A Brief **Shang** Tutorial

1 Using Shang as a calculator

1.1 Doing Arithemetic

If an arithmetic expression is entered at command prompt the interpreter will evaluate it and display the answer.

```
>> (75.00 + 89.50 + 97.50) / 3 * 0.7 + 92.00 * 0.3
    88.73333333
```

The expression may contain numbers and operators +, -, *, and / that represent addition, subtraction, multiplication, and division, and parentheses can be used for grouping.

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^`	`a^b` is equal to a^b
`()`	grouping

Finding the square root of $6^2 + 8^2$

```
>> (6^2 + 8^2)^(1/2)
    10
```

Numbers can also be enter using *exponential format*. For example, `-1.1e-3` is the same as `-0.0011`, and `1e+6` or `1e6` is another way of writing `1000000`, a million.

1.2 Common Functions

Many common elementary functions such as `sqrt, exp, log, sin, cos, tan, asin, acos`, and `atan` are built in the interpreter can be used in the expressions directly. For example, `sqrt` is the name of square root function. To find the square root of $6^2 + 8^2$

```
>> sqrt(6^2 + 8^2)
    10
```

The built-in name for *exponential function* e^x is `exp`. To evaluate $e^{-0.5}$ we should write `exp(-0.5)` instead of `e^(-0.5)`. Although `e^(-0.5)` would in theory give the same answer, it's computed differently and might be less accurate. `log` is the name of the *natural logarithm* function $\ln x$.

1.3 Common Constants

The mathematical constants e, π and imaginary unit i are represented by `e`, `pi`, and `i` respectively, and can be used directly. For example:

```
>> cos(pi)
    -1
>> sin( pi)
   1.224606354e-16
```

Note that the exact answer to $\sin(\pi)$ is 0 and what we are given here is only an approximate answer since π is just an approximate representation of π, and the sin function uses numerical method.

1.4 Arbitrary Precision Computation

If a number has the suffix `M` it is treated as an arbitrary precision floating point number, which by default has about 38 significant decimal digits. For example

```
>> sqrt(3M)
   1.7320508075688772935274463415058 7E0
>> x = 3.14159265358979323846264338327950M
     3.14159265358979323846264338327950E0
>> sqrt(x)
     1.77245385090551602729816748334114E0
```

The precision of MPF can be rest by assigning a multiple to global variable `global.mpf_ndigits`. The default value of `mpf_ndigits` is 128, therefore an `mpf` may have 128 significant binary digits as opposed to 52 for `double`. The value of `mpf_ndigits`. can be set to a multiple (at least 2) of 32.
If a number has the suffix `L` it is treated as a long integer, which can be as big as the computer memory allows. For example

```
>> 128L ^ 20L
     1393796574908163946345982392040522594123776
```

1.5 Complex Numbers

Complex numbers are supported interally. A complex number of real and imaginary parts `a` and `b` is displayed as `a+bi`, and can be entered as either `a+bi, a+bI, a+bj`, or `a+bJ`. For example

```
>> sqrt(-5)
    0 + 2.236067977i
>> (3+5i) / (2-3j)
    -0.6923076923 + 1.461538462i
```

1.6 Define Variables

Variables can be defined using the assignment operator `=`. For example

```
>> h = 1.25
```

creates a variable named "`h`" and assign value 1.25 to it. If there is already a variable named "`h`", then its value will be updated to 1.25.
The interpreter remembers the values of the variables, so that you can recall the names of the variables to use their values. For example:

```
>> c = 10;
>> b = 8;
>> sqrt(c^2 - b^2)
    6
```

A variable name can be a string of letters and digits, and underscore _, but cannot begin with a digit. For example, `x1`, `value_1`, and `_val03` are all valid variable names.

