Mathematica as Lisp

Outline of this talk:
1) Show a few "Gee Whiz" features of *Mathematica*
   -show some algebraic calcs
   -show some nice typesetting
   -show some pretty graphics
2) Is it lisp?
- front end / back end
- Functions/transformations
- "Macros"
- Functional programming
- closures
3) Some more interesting examples

It Does Algebra

In[1]:=
```
2 + 2
```

Out[1]= 4

In[2]:=
```
Integrate[x^n, x]
```

Out[2]= $\frac{x^{1+n}}{1+n}$

In[3]:=
```
Integrate[1 / (x^4 - a^4), x]
```

Out[3]= $-\frac{\text{ArcTan}\left[\frac{x}{a}\right]}{2\,a^3} + \frac{\text{Log}[a-x]}{4\,a^3} - \frac{\text{Log}[a+x]}{4\,a^3}$

In[4]:=
```
Integrate[(1 - x^2)^n, x]
```

Out[4]= $x\,\text{Hypergeometric2F1}\left[\frac{1}{2}, -n, \frac{3}{2}, x^2\right]$

In[5]:=
```
b = Solve[{x^2 + y^2 == 1, x + y == a}, {x, y}]
```

Out[5]= $\left\{\left\{x \to \frac{1}{2}\left(a - \sqrt{2-a^2}\right), y \to \frac{1}{2}\left(a + \sqrt{2-a^2}\right)\right\}, \left\{x \to \frac{a}{2} + \frac{\sqrt{2-a^2}}{2}, y \to \frac{1}{2}\left(a - \sqrt{2-a^2}\right)\right\}\right\}$

Note we have a list of lists, and the elements inside are expressions: we'll see more explicit functional constructs later

◄ | ►

It Can Do Typesetting

TeX *Mathematica* figure

Not as well as TEX, but better than Word! Note this presentation is in *Mathematica*. (Note it knew how to format TEX automatically). It has a very nice collection of special characters, viz: α, β, γ, ∂, ƒ, ℧, ☑, ♯, α ∫Æ∋| △ □

Note you can type these in from the keyboard, or from a pallate (q.v.), and you can evaluate some things in place:

$$\int x^2 \, dx + \int x^2 \, \mathrm{Sin}[x] \, dx \tag{1}$$

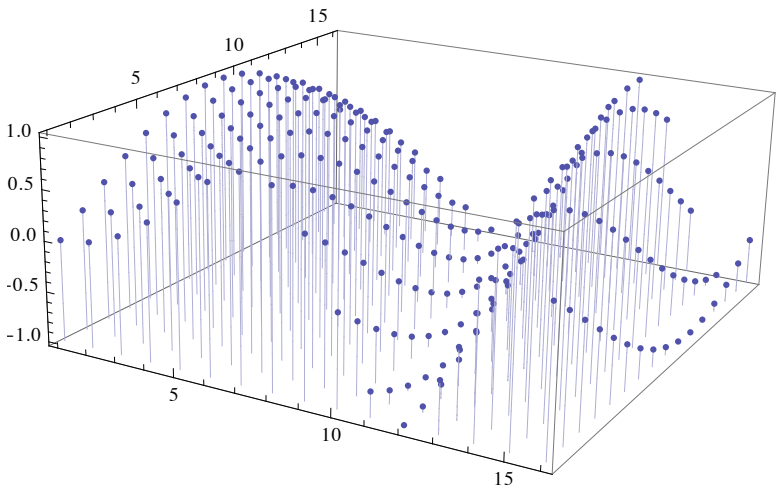
$$\int x^2 \, dx + \int x^2 \, \mathrm{Sin}[x] \, dx \tag{2}$$

You can produce fairly nice papers in pure *Mathematica*, and it can serve as a fairly good outliner.

