SQL databases

An introduction

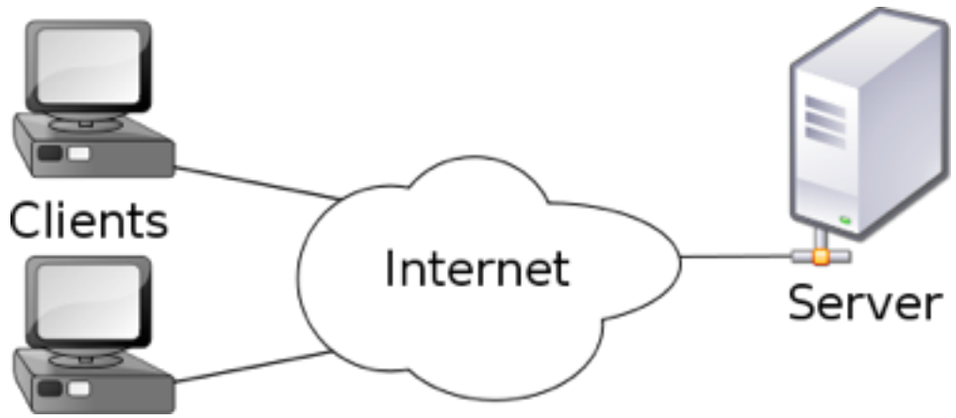
			3	11798	Carnivora	Felidae	Felis	sp.	S. Kuemmell	C. Linnaeus	1758	10883	Lufeng	China	Yunnan
			4	11798	Carnivora	Felidae	Metailurus	sp.	H. O'Regan	Zdansky	1924	10883	Lufeng	China	Yunnan
			5	11803	Carnivora	Felidae	Panthera	pardus	W. Clyde	C. Linnaeus	1758	10604	Tabun Cave Level C & D	Israel	Mount Carmel
			6	13066	Carnivora	Felidae	Panthera	onca	M. Uhen	C. Linnaeus	1758	10272	Ladds Quarry	United States	Georgia
			7	13066	Carnivora	Felidae	Lynx	rufus	M. Uhen	Schreber	1777	10272	Ladds Quarry	United States	Georgia
			8	13066	Carnivora	Felidae	Miracinonyx	inexpectatus	M. Uhen	E. D. Cope	1895	10272	Ladds Quarry	United States	Georgia
			9	13066	Carnivora	Felidae	Puma	concolor	J. Alroy	Linnaeus	1771	10272	Ladds Quarry	United States	Georgia
			10	13066	Carnivora	Felidae	Felis	sp.	M. Uhen	C. Linnaeus	1758	10272	Ladds Quarry	United States	Georgia

AMP: Apache, mySQL, PHP

This installations installs the Apache webserver, the PHP scripting language, and the mySQL database on your computer:

- **Apache:** runs in the background on your computer and waits for requests from web browsers, sending back the requested files (http://localhost/ to see the site now running on your laptop)
- **mySQL:** a relational database server that runs in the background of your computer and waits for requests from SQL clients, returning to them the requested data
- **PHP:** a computer language that was specially designed for use on internet servers, making it easy to extract customized data from databases and to reform it into web pages

Client-server model



A method of distributing access to a program

Server is the main program that handles information, calculations, data analysis, etc.

Client is the computer or program that asks for and receives information from the server

Web: client is your web browser, server is a web server application like Apache.

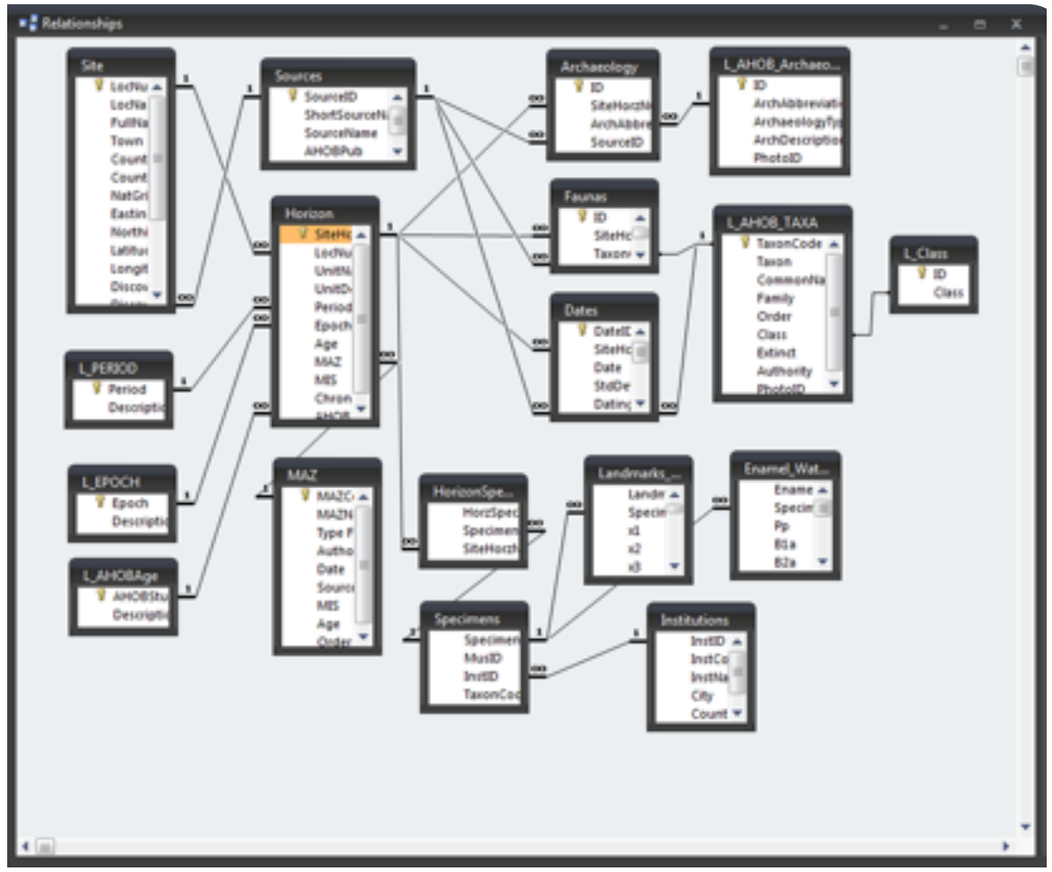
Databases: client is the program you use to query database, server is mySQL installed on a machine somewhere

iPhone Apps: client is the app installed on iPhone, servers is (usually) a web server like Apache

Advantage of client server model for databases: you can access data from MANY different programs without reformatting your data.