1.7 Unwanted Printing

Usually the result of a command is displayed after `Enter` is hit.

```
>> h = 1.25
   1.25
```

To prevent this, a semicolon can be appended. By suppressing unwanted displays, the command window can be kept cleaner and the program may run more efficiently.

```
>> h = 1.25;
>> // nothing is displayed here
```

1.8 Command Editing and History

By using the up and down arrow keys on the keypad, the previously entered commands can be brought up for editing and entered again.

1.9 Two ways to enter commands

(a) Type in

You can always type the commands directly in the command window and see the answers right away.

(b) Edit and paste

Alternatively, you can edit all the commands using a text editor and when you are done, copy the commands to clipboard and then paste it to the Shang interpreter window. This can be more efficient since whenever anything goes wrong, you just need to correct the commands in the text editor and paste them again and don't have to retype everything.

1.10 Submit a Command and Abondon a Command

If the cursor position is at the end of the command, pressing enter will submit the command for execution.
When you press enter, if the interpreter doesn't show answer, but keep opening new lines, it thinks the input commands are not complete. It is expecting some closing brackets or quotes. If you want to start over and abondon the current commands, use the `ctrl-a` command.

1.11 Run a script

A script file is a text file that contains a sequence of Shang commands. For example, a file with name "testscript.txt" may contain the following lines

```
f = (h, r, theta) -> r^2 + h * (1 + cos(theta));
h = 3.5;
r = 5;
theta = pi / 2;
f(h, r, theta)
```

In the interactive mode, if the following command is issued

```
>> run("testscript.txt");
   28.5
>>
```

All the commands in the script file will be executed.

1.12 Commenting

It's always a good idea to use comments to annotate your commands and programs.

1.12.1 Single line comment

All the characters in a line following the symbol `//` are ignored, and therefore can act as comments. For example

```
>> // this is a comment
>> sqrt(-3)   // this is a comment as well
```

1.12.2 Multi-line comment

Comments can run multilines as well. Any characters between a pair of `/*` and `*/` are ignored.

```
>> /* this is a comment
      this is still a comment
this is  still a comment
      ...
      we're done commenting (finally) */
```

Shang interpreter can recognize five levels of nested comments.

2 Flow Control

2.1 Conditionals

If we want to assign `c` to `b` when the condition `f(a) * f(c) < 0` is true:

```
if f(a) * f(c) < 0
       b = c;
end
```

If we want to assign `c` to `b` when the condition `f(a) * f(c) < 0` is true, and assign `c` to `a` when the condition is false:

```
if f(a) * f(c) < 0
       b = c;
else
       a = c;
end
```

To test the condition "`n` equals to `1`", we need to use two equal signs, like `n == 1`, not `n = 1`. Note that a single `=` is the assignment operator. The statement `n = 1` would assign `1` to `n` instead of testing the condition. The following assigns `c` to `b` when `n` equals 1, and assigns `c` to `a` when `n` equals to 2:

```
if n == 1
       b = c;
elseif n == 2
       a = c;
end
```

2.2 Loops

If we want to repeatedly execute a bunch of commands for a million times, we can use the `for` loop

```
s = 0;  // initialize s
for k = 1 : 1000000
    s += k;      // add k to s --- equivalent to "s = s + k"
end
```

The above loop compute the sum

$$\sum_{k=1}^{1000000} k$$

The `while` loop enables us to repeatedly execute a bunch of commands as long as a condtion is satisfied

```
s = 0;
k = 1;
while k <= 1000000   // do the following as long k is less than a million
    if k % 2 != 0    // k % 2 is the remainder of k/2
       s += k;
    end
    ++k;   // increment k, same as k = k + 1;
           // without the increment, k will stays as 1 and loop runs forever
end
```

The above `while` loop sums up all the odd numbers between 1 and a million; it is equivalent to the following `for` loop

```
s = 0;
for k = 1 : 2 : 1000000
    s += k;
end
```

3 Define Functions

3.1 One Liner Functions

To define a simple one-liner function, one can use the arrow to join the input argument and the output value. For example, function $f(x) = \frac{1-x}{1+x+x^2}$ can be defined by

```
f = x -> (1 - x) / (1 + x + x^2)
```