It Does Graphics

This is a text cell.

```
ListPointPlot3D[
 Table[Sin[i^2 + i], {i, 0, 3, 0.2}, {j, 0, 3, 0.2}], Filling → Bottom]
```



Show that the above plot is REALLY an expression-

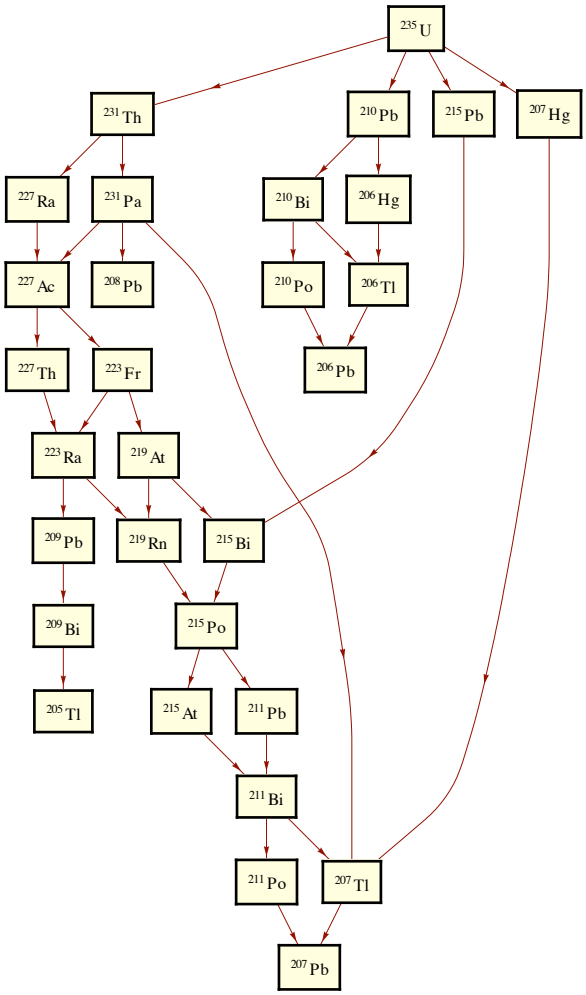
A Neat Graphics/Programming Example

We WON'T go over all of this in great detail- let's just tear apart the first one to get a feeling for what kinds of things are possible!

In[6]:=
```
DaughterNuclides[s_List] := DeleteCases[Union[
   Apply[Join, Map[[#, "IsotopeDataDaughterNuclides"] &, DeleteCases[s, _Missing]]]], _Missing]

ReachableNuclides[s_List] := FixedPoint[Union[Join[#, DaughterNuclides[#]]] &, s]

DecayNetwork[iso_List] :=
 Apply[Join, Map[Thread[# → DaughterNuclides[{#}]] &, ReachableNuclides[iso]]]

DecayNetworkPlot[s_] := LayeredGraphPlot[
  Map[IsotopeData[#, "Symbol"] &, DecayNetwork[{s}], {2}], VertexLabeling → True,
  VertexRenderingFunction → (Text[Framed[Style[#2, 8], Background → LightYellow], #1] &)]
```

Neat Graphics Example: Denouemont

```
DecayNetworkPlot["Uranium235"]
```



Architecture: Analogous to Listener and Emacs

There are two components you are seeing:
1) A graphics-less ASCII only (?) "dumb terminal" back end which does the heavy algebraic lifting: "The Listener". It is basically an interpreter.
2) A Front-End which communicates with the back end and does the pretty printing and does things like plots- a lot like emacs, programmable in a somewhat obscured subset of *Mathematica*.

Almost everything you see is an expression rendered by the Front End : look at a few of the figures on the last few pages.

The Notebook "front end" (see later) is very configurable (show options)

You can also use a more traditinal coding interface: *Wolfram Workbench*, based on *Eclipse,* with breakpoints, etc.

◄ | ►

Computational Model

When a line is executed, symbols are looked up, and appropriate substitutions are made from the system symbol tables, which have packages and all the "normal" niceties. The symbols have documentation and source linked to them at runtime.

```
?? Sin
```

Sin[z] gives the sine of z. ≫

```
Attributes[Sin] = {Listable, NumericFunction, Protected}
```

```
?? DecayNetworkPlot
```

```
Global`DecayNetworkPlot
```

```
DecayNetworkPlot[s_] :=
 LayeredGraphPlot[Map[IsotopeData[#1, Symbol] &, DecayNetwork[{s}], {2}], VertexLabeling → True,
  VertexRenderingFunction → (Text[Framed[Style[#2, 8], Background → LightYellow], #1] &)]
```

Mathematica effectively stores all definitions you give as lists of transformation rules. When a particular symbol is encountered, the lists of rules associated with it are tried. The mechanisms for this is very rich;
we'll explore a couple

Functions

With no knowledge of a symbol, we get the input back

```
Remove[earl]
```

```
? earl
```

Information:notfound: Symbol earl not found. ≫

```
earl[1, 2]
```

```
earl[1, 2]
```