What is a relational SQL database?

- Software that stores and processes data
- Relational aspect refers to the ability to keep data in many tables that are linked by particular variables (e.g., "specimen" table may list of data about specimens in a museum, "taxon" table may contain information about species, the link allows users of the specimen table to find out about the species).
- SQL (Structured Query Language) is a simple language or syntax for issuing instructions to the database software
- Databases are useful for managing large amounts of data and for reformatting or collating them



from Polly & Stringer, 2011

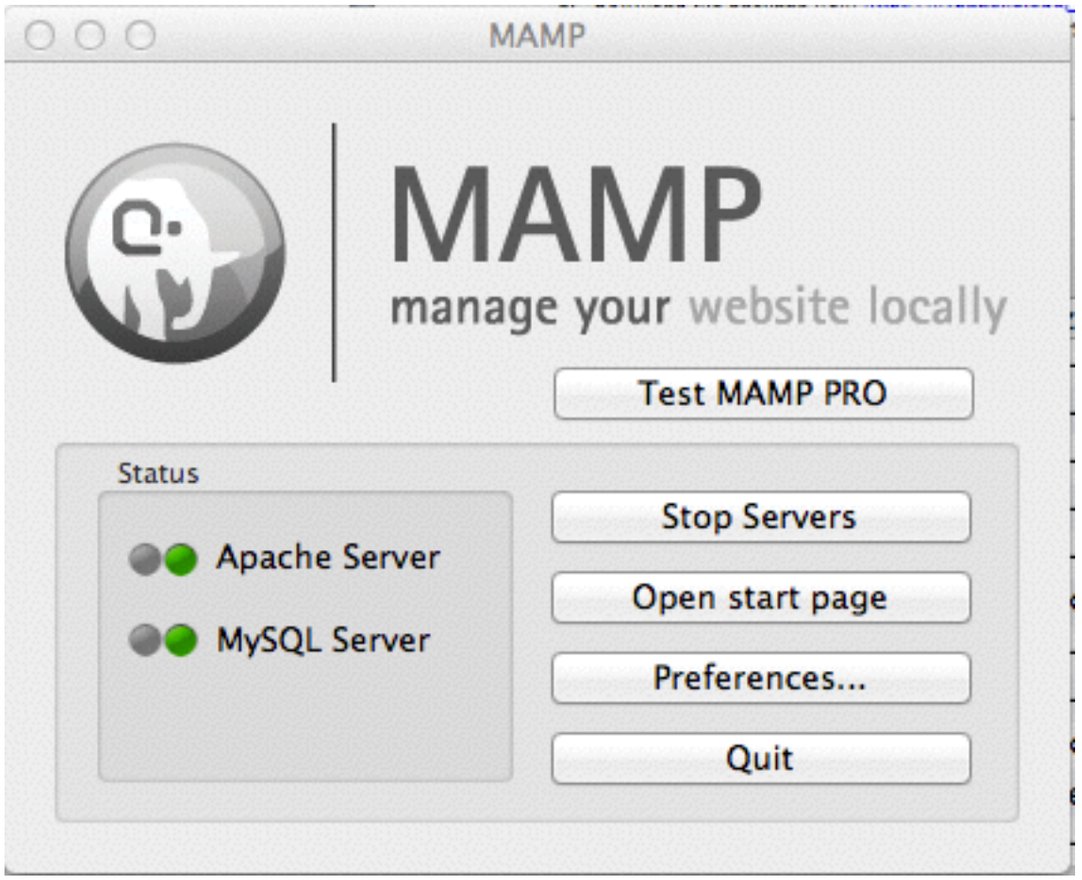
Graphic summary of relationships in a complex database with eighteen tables

mySQL database

- A free, open-source relational database (now owned by Oracle)
- SQL compliant
- Built on a client-server model
- Fast, can handle huge data sets
- Cross-platform
- Specially designed to interface with web applications using scripts (PHP, PERL, Python, Java, etc.)

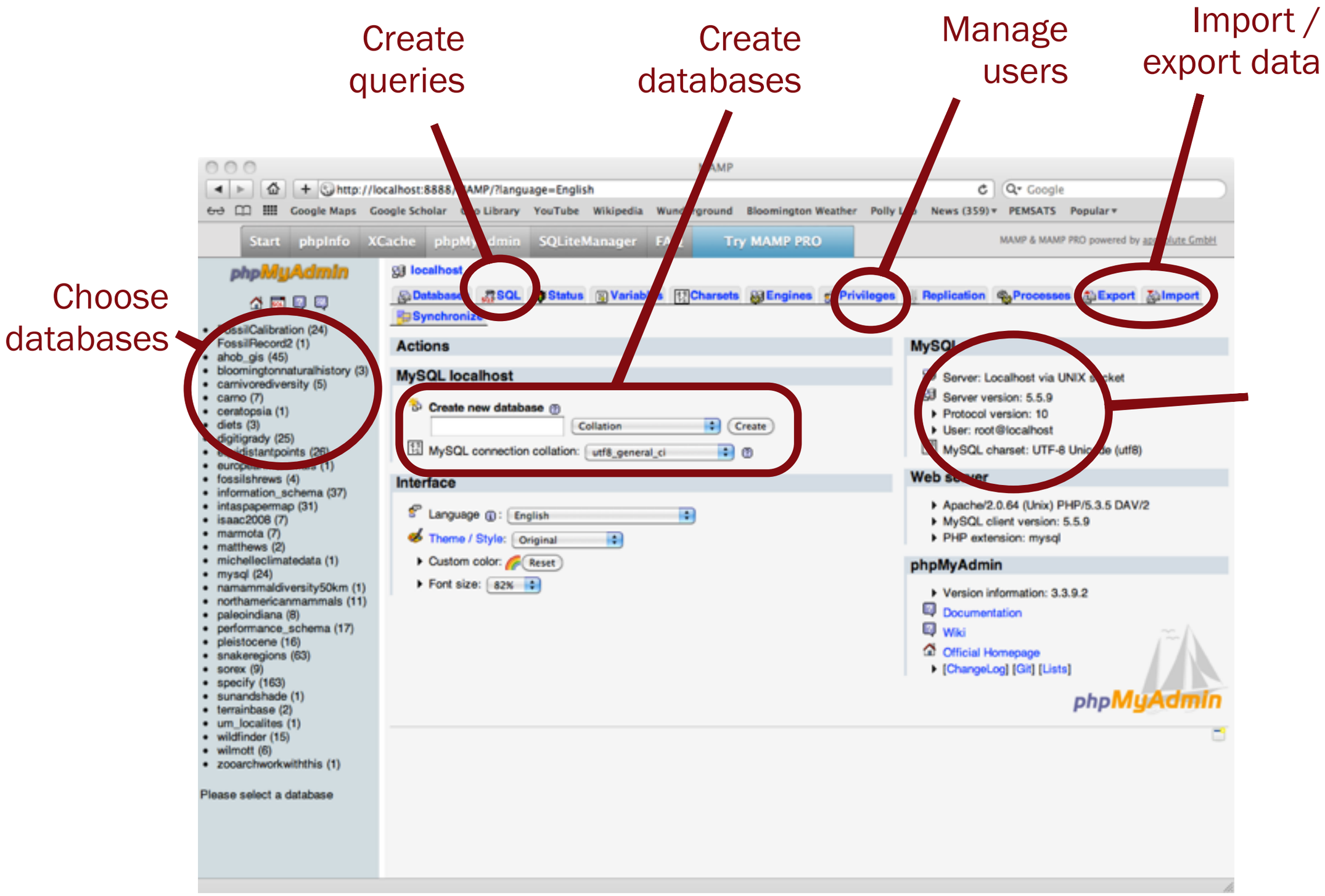
Start the mySQL Database Server and phpMyAdmin