A defined function can be called in the obvious way

```
f = x -> (1 - x) / (1 + x + x^2)  // define a function
f(-1)  // call the function
```

A function can have several input arguments. For example

$$f(r, \theta) = r(1 + \cos(\theta))$$

can be defined by

```
 f = (r, theta) -> r * (1 + cos(theta));  // define
```

3.2 More Complicated Functions

If a function definition has more than one line of statements, the keyword `function` should be used. The syntax is

```
function_name = function (input_arguments) -> (output arguments)
                    ... // a bunch of statements
                end
```

Note that in the body of the function, the input arguments can be used as if they are variables (but they are not visible outside the function), and the output arguments should be given values before the end of the function. The following function evaluates the Taylor expansion of $\cos(x)$ function for a small x value

$$\cos x \approx \sum_{n=0}^{N} (-1)^n \frac{x^{2n}}{(2n)!}$$

```
cos_taylor = function (x, N) -> v
       v = 1;
       vk = 1;
       xsq = x * x;
       for k = 1 : N
           vk = - vk * xsq / (2 * k) / (2 * k - 1);
           v += vk;
       end
end
```

4 Matrices

A matrix is a rectangular array of numbers. it can be created with elements included in a pair of square brackets. The rows are separated by semicolons, while elements in the same row are separated by commas.

```
>> A = [1,4, 9; 2, 3, 5; -2, 5, 10]
   1   4    9
   2   3    5
  -2   5   10
```

To refer to the element of matrix `A` at second row and third column, use `A[2,3]`

```
>> A[2,3]
   5
```

Create Matrices

Alternatively, a matrix of a required size can be created and initialized using built-in functions `zeros`, `ones`, or `rand`. The command

```
A = zeros(3, 5)
```

will return a matrix of three rows and five columns, with each element being zero. Similar usage of `ones` and `rand` will create matrices of 1's and random numbers (between 0 and 1) respectively. These three functions can also be called with a single parameter, in which case the second parameter is assumed to be 1, and thus a column vector is created.

```
>> B = rand(5)
    0.289
    0.353
    0.154
    0.566
    0.821
```

By the `dimension` of a matrix we refer to the number of rows and the number of columns. For example, the dimension of the scalar `-5` is 1×1, while the dimension of the matrix

```
   -2    3    9
    10   1    -2
```

is 2×3.

Create Even Spaced Vectors Using the Colon Operator

The symbol `:` can be used to create a row matrix whose elements are evenly spaced. The default step-size of the vector is 1, which is assumed when one colon is used.

```
>> A = -1 : 5
   -1  0  1  2  3   4   5
```

To specify a step-size other than 1, two colons are needed.

```
>> A = 3 : 0.5 : 5
   3  3.5  4  4.5  5
```

When you use `A : B` to create a vector, the value `B` is not always included in the vector. Alternatively, the built-in function `linspace` can be used to created even spaced vector with the specified end points included. `linspacea, b, n` will create a column vector of length `n`, with `a` and `b` being the two end points.

5 Matrix Indexing

5.1 Single Index

If a matrix `A` is a row vector or a column vector, then `A[k]` refers to the kth element of `A`. The index `k` is a number between 1 and the length of the vector. Note that 0 is not a valid index value.

```
>>  A = [3, 5, -2, 7, 0.9];
>>  A[3]
    -2
```

The index itself can be vector such that a bunch of the elements of `A` are referenced.

```
>>  A = [3, 5, -2, 7, 0.9];
>>  A[[1, 3, 5]]
    3  -2   0.9
>> A[1:3]
    3   5    -2
>> A[1 : 2 : 5]   // 1 : 2 : 5 is the same as [1, 3, 5]
    3  -2   0.9
```

The dollar sign `$` when used as an index, is the largest value of the index, therefore `A[$]` is the last element of `A`.

```
>>  A = [3, 5, -2, 7, 0.9];
>>  A[$]
    0.9
>>  A[3:$]
    -2  7  0.9
```