We can also define a simple substitution that will be performed dynamically at runtime- a_ and b_ are patterns that will be substituted for-

```
earl[a_, b_] := a + b
```

```
?? Plus
```

$x + y + z$ represents a sum of terms. ≫

```
Attributes[Plus] = {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}
```

```
Default[Plus] := 0
```

```
?? earl
```

```
Global`earl
```

```
earl[a_, b_] := a + b
```

```
earl[γ, γ]
```

2γ

$(1+\gamma)^5$

$(1+\gamma)^5$

We've defined a symbol in a global table, and the substitution will be performed until the symbol definition is cleared/deleted/etc.

Transformations

You can also perform transformations in an ad-hoc fashion:

```
Remove[a]
```

```
a + b /. a -> Cos[a]
```

```
b + Cos[a]
```

One of the good/bad things about Mathematica is all the syntactic sugar. In expressions this is:

```
ReplaceAll[ a + b, Rule[a, Cos[a]]]
```

```
b + Cos[a]
```

You can apply rules once, or repeatedly

```
x^2 + y^6 //. {x -> 2 + a, a -> 3}
```

$25 + y^6$

```
log[a b c d] //. {log[x_ y_] -> log[x] + log[y], a → 3}
```

```
log[3] + log[b] + log[c] + log[d]
```

Here we wee how you might start building up an algebra system!

◄ | ►

Macro Like Thingys: HoldALL

One of the essences of Macros is to change the order of evaluation. This is enabled by HoldAll, which is an attribute which specifies that all arguments to a function are to be maintained in an unevaluated form.

```
until::usage = "until[cond,cmd] repeats evaluating cmd until cond is True.";

SetAttributes[until, HoldAll]

until[cond_, cmd_] := While[True, cmd; If[cond, Return[]]]

i = 1; until[Prime[i] > 10^6, i++]; i
```

```
2
```

◄ | ►

Functional Constructs

I've used *Mathematica* pretty heavily to do data analysis using functional constructs operating on lists. Rather than enumerating, let's browse the documentation for some examples. Note that functions *are* first class values.
To see the basic functions, highlight the following and hit Flower F Functional Programming

```
((# + 1) &)[4]
```

```
5
```

```
a = ((# + 1) &)
```

```
#1 + 1 &
```

```
a @ 5
```

```
6
```

◄ | ►

Closures - 1

An example of a closure from Wikipedia
; Return a function that approximates the derivative of f
  ; using an interval of dx, which should be appropriately small.
  (define (derivative f dx)
    (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))

```
derivative[ f_, dx_ ] := Function[(f[# + dx] - f[#])/dx]
```

```
derivative[ #^2 &, 0.001]
```

$$\frac{(\#1^2\ \&)[\#1 + 0.001] - (\#1^2\ \&)[\#1]}{0.001}\ \&$$

```
derivative[ #^2 &, 0.001][2]
```

```
4.001
```

But is this really a closure? What about sharing a value between functions?

Closures 2

Well, I have a feeling I'm missing something simple and transparent, but here's an implementation of a closure using Module: Module implements lexical scope, Block implements Dynamic Scope

```
a = Module[{x},
 x = 1;
 {Print[x] &,
  ++x &}
 ];
```

```
a
```

```
{Print[x$986] &, ++x$986 &}
```

```
a[[1]][]
```

```
1
```

```
a[[2]][]
```

```
3
```

```
a[[1]][]
```