- "Start Servers" starts both mySQL database and Apache web server
- Always stop servers when you are not using them
- Start page opens the PHPMyAdmin control interface

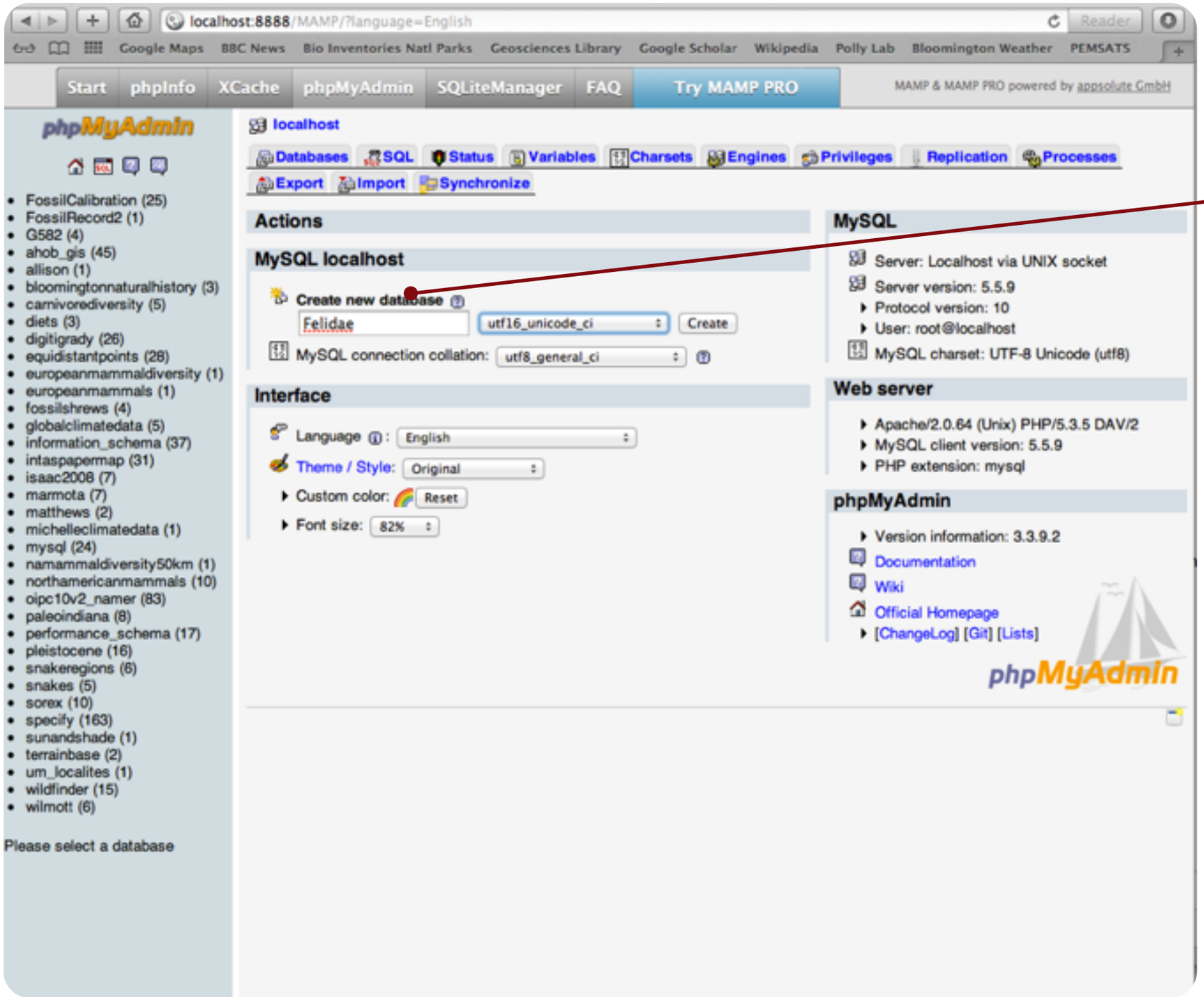


Mac OS X version. WAMP control looks different.