If matrix `A` is not a vector, then the element referred to by `A[k]` is the `k`-th element of the row vector obtained by horizontally joining all the rows of `A`.

```
>> A = [1,4, 9; 2, 3, 5; -2, 5, 10]
   1   4    9
   2   3    5
  -2   5   10
>> A[5]
   3
>> A[9]
   10
```

5.2 Two Indices Separated by a Comma

`A[i, j]` is the element of `A` at `i`-th row and `j`-th column. Both `i` and `j` (or their elements) must be positive integers. Note that both indices can be vectors

```
>> A = [8, 1, 6; 3, 5, 7; 4, 9, 2]
   8    1    6
   3    5    7
   4    9    2
>> A[2,3]
```

```
   7
>> A[1, 1 : 3]   // the first row
   8    1    6
>> A[1 : 3, 2]   // the second column
   1
   5
   9
>> A[[2, 3], [2, 3]]  // the 2x2 submatrix on the lower right corner
   5    7
   9    2
```

5.3 The colon : alone as an index

If an index consists of a single colon, then it is equivalent to `1 : $`. For example, `A[1, :]` returns the first row, `A[:, 3]` returns the third column.

```
>> A = [8, 1, 6; 3, 5, 7; 4, 9, 2]
   8    1    6
   3    5    7
   4    9    2
>> A[1, :]   // the first row
   8    1    6
>> A[:, 2]   // the second column
   1
   5
   9
>> A[1 : 2, :]  // the first two rows
   8    1    6
   3    5    7
```

5.4 The colon : alone as an index

If an index consists of a single colon, then it is equivalent to `1 : $`. For example, `A[1, :]` returns the first row, `A[:, 3]` returns the third column.

```
>> A = [8, 1, 6; 3, 5, 7; 4, 9, 2]
   8    1    6
   3    5    7
   4    9    2
>> A[1, :]   // the first row
   8    1    6
>> A[:, 2]   // the second column
   1
```

```
   5
   9
>> A[1 : 2, :]  // the first two rows
   8    1    6
   3    5    7
```

5.5 Change the Elements of a Matrix

Any of the indexing expression can be used to modify part of the elements of a matrix. For example, `A[2, 3]=-1` would change the element at row `2` and column `3` to -1.

```
>> A = zeros(3,5)
   0   0   0   0   0
   0   0   0   0   0
>> A[2,3]=2.3
   0   0    0    0   0
   0   0  2.3    0   0
>> A[:, 1] = [5; 10]
    5   0    0    0   0
   10   0  2.3    0   0
>> A[:, 3] = A[:, 3}  + A[:, 1]
    5   0    5    0   0
   10   0 12.3    0   0
>> A[2, :] = 9
    5   0    5    0   0
    9   9    9    9   9
```

Usually when an indexing expression is used as an lvalue, the dimension and size of the right hand side of the assignment should match that of the indexing expression to make the assignment possible. The only exception is when the right hand side is a scalar, then all the indexed elements of the matrix will be set to the same value.

```
>> A = zeros(3,3)
   0   0   0
   0   0   0
   0   0   0
>> A[\] = 1;
   1   0   0
   0   1   0
   0   0   1
```

When updating the contents of a matrix, the indices don't have to be within the upper bounds, which makes it possible to make the size of the matrix grow. As the size of a matrix is being

extended, the new entries are set to zero, except for those being specified by the assignment statement. For example,

```
>> X = [1,4,9]
   1   4   9
>> X[4] = 16
   1   4   9   16
>> x[8]=36
   1   4   9    16    0    0   0   36
```

6 Matrix Operation

6.1 Arithmetic

If `A` and `B` are two matrices, then `A+B`, `A-B`, and `A * B` are the sum, difference, and product of `A` and `B`.
`A+B`, `A-B` are possible only when `A` and `B` have the same dimensions, or one of them is a scalar.
`A*B` is possible only when the number of columns of `A` equals the number of rows of `B`, or one of them is a scalar.

