MODULES IN MATHEMATICA

As we have seen in many instances, Mathematica defines functions and values to be global, meaning if we say x = 3, then x = 3 until we change it or redefine it. Sometimes in programming, it is useful to define variables locally, meaning that we can define these variables in an isolated region of our program. In these localized environments, the variables can have a different set of values (from the global definitions) and be used in local operations.

In Mathematica, we call these isolated portions Modules. If we refer to the online doc center, we see that Mathematica defines the structure of modules as :

```
Module[{x, y, ...}, expr]
```

This means that x, y in the expression that follows should be treated as local variables only. If we write :

```
Module[{x = x0, y = y0}, expr]
```

we are setting the local variables x and y to have pre defined values of x0 and y0. Let' s see how this works :

Let' s start with a very simple program :

```
x=3;
Module[{x=8},x=x^2]
```

If we execute this statement, we expect to return the square of x; but will we get 3^2 or 8^2? Let' s execute and see :

```
64
```

Now, if we just type x and enter :

```
x
```

```
3
```

We get the original value of 3 without any further instructions. This shows the essential nature of modules; within the module, we can define variables to have local values (i.e., values that are valid only in the module) and can perform various operations on those variables within the module.Once when exit the module, the variable x takes on its global value.

Let' s try another Module :

```
x=2; y=3;
Module[{x=1, y=2},x=x+3;y=y+4;Print[x y]]
Print[x y]
```

```
24
6
```

What is going on inside the Module? We have two separate commands to Print[x y], one yields the output 24 and the other yields the output 6. Can you explain these different results?

We can use Modules to define more complex processes. Suppose we want to list all the factors of an integer, we define a function called factorlist :

```
factorlist[x0_]:=Module[{x=1},While[x≤x0, If[Mod[x0,x]==0, Print[x]];x++]]
factorlist[1250]
```

```
1
2
5
10
25
50
125
250
625
1250
```

Can you explain what is going on inside the Module; what is the internal logic in this Module that allows us to produce a complete list of factors?

In class exercise : For this exercise, we will investigate a conjecture in Mathematics which states that if you start with any positive integer n, replace it with n/2 if n is even and with 3 n + 1 if n is odd, and then repeat that process repeatedly, you will eventually end with the value 1 :

Example : Starting with 30 :

30/2 = 15
3 x 15 + 1 = 46
46/2 = 23
23 x3 + 1 = 70
70/2 = 35
35 x3 + 1 = 106
106/2 = 53
53 x3 + 1 = 160

160/2 = 80
80/2 = 40
40/2 = 20
20/2 = 10
10/2 = 5
5 x3 + 1 = 16
16/2 = 8
8/2 = 4
4/2 = 2
2/2 = 1 ....that was easy.

Write a short program involving a Module that will perform this procedure and verify the conjecture. It will be helpful to write a statement that models the conjecture, and a module that uses the conjecture to do the various calculations. Remember to print your output.

In[3713]:=
```
(* Writing the conjecture in the form of an IF statement. *)
(* We define the function nextnumber to compute the result of our process: *)

nextnumber[n_]:= If[EvenQ[n],n/2,3n+1]

(* Now we define a new function named allvalues that will print out all the results of the

allvalues[n_]:=Module[{m=n},While[m≠ 1, m = nextnumber[m];Print[m]]]
allvalues[30]
```

```
15
46
23
70
35
106
53
160
80
40
20
10
5
16
8
4
2
1
```

And we obtain the expected set of results. As we demonstrated in class, a simple change in the program can have significant impact on the output:

In[3716]:=
```
Clear[nextnumber,allvalues]
nextnumber[n_]:= If[EvenQ[n],n/2,3n+1]
allvalues[n_]:=Module[{m=n},While[m≠ 1, m = nextnumber[m]];Print[m]]
allvalues[30]
```

During evaluation of In[3716]:=
```
1
```

The two programs look identical at first glance; but if you look carefully, there is one more bracket at the very end of the Module in the first version, and one more bracket after nextnumber[m] in the second. Clearly this seemingly minor shift has had major consequences, but how and why?

In the first versio of the program, the Print statement was inside the While loop. Therefore, everytime the program stepped through the loop, a new value of m was computed, and this value of m was printed each time an iteration was completed.

In the second version of the program, the Print statement is still inside the Module, but it is now outside the While loop. Therefore, the Print statement is reached only after the condition that m != 1 is violated; in other words, the Print statement is accessed only when m = 1, and that is the only value of m that gets printed out.

Now, let' s see if we can amend this program slightly to allow us to print out the total number of iterations without having to print each intermediate result. I will introduce a counter, and set it equal to zero before any iterations are completed :

In[3727]:=
```
allvalues[n_] := Module[{m = n, k = 0},
  While[m ≠ 1, m = nextnumber[m]; k++]; Print["The final value reached = ", m, "
 The total number of iterations = ", k]]
allvalues[30]
```

```
The final value reached = 1
 The total number of iterations = 18
```

Notice that we increment k inside the While statement, so that each time a new value of m is calculated, the k counter increases by 1. When m = 1 is reached, the program exits the While statement without incrementing k, so the value of k that is passed to the Print statement represents the total number of steps needed to reach the final value of 1

How many steps would it take to reduce a really big number to m = 1? :

In[3726]:=
```
Timing[allvalues[(54 879 315 792 454 928 320 487 963 547)^453]]
```

```
The final value reached = 1 The total number of iterations = 308 729
```

Out[3726]=
```
{5.593, Null}
```

Try this one and see how long it takes for Mathematica. Compare how long it would take you by hand ...