phpMyAdmin layout



Creating a database in phpMyAdmin

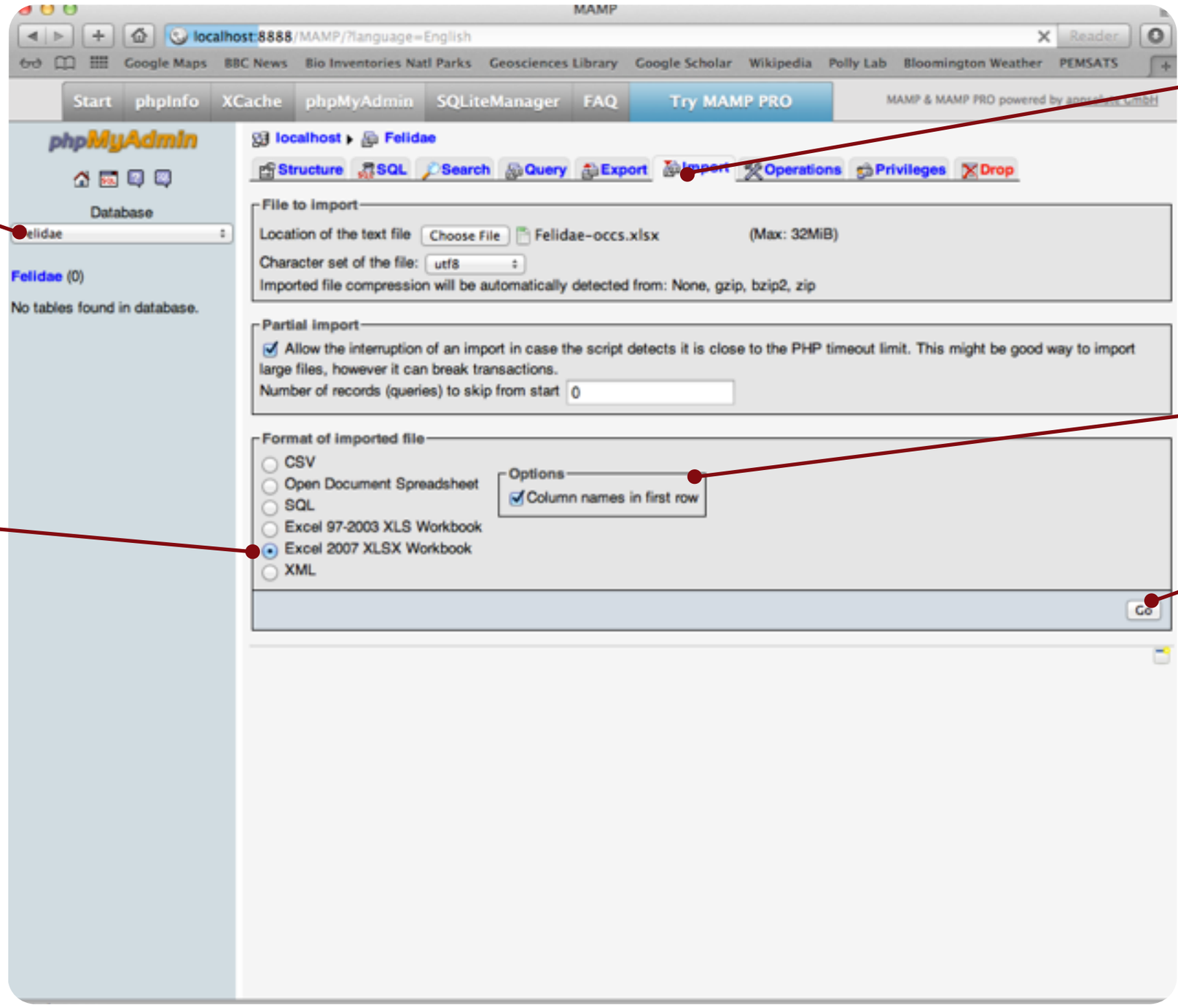


Create new database form (on home page)