6.2 Solving Linear System: \

If both `A` and `B` are matrices, then `A \ B` is the (numerical) solution of matrix equation `A X = B`. The matrix `A` must be an `n` × `n` non-singular square matrix and `B` must have `n` rows. For example

```
>>  A = [1, 2, -3; 2, 1, 5; 0, -1, 5];
>>  b = [3; 1; -2];
>>  x = A \ b
     0.75
     0.75
    -0.25

>>  A * x - b
    0
    0
    0
```

7 Graphics

Shang has no built-in functionalities for handling graphics. However there is a package of programs written in the Shang language which can create 2D and 3D plots. The package saves pictures as

encapuslated postscript files. In order to print or display the images created by plot.x, you need to download and install the Ghostscript and GSview programs at

`http://pages.cs.wisc.edu/~ghost/`

To use this package, you need to run the file `plot.x`, which is located in the `programs` directory. Or just run the following command before you plot

```
include("plot.x");
```

The package `plot.x` defines a class named `figure`. To make a plot, first create a `figure` object using

```
fig = figure.new();
```

If `X` and `Y` are the coordinates of a sequence of points, then

```
fig.plot(X, Y);
```

will plot `Y` against `X`. The result is the points joined by straight line segments.
To plot `(X,Y)` as dots, use

```
fig.plot(x=X, y=Y,  linetype = "dot");
```

To plot `(X,Y)` as circles, use

```
fig.plot(x=X, y=Y,  linetype = "circle");
```

To plot a smooth curve through the `(x,y)` points, use

```
fig.plot(x=X, y=Y,  linetype = "curve");
```

To plot a smooth curve through the `(x,y)` points, using red color, and making the line thicker

```
fig.plot(x=X, y=Y,  linetype = "curve", linewidth = 2, color=[1,0,0]);
```

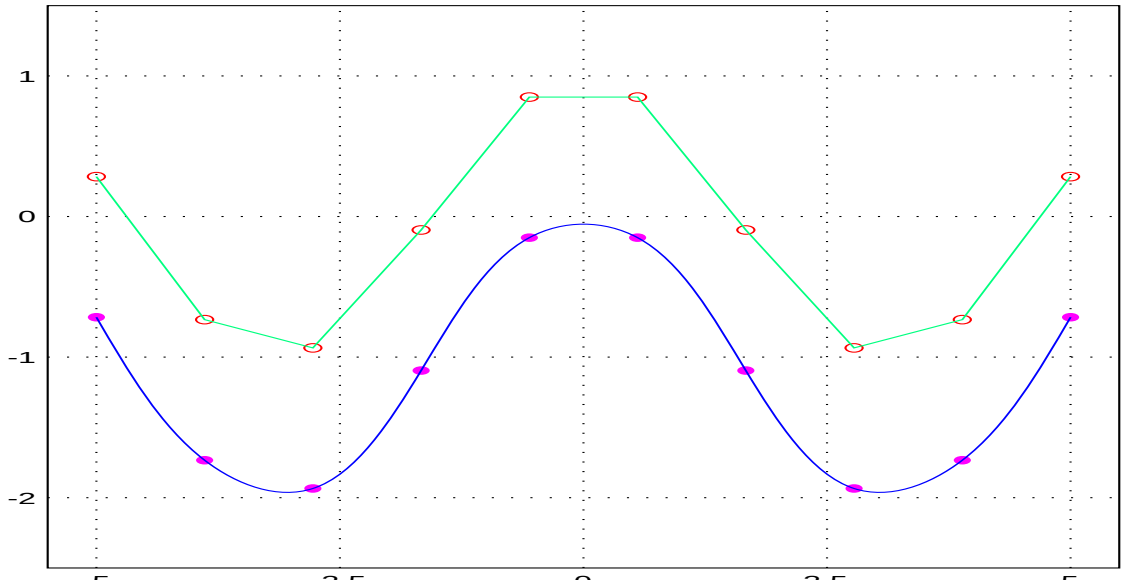
When the plotting is done, pick a file name (extension eps) and save the picture

```
fig.save("file_name.eps");
```

