Functions, Conditional Statements

**Functions**

Mathematica has a large amount of functions already built in. The arguments of a function are put in between square brackets and separated by commas. Some basic functions are given below. Note that all built in functions and variables start with capital letters. To avoid potential conflicts it is a good idea to start your own functions and variables with lower case letters.

- `Sqrt[x]` square root (`\sqrt{x}`)
- `Exp[x]` exponential (`e^x`)
- `Log[x]` natural logarithm (`\log_e x`)
- `Log[b, x]` logarithm to base `b` (`\log_b x`)
- `Sin[x]`, `Cos[x]`, `Tan[x]` trigonometric functions (with arguments in radians)
- `ArcSin[x]`, `ArcCos[x]`, `ArcTan[x]` inverse trigonometric functions
- `Abs[x]` absolute value
- `Round[x]` closest integer to `x`

For example, we can compute:

- `Exp[3]`
- `\[Pi] Exp[3]`
- `Round[Sqrt[5]]`

**User defined functions**

We can define our own functions as follows:

```math
f[x_] := x^2 + Sin[x]
```

In this expression the underscore represents a pattern: A single underscore will match a single expression (argument) and `x` is the name of the pattern (it can be used to refer to the matched expression on the right hand side). The colon tells Mathematica not to evaluate the right hand side. It will only be evaluated when you invoke the function. This is why you get no output. Now you can use your function:

- `f[angle] \cdot \Pi`
- `f[2 \Pi]`

Try to assign some value to `x`, then compute `[2\Pi]`. What do you notice? Does `f` use the global or the local values of `x` when computing `[2\Pi]`?

**Warning:** Do not omit the colon when defining functions!

```math
g[x_] = x^2 + Sin[x]
```

Can you explain this result? You can look at the definition of a function by using a question mark mark:

- `? f`

To clear the function or variable definitions, use `Clear[...]`

- `Clear[x, f, g]`

The function `f` gives the sum of two variables. It works for any type of arguments that can be added in Mathematica. Let us evaluate `f` with some integer arguments, and then with some vector arguments.

- `f[2, 3]`
- `f[{1, 2, 3}, {3, 4, 5}]`

The condition operator `/;`

Many functions are only valid if certain conditions are met. Mathematica can handle such situations. The Condition operator (`/;`) can be used to make sure a pattern is used only if a condition is met.

```math
f[x_] := 0 /; x < 0
f[0] := 1
f[x_] := Log[x] /; x > 0
```

You can list all definitions associated with a function as before. In particular, this will also show the order in which the definitions are applied.

- `? f`

Exercise: Compute `fun[0]`, `fun[-5]` and `fun[0]`. You can also place the condition operator in the argument of the function

- `fun[x_ /; x < 0] := 0`
- `fun[0] := 10`
- `fun[x_ /; x > 0] := Log[x]`
Some functions only make sense for some type of input, like integers.

- `IntegerQ[expr]` true if expr is an integer, false otherwise
- `EvenQ[expr]` even number
- `OddQ[expr]` odd number
- `PrimeQ[expr]` prime number
- `NumberQ[expr]` explicit number of any kind
- `NumericQ[expr]` numeric quantity
- `VectorQ[expr]` a list representing a vector
- `MatrixQ[expr]` a list of lists representing a matrix
- `FreeQ[expr, form]` form matches nothing in expr
- `MatchQ[expr, form]` expr matches the pattern form
- `TrueQ[expr]` tests whether expr is true

```mathematica
f3[x_] := NumberQ[x] := x^3
```

Evaluate the function `f3`

```mathematica
f3[3]
f3[1, 2, 3]
```

### Plotting a function

Documentation available at
http://reference.wolfram.com/language/ref/Plot.html

```mathematica
Plot[f3[x], {x, -3, 3}]
```

There are a number of options that can be used with the `Plot` command, see the Mathematica documentation for a comprehensive list of available options and examples. For instance, `AxesLabel` specifies labels for the x and y axes. `PlotStyle` sets the style of the curve. You can fill in the space between the curve and the x-axis using the option `Filling`.

```mathematica
Plot[f3[x], {x, -3, 3}, AxesLabel -> ("x-axis", "f3[x]")]
```

### Conditional Statements

`If[condition, t, f]` - the "If" command returns `t` if condition evaluates to true and `f` if condition evaluates to false.

```mathematica
If[3 > 4]
If[3 > 4, 10, 5]
If[3 < 4, 10, 5]
```

**Elementary numerical relations:**
- `==` (is equal to)
- `!=` (is not equal to)
- `<` (is less than)
- `<=` (is less than or equal to)
- `>` (is greater than)
- `>=` (is greater than or equal to)

**Elementary logical relations:**
- `||` (or)
- `&&` (and)
- `!` (not)

```mathematica
x = 2; If[x == 0, Print["x is 0"], Print["x is different from 0"]]"
Exercise: Write some `If[...]` command that tests whether x is a prime number between 10 and 20. If true, it prints x, otherwise, it prints x + 1.

More on Conditional Statements

Sometimes we only need to perform an action when the condition is true, so we can use the command `If[...]` with only two arguments.

```math
x = 0; If[x = 0, Print["x is 0"]]
```

If we want to execute more commands when x is 0, we put a semicolon `;` between the successive commands.

```math
x = 0; If[x = 0, Print["x is 0."]; x = x + 2; Print["We have assigned the value 2 to x."], Print["x is not 0"]]
```

Switch

`Switch[expr, form1, value1, form2, value2, ..., default]` - compares `expr` with each of the forms `form1`, `form2`, ..., giving the value associated with the first form it matches. If `expr` does not match any form, then the default value is returned.

```math
expr = 3; Switch[expr, 
  1, Print["expr is 1"], 
  2, Print["expr is 2"], 
  3, Print["expr is 3"], 
  _, Print["expr has some other value"]]
```

Recall from last time

`Table[f, {i,m}, {j,n}]` builds an m x n matrix by evaluating the function `f` with arguments `i` and `j`, where `i` ranges from 1 to `m` and `j` ranges from 1 to `n`. The lower bound is implicitly 1, so we only specify the upper bound.

The following commands generate random 4x4 matrix with entries between 0 and 1:

```math
Rn = Table[Random[], {i, 4}, {j, 4}]
R1 = Table[RandomInteger[], {i, 4}, {j, 4}]
```

We can also define our own functions and use them to generate matrices with certain patterns.

```math
Clear[f]; 
f[i_, j_] := i + j

NwM = Table[{i, j}, {i, 4}, {j, 4}]; 
MatrixForm[NwM]
```