Name your database and choose utf16_unicode_ci from the Collation dropdown menu

Press Create when ready

Importing data into your new database



Make sure the database you want is selected

Import tab

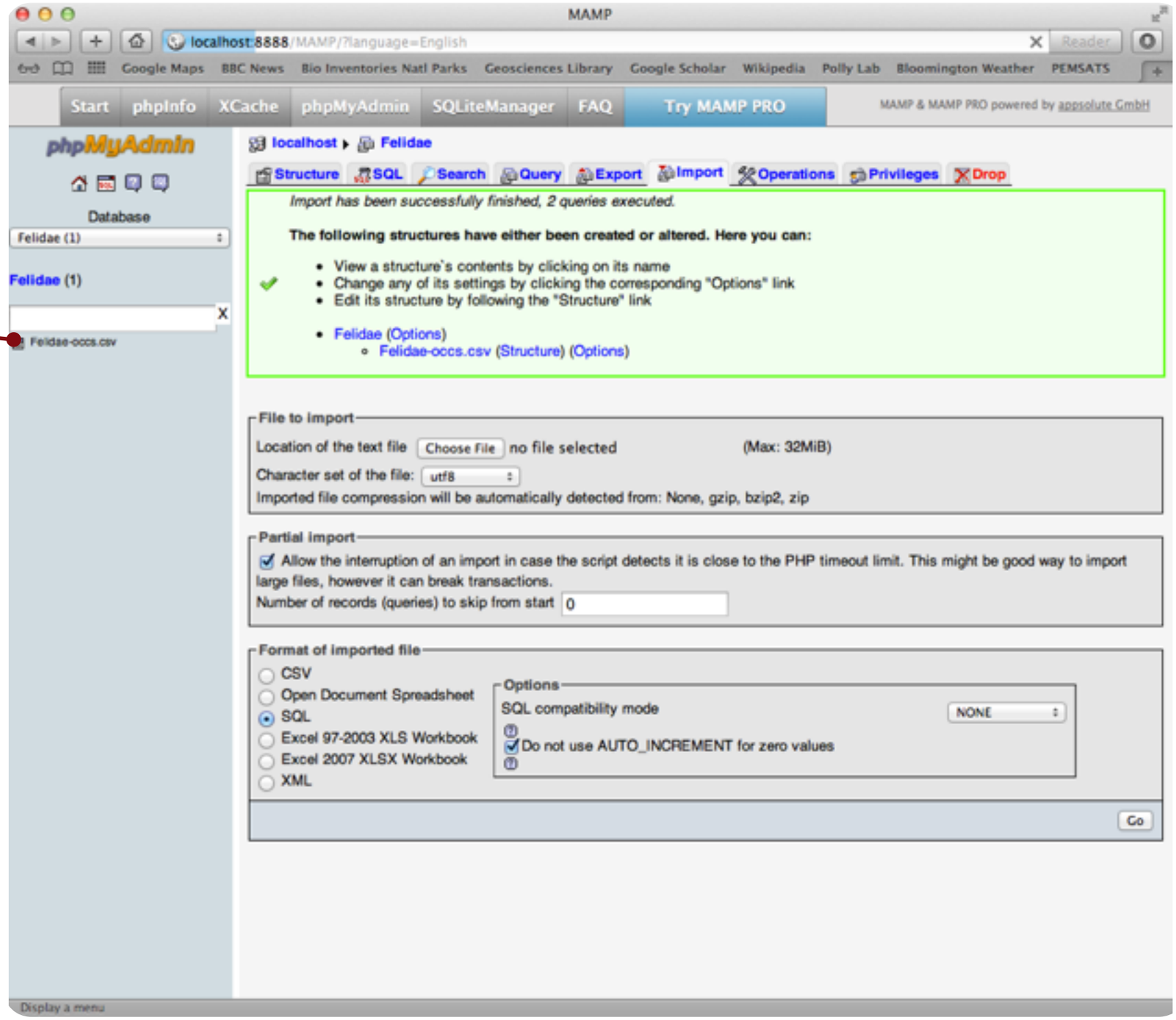
Tick column headers

Select Excel

Start importing

If successful....

New table is listed here



To import data from comma-delimited text file

1. Create new database to be home to your imported tables

2. Select database and click on Import tab

3. Choose file

4. Note that “import” is context dependent. If you have selected database, it will import as new table, but if you have selected a table it will import into that table rather than a new one

5. Once imported, select table from list at left

6. Change table name using the “Operations” tab

7. Add a “key” to your table by selecting your table then selecting the “structure” tab. At bottom add one variable at the beginning of the table. Name that (e.g., “ID”), give it an Integer type (INT), make it the primary index, and tick autoincrement (A_I). This key assigns a unique number to each line in the table, which is important for some operations.