For example, the following commands will plot some curves and dots

```
with("plot.x");
x = linspace(-5, 5, 10);
fig = figure.new();
/* draw small circles at (x,y) locations */
/* color is vector of 3 numbers between 0 and 1, in rgb format */
fig.plot{x=x, y=cos(x), linetype="circle", color=[1, 0, 0]};
/* connect the dots with solid lines, linetype is the default value "solid" */
fig.plot{x=x, y=cos(x), color=[0,1,0.5]};
```

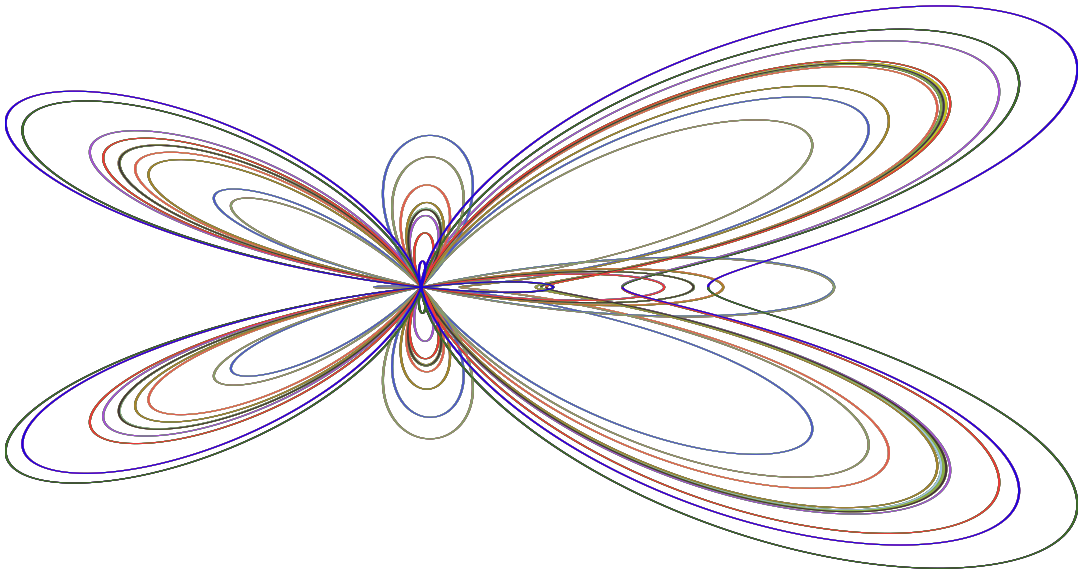
```
/* draw circular dots at location underneath the previous curve */
fig.plot{x=x, y=cos(x) -1, linetype="dot", color=[1, 0, 1]};
/* connect the dots with smooth curve */
fig.plot{x=x, y=cos(x) -1, linetype="curve", color = [0, 0, 1]};
/* set the xrange and yrange of the picutre */
fig.setXrange([-5.5, 5.5]);
fig.setYrange([-2.5, 1.5]);
fig.save("curves.eps");
```



The following commands will draw a butterfly

```
with("plot.x");
fig = figure.new();
for k = 0 : 99
   theta = linspace(k * 2 * pi, (k + 1) * 2 * pi, 500);
   rho = exp(cos(theta)) - 2 * cos(4*theta)+sin(theta/12).^5;
   x = rho .* cos(theta);
   y = rho .* sin(theta);
   fig.plot{x=x, y=y, linetype="curve", color=rand(3)};
end
fig.style = "empty";
fig.save("butterfly.eps");
```

For more examples, check out Shang website.



8 More Advanced Topics

There are many, many other features of Shang not covered in this tutorial. If you're interested more advanced topics on Shang programming, please take a look at the language reference manual and other documentations.