Getting Started with Mathematica (II)

Loops (For, While, Do)

Control structures (Break, Return)

Graphs and adjacency matrices

Loops and control structures

- `Do[expr, {i, imax}]` - evaluates expr repetitively, with i varying from 1 to imax in steps of 1.
- `Do[expr, {i, imin, imax, di}]` - evaluates expr with i varying from imin to imax in steps of di.
- `Do[expr, {i, list}]` - evaluates expr with i taking on values from a list.
- `Do[expr, {n}]` - evaluates expr n times.

Examples with Do:

- Print the first 10 positive integers:
  ```mathematica
  Do[Print[i], {i, 1, 10}]
  ```

- Print only the odd integers between 4 and 10:
  ```mathematica
  Do[Print[i], {i, 4, 10, 2}]
  ```
  ```mathematica
  Do[If[Mod[i, 2] != 0, Print[i]], {i, 4, 10, 1}]
  ```

Loops and control structures (Break, Return)

We can use some control flow functions to terminate the loop:

- `Break[]` - causes the loop to terminate
- `Return[expr]` - causes the loop/function to terminate and returns the value expr.

Print the first odd integer between 4 and 10

```mathematica
Do[If[Mod[i, 2] != 0, Print[i]], {i, 4, 10, 1}]
```

Print the first odd integer between 4 and 10

```mathematica
Do[If[Mod[i, 2] != 0, Return[i]], {i, 4, 10, 1}]
```

Loops (For)

- `For[start, test, incr, body]` - evaluates start, then repetitively evaluates body and incr, until test fails

```mathematica
For[i = 1, i < 4, i++, Print[i]]
```

First i is assigned the value 1. As long as the condition i < 5 remains true, i is incremented by 1 (i++ is just a shortening for i = i + 1) and the body of the function For is executed (that is, we print the integer i).

```mathematica
v = {1, 2, 5, 4, 10}
```

Recall that to access the first element of the list, we use `v[[1]]`.

```mathematica
v[[1]]
```

We can use a For loop to find the product, as follows:

```mathematica
prod = 1;
For[i = 1, i ≤ Length[v], i++, prod = prod * v[[i]]];
prod
```

Mathematica also has a built-in command for finding the product of the elements of a list/vector.

```mathematica
Product[v[[1]], {i, Length[v]}]
```
Loops (While)

While[condition, body] - evaluates body repetitively, so long as condition is True.

```mathematica
n = 17; While[(n = Floor[n/2]) ≠ 0, Print[n]]
```

Even if the last value printed by the While function is 1, the current value of n is not 1, as one may assume, but 0. This value was not printed when we ran the While command, because when n is 0, the condition n != 0 is false, so the body of the function While (which would have been Print[0]) is not executed.

```mathematica
n = 0
```

In the Wolfram Language, both While and For always evaluate the loop test before evaluating the body of the loop. As soon as the loop test fails to be True, While and For terminate. The body of the loop is thus only evaluated in situations where the loop test is True.

```mathematica
While[False, Print["False"]]
```

The command While[False, Print["False"]]) does not print anything, because the test condition is always False, so the body of the function is never executed.

The functions While and For in the Wolfram Language are similar to the control structures While and For in languages such as C++. Notice, however, that there are a number of important differences. For example, the roles of comma and semicolon are reversed in Wolfram Language For loops relative to C++ language ones.

Exercise: Find the sum of the elements of a matrix.

```mathematica
A = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
MatrixForm[A]
```

We can use Do:

```mathematica
sumElements = 0;
Do[sumElements = sumElements + A[[i]][[j]],
 {i, 1, Length[A]}, {j, 1, Length[A]}];
Print[sumElements]
```

We can use two For loops to find the sum, as follows:

```mathematica
sumElements = 0;
For[i = 1, i ≤ Length[A], i++,
 For[j = 1, j ≤ Length[A],
  j++,
  sumElements = sumElements + A[[i]][[j]]]]; 
Print[sumElements]
```

We can use two While loops:

```mathematica
sumElements = 0;
i = 1;
While[i ≤ Length[A],
   j = 1;
   While[j ≤ Length[A],
      sumElements = sumElements + A[[i]][[j]];
      j++;
   i++
 ];
Print[sumElements]
```

We can use a built-in command in Mathematica for computing the sum, whose syntax resembles that of a Do command.

```mathematica
Sum[A[[i]][[j]], {i, 1, 3}, {j, 1, 3}]
```

Graphs

Definition: A graph G=(V, E) consists of a set of vertices V (also called nodes) and a set of edges E. If there is an edge between the nodes u and v if and only if there is an edge between v and u, then G is called an undirected graph. Otherwise it is
called a directed graph.

In *Mathematica*, a graph is represented either by a list of rules of the form \{vi->vj,...\}, where vi and vj are vertices, and vi->vj is the edge between vi and vj, or by the adjacency matrix of the graph.

\[
G = \text{Graph}[\{1 \rightarrow 2, 3 \rightarrow 2, 3 \rightarrow 4, 4 \rightarrow 1\}]
\]

You can write click on the graph to set the display properties of the graph (layout, size, arrow shape, vertex shape, etc.). You can also plot the graph by explicitly listed the display options.

\[
\text{GraphPlot}[G, \text{VertexLabeling} \rightarrow \text{True}, \\
\text{DirectedEdges} \rightarrow \{\text{True}, \text{"ArrowheadsSize"} \rightarrow 0.15\}, \\
\text{PlotStyle} \rightarrow \text{Dashed}]
\]

### Adjacency matrix

The adjacency matrix of a graph G is built by the following rule: the (i, j) entry in the matrix is 1 if there is an edge between i and j in the graph G, and 0 otherwise.

\[
\text{AdjacencyMatrix}[G]
\]

By default, if the graph does not have a lot of edges, its adjacency matrix in *Mathematica* is given as a sparse array (only the non-zero elements are listed), but we can convert it to the regular form by using MatrixForm. In practice, for directed graph we will use more often the transpose of the adjacency matrix.

\[
\text{MatrixForm}[\text{AdjacencyMatrix}[G]]
\]

### Random Graphs

We can construct a random graph using a random matrix with 0 and 1.

\[
\text{Ri} = \text{Table}[\text{RandomInteger[]}, \{4\}, \{4\}]
\]

\[
\{(0, 1, 0, 1), (0, 1, 0, 1), (1, 0, 1, 1), (0, 1, 0, 1)\}
\]

\[
\text{RG} = \text{GraphPlot}[\text{Ri}, \text{VertexLabeling} \rightarrow \text{True}, \text{DirectedEdges} \rightarrow \text{True}]
\]