8. Browse data with browse tab. Note that you can delete or edit lines of data using the “X” and the pencil icons.

Browsing and editing data

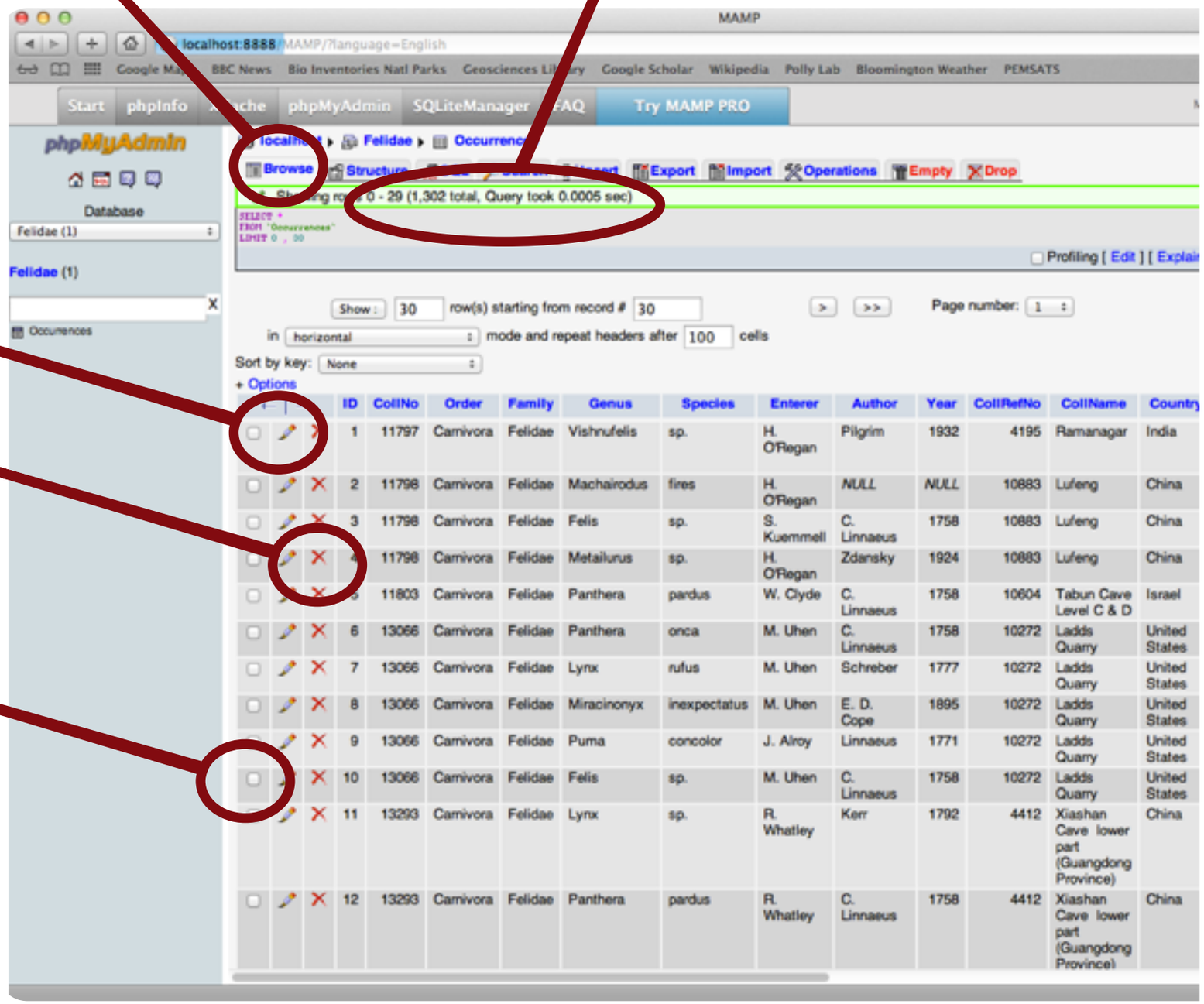
Browse tab

Statistics about data

Click pencil to edit a record

Click x to delete a record

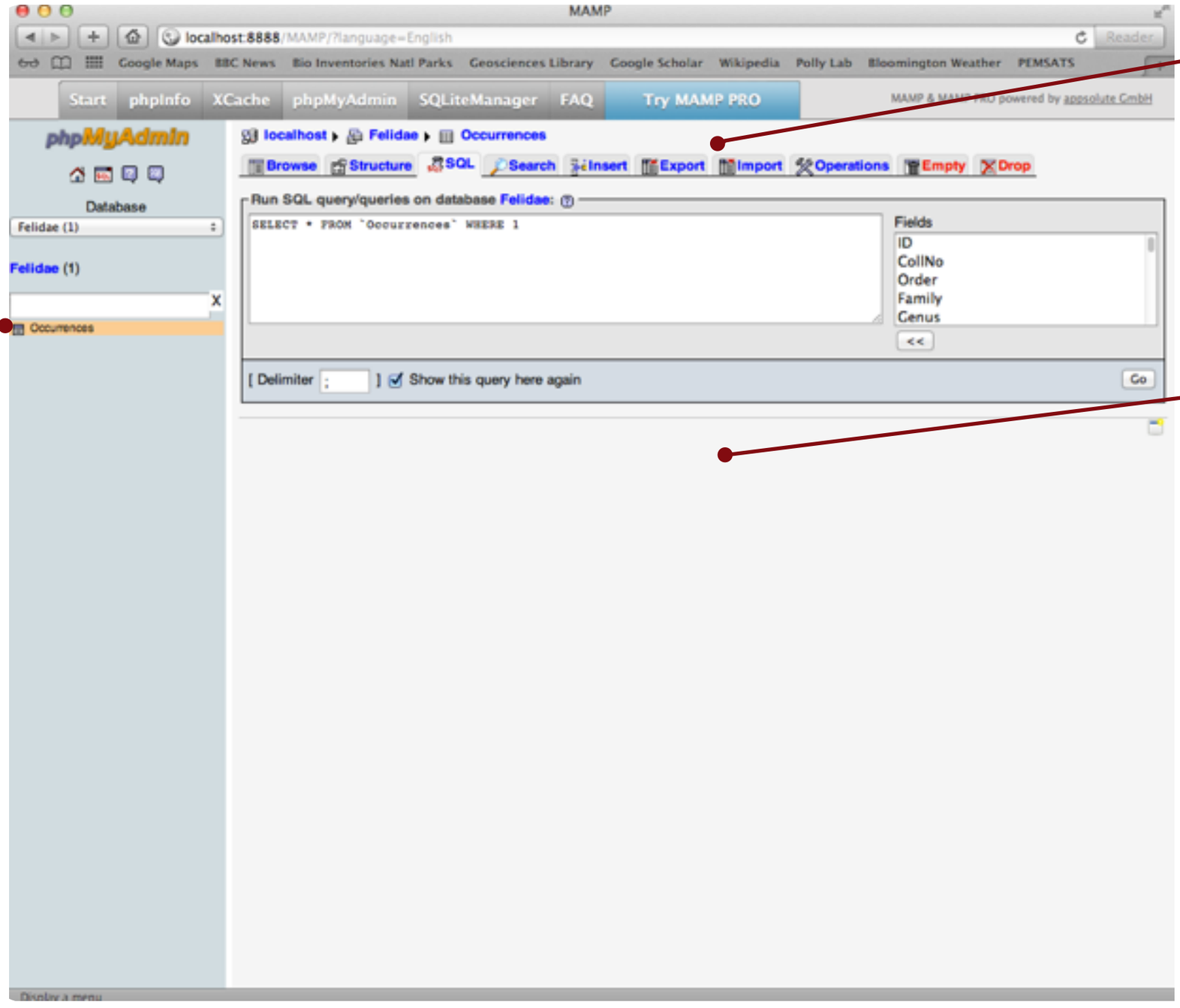
Tick box to select a record for editing or deleting multiple lines with control at bottom of the page



ID	CollNo	Order	Family	Genus	Species	Enterer	Author	Year	CollRefNo	CollName	Country
1	11797	Carnivora	Felidae	Vishnufelis	sp.	H. O'Regan	Pilgrim	1932	4195	Ramanagar	India
2	11798	Carnivora	Felidae	Machairodus	fires	H. O'Regan	*NULL*	*NULL*	10883	Lufeng	China
3	11798	Carnivora	Felidae	Felis	sp.	S. Kuemmell	C. Linnaeus	1758	10883	Lufeng	China
4	11798	Carnivora	Felidae	Metailurus	sp.	H. O'Regan	Zdansky	1924	10883	Lufeng	China
5	11803	Carnivora	Felidae	Panthera	pardus	W. Clyde	C. Linnaeus	1758	10604	Tabun Cave Level C & D	Israel
6	13066	Carnivora	Felidae	Panthera	onca	M. Uhen	C. Linnaeus	1758	10272	Ladds Quarry	United States
7	13066	Carnivora	Felidae	Lynx	rufus	M. Uhen	Schreber	1777	10272	Ladds Quarry	United States
8	13066	Carnivora	Felidae	Miracinonyx	inexpectatus	M. Uhen	E. D. Cope	1895	10272	Ladds Quarry	United States
9	13066	Carnivora	Felidae	Puma	concolor	J. Alroy	Linnaeus	1771	10272	Ladds Quarry	United States
10	13066	Carnivora	Felidae	Felis	sp.	M. Uhen	C. Linnaeus	1758	10272	Ladds Quarry	United States
11	13293	Carnivora	Felidae	Lynx	sp.	R. Whatley	Kerr	1792	4412	Xiashan Cave lower part (Guangdong Province)	China
12	13293	Carnivora	Felidae	Panthera	pardus	R. Whatley	C. Linnaeus	1758	4412	Xiashan Cave lower part (Guangdong Province)	China

Querying your database with SQL

Make sure the database and table you want is selected



Import tab

Tick column headers

Structured Query Language (SQL)

SQL is a standardized way to format instructions to a database server. It resembles English in its grammar and syntax.

Verb (required), Object (required), Object Modifiers (optional),
From Clause (required), Modifying Clause (optional)

```
SELECT  *  FROM  pbdb  WHERE  Genus='Hesperocyon';
```

Select Clause | From Clause | Modifying Clause

Verb gives the command (e.g., “Select”)
Object indicates what fields to select
Clause indicates what table the fields should be selected from
Modifying clause tells what criteria should be used to make the selection

* is the wildcard character, meaning “all fields”

SQL queries

Sentence form:

SELECT *whichever columns you want* **FROM** *whichever table you want* **WHERE** *your criteria are true*

For example, the following query....