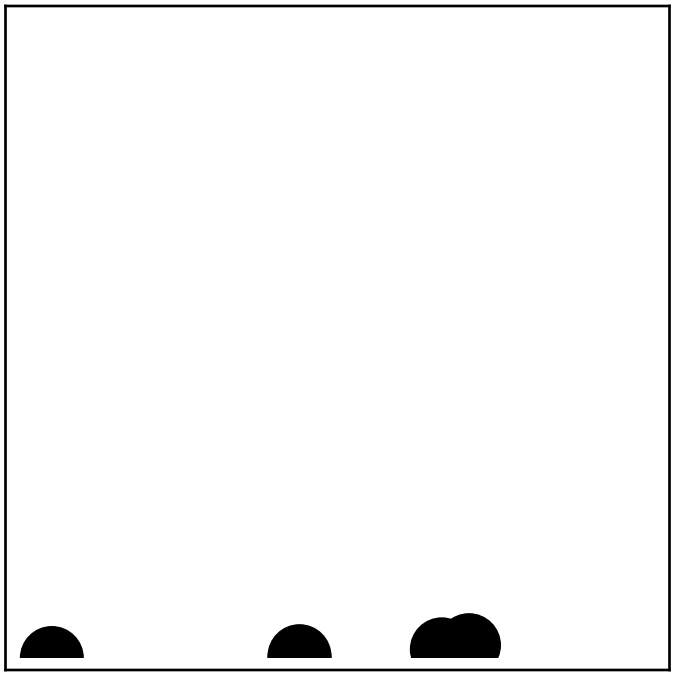
```
3
```

◄ | ►

A Couple of More Fun Things

An example from the documentation: Dynamic and DynamicModule are new "interactive" graphics items. Somehow this feels very dynamic and lispy to me.

```
Framed@DynamicModule[{contents = {}},
  EventHandler[Graphics[{PointSize[0.1], Point[Dynamic[contents = Map[If[#[[1, 2]] ≥ 0,
            {#[[1]] - #[[2]], #[[2]] + {0, 0.001}}, {{#[[1, 1]], 0}, {1, -0.8} #[[2]]}] &,
          contents]; Map[First, contents]]]}, PlotRange → {{0, 1}, {0, 1}}],
   "MouseDown" :→ (AppendTo[contents, {MousePosition["Graphics"], {0, 0}}])]]
```



Integrated Data

In the latest version several sets of "integrated" data have been incorporated - heck, see the documentation

◄ | ►

Symbolic Data Handling: XML

Mathematica has a very sophisticated pattern language which can match against Expressions being processed. It's beyond the scope of this talk to go into this in too much detail, and some of the details have leaked out of my ears, but here's just a superficial survey/example

The built in pattern maniuplation makes XML quite a bit easier to hack - heck, see the documentation-XML/tutorial/TransformingXML.

◄ | ►

```
$Aborted
```

New Slide

```
a = {1, 2, 3, 4, 5}
```

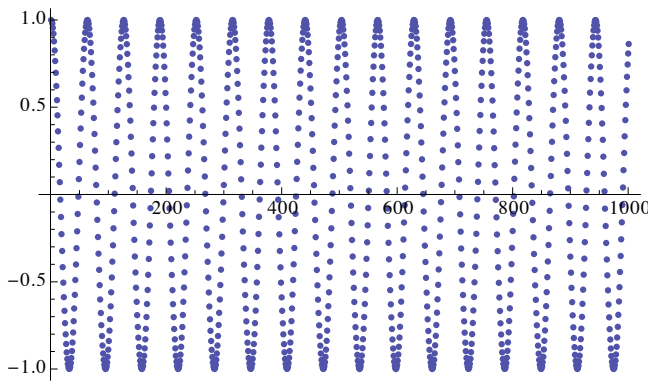
```
{1, 2, 3, 4, 5}
```

```
N[Cos[a]]
```

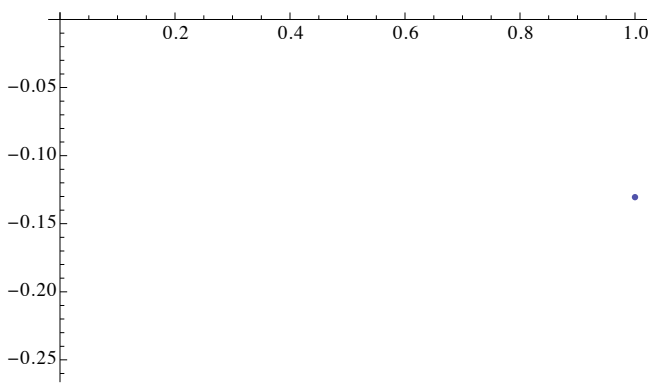
```
{0.540302, -0.416147, -0.989992, -0.653644, 0.283662}
```

```
a = Table[ Cos[x], {x, 0, 100, 0.1}];
```

```
ListPlot[a]
```




```
ListPlot[Chop[Fourier[a]]]
```




```
b = {{1, 2}, {3, 4}}
```

```
{{1, 2}, {3, 4}}
```

```
b // MatrixForm
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
a = {1, 2}
```

```
{1, 2}
```

```
a b
```

```
{{1, 2}, {6, 8}}
```

```
b a
```

```
{{1, 2}, {6, 8}}
```

◄ | ►