the apostrophe operator gives the complex-conjugate of the number.

Here's a final example that illustrates how to use Scilab to solve a sinusoidal steady-state circuit problem. Suppose we want to find the branch phasor currents in the phasor domain circuit shown in Figure 1.

![Figure 1: Example Frequency Domain Circuit](image-url)

Here's a final example that illustrates how to use Scilab to solve a sinusoidal steady-state circuit problem. Suppose we want to find the branch phasor currents in the phasor domain circuit shown in Figure 1.
We will use mesh analysis to solve this problem. We can write the following set of mesh equations by inspection:

\[
\begin{align*}
100 \angle -90 &= j12(I_A-I_C)+80(I_A-I_B) \\
-500 \angle 0 &= 80(I_B-I_A)+20(I_B-I_C) \\
0 &= -j20I_c+20(I_c-I_B)+j12(I_c-I_A)
\end{align*}
\]

Collecting terms corresponding to each phasor current yields:

\[
\begin{align*}
100 \angle -90 &= (j12+80) I_A + (-80) I_B + (-j12) I_C \\
-500 \angle 0 &= (-80) I_A + (80+20) I_B + (-20) I_C \\
0 &= (-j12) I_A + (-20) I_B + (-j20+20+j12) I_C
\end{align*}
\]

Instead of simplifying these equations, let's go directly to Scilab. We can define the voltage array as:

```scilab
-->V = [p2z(100, -90); p2z(-500, 0); 0];
```

The `p2z()` function was used to allow us to easily enter in the voltage array using magnitude and phase angle values.

The elements of Z are formed directly from the mesh equations above (let Scilab do the work of combining the terms for you):

```scilab
-->Z = [%i*12+80,   -80,          -%i*12  \\
     -80, 80+20,             -20  \\
    -%i*12,   -20, -%i*20+20+%i*12];
```

Although not necessary here, the `p2z()` function could have been used to allow us to easily enter in any values that were in polar form.

Since \( V = ZI \), then \( I = Z^{-1}V \). Computing the currents and then using `z2p()` to display the currents in polar form yields:

```scilab
-->I = inv(Z)*V;
-->z2p(I);

! 22.022716  - 129.47246 !
! 24.020824  - 129.25584 !
! 25.495098  - 78.690068 !
```

The currents are therefore equal to \( I_A = (22.0 \angle -129.5^\circ) \), \( I_B = (24.0 \angle -129.3^\circ) \), and \( I_C = (25.5 \angle -78.7^\circ) \).
Scilab Complex Number Routines

Note: The versions of these routines that are available for download have more comments, better error checking, and support additional options. Only the core code is shown here to limit the size of this document.

// A = p2z(R,Theta) - Convert from polar to rectangular form.  
//    R is a matrix containing the magnitudes 
//    Theta is a matrix containing the phase angles (in degrees).
function [A] = p2z(R,Theta)
    if argn(2) <> 2 then
        error("incorrect number of arguments.");
    end
    if ~and(size(R) == size(Theta)) then
        error("arguments must be of the same dimension.");
    end
    A = R.*exp(%i*%pi*Theta/180.);
endfunction

// [R, Theta] = z2p(A) - Display polar form of complex matrix.
function [R, Theta] = z2p(A)
    if argn(1) == 1 then
        // Only a single return argument.  
        // Display the mag and phase in adjacent columns.
        Theta = zeros(A);
        // Protect against 0 entries
        i = (A <> 0);
        Theta(i) = atan(imag(A(i)),real(A(i)))*180./%pi;
        disp(matrix([abs(A); Theta],size(A,1),2*size(A,2)));
        R = A;
    elseif argn(1) == 2 then
        // With two return arguments, the mag and phase are in stored in 
        // separate matrices
        // Protect against 0 entries
        i = (A <> 0);
        Theta = zeros(A);
        Theta(i) = atan(imag(A(i)),real(A(i)))*180./%pi;
        R = abs(A);
    else
        error("invalid number of return arguments");
    end
endfunction