SELECT * FROM occurrences

...generates the same browse page you already saw because it selects all columns (*) from the table *taxa* where no criteria are specified (i.e., it selects all rows).

And the following query....

SELECT * FROM occurrences WHERE Country='United States'

selects all the columns from only those rows where the Country column contains the word “United States”.

Finally, the following query....

SELECT Genus FROM taxa WHERE Country='United States'

selects only the taxon name from the rows where the TaxonLevel column contains the word “Species”.

Some powerful modifiers for SQL queries

Select a list of all the unique Countries the database
SELECT DISTINCT Country FROM occurrences

Count the number of entries for each Country
SELECT Country, Count(Country) FROM occurrences GROUP BY Country
note that “group by” is used to define the grouping variable and then the Count() function counts all the entries of Country in each group

Count the number of genera in the database from the USA
SELECT Genus, Count(Genus) FROM occurrences WHERE Country=‘United States’ GROUP BY Genus

Useful functions that can be used in the SELECT statement in conjunction with a GROUP BY statement include: Count(), Avg(), Max(), Min(), Std(), Sum(), Variance(). You can also use multiplication, division, addition, etc. (e.g., “Sum(TaxonID)+Count(TaxonLevel)”).

Quickly tabulate data with SQL

localhost ▸ Felidae ▸ occurrences

Browse | Structure | SQL | Search | Insert | Export | Import | Operations | Empty | Drop

Run SQL query/queries on database Felidae:

```
SELECT Genus, Count(Genus) FROM occurrences WHERE Country='United States' GROUP BY
Genus ORDER BY Count(Genus)
```

Fields: ID, CollNo, Order, Family, Genus, <<

[Delimiter ;] Show this query here again Go

Genus	Count(Genus)
Metailurus	1
Pratifelis	1
Xenosmilus	2
Dinofelis	4
Adelphailurus	5
Megantereon	7
Leopardus	7
Felis	14
Nimravides	19
Miracinonyx	26
Machairodus	26
Pseudaelurus	30
Homotherium	33
Puma	35
Panthera	48
Smilodon	53
Lynx	68

Mathematica

(c) Wolfram Research

Uses for *Mathematica*

Calculations, simple or complicated

```
In[1]:= 12 + 20
Out[1]= 32
```

```
In[2]:= ((10 * 5) + (20 * 30)) / 10^0.5
Out[2]= 205.548
```

Mathematical functions

```
In[7]:= Log[25] // N
Out[7]= 3.21888
```

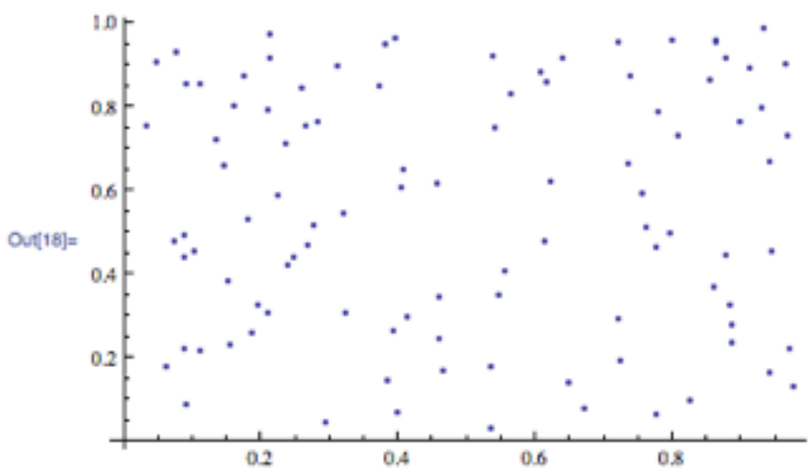
Statistical analysis

```
In[15]:= ANOVA[data]

                          DF    SumOfSq      MeanSq       FRatio      PValue
Out[15]= {ANOVA → Model   1     0.0682131    0.0682131    0.834059    0.363343
                  Error   98    8.01489      0.0817845                       ,
                  Total   99    8.0831

                      All          0.506327
          CellMeans → Model[1]     0.477453}
                      Model[2]     0.529952
```

Graphics in *Mathematica*

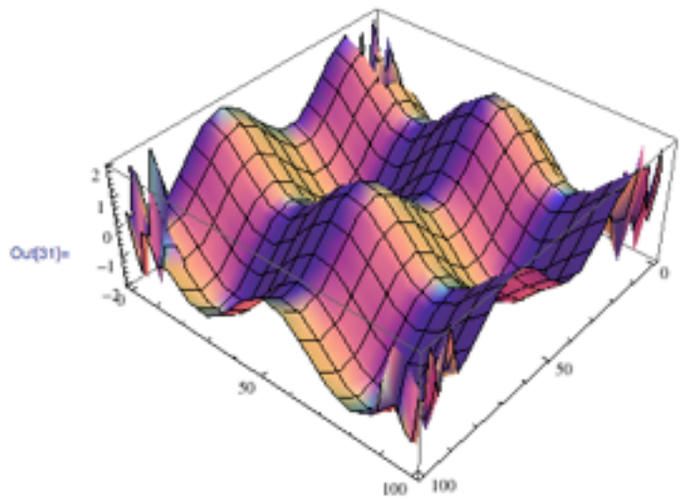
Plots of data

```
In[18]:= ListPlot[data]
```



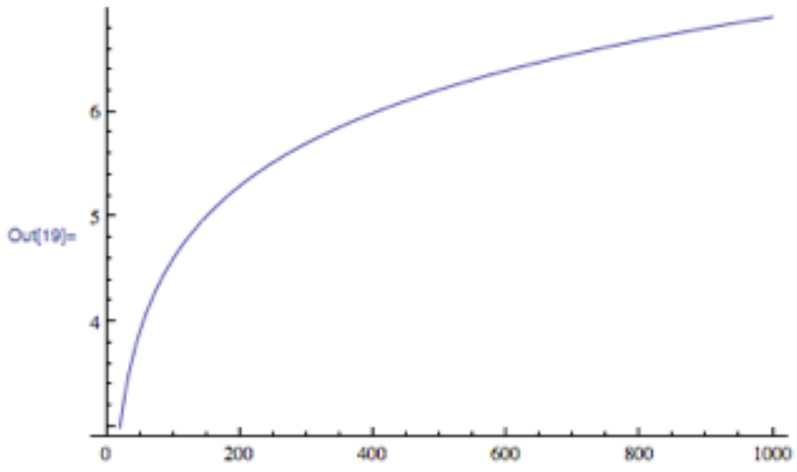
Three-dimensional plots

```
In[31]:= Plot3D[{Sin[x] + Cos[y]}, {x, 0, 100}, {y, 0, 100}]
```



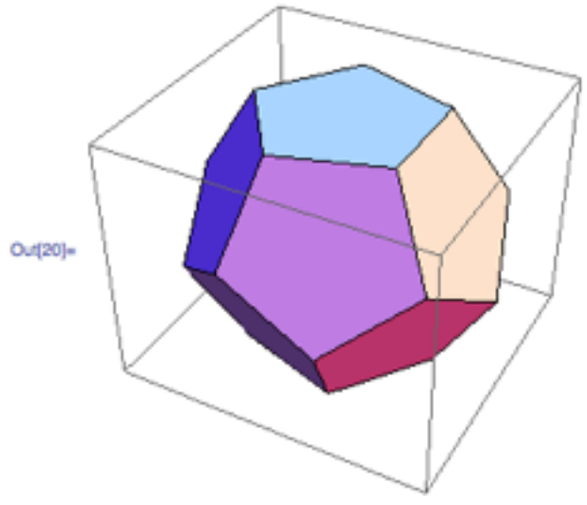
Plots of functions

```
In[19]:= Plot[Log[x], {x, 0, 1000}]
```



Specialized objects

```
In[20]:= PolyhedronData["Dodecahedron"]
```



Kernels and Notebooks

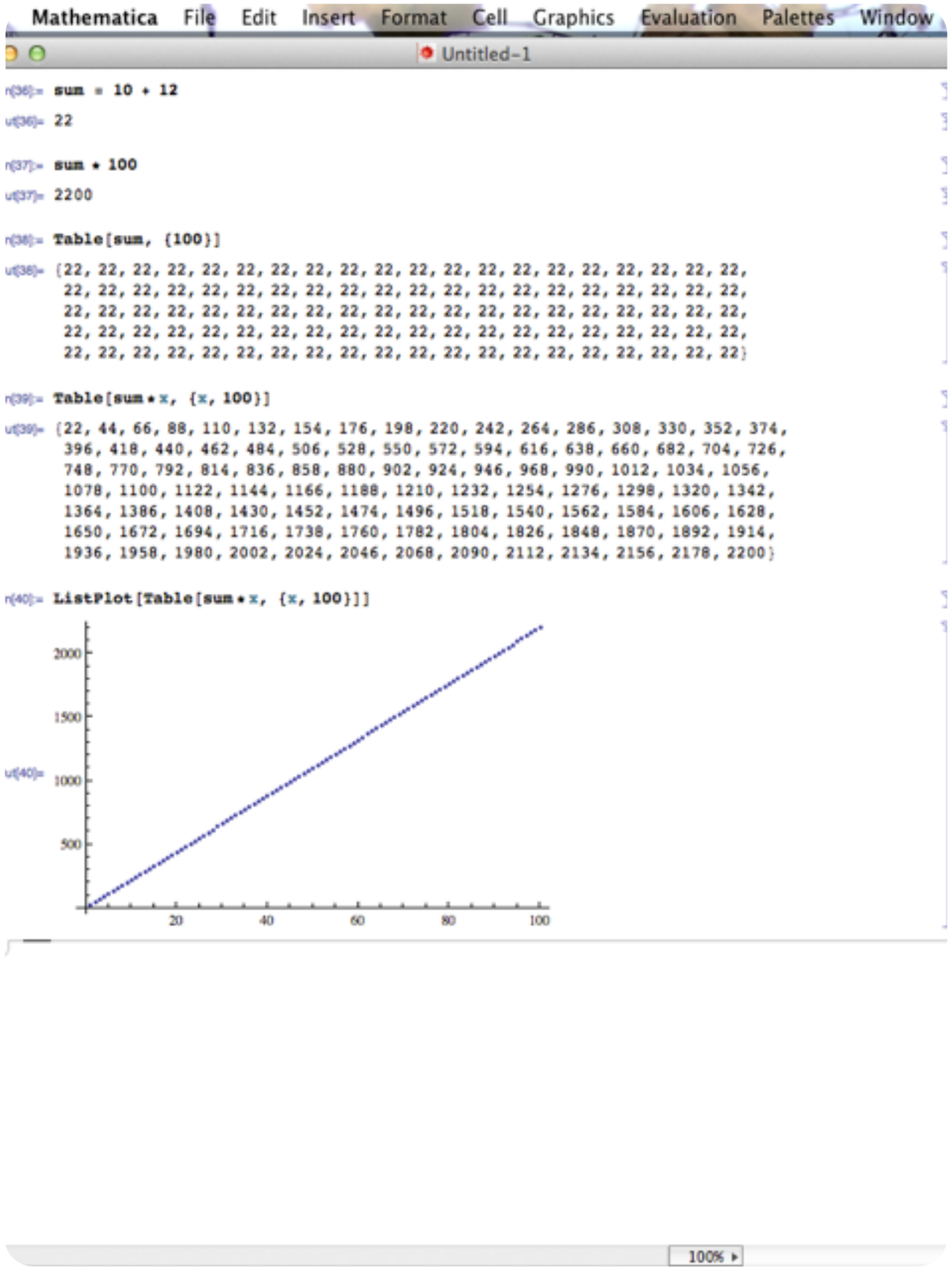
Mathematica has two components, the *kernel* and the *notebook*

The *kernel* is the invisible part of the program that does all the calculations

The *notebook* is the main user interface, its purpose is to allow you to perform analyses and to save them for re-use or for later reference

You can work with many notebooks at once. They share information between them because they interface with the same kernel

For advanced work you can work with two kernels, which allows you to run two sets of calculations in different notebooks at the same time



A typical Mathematica notebook

Notebooks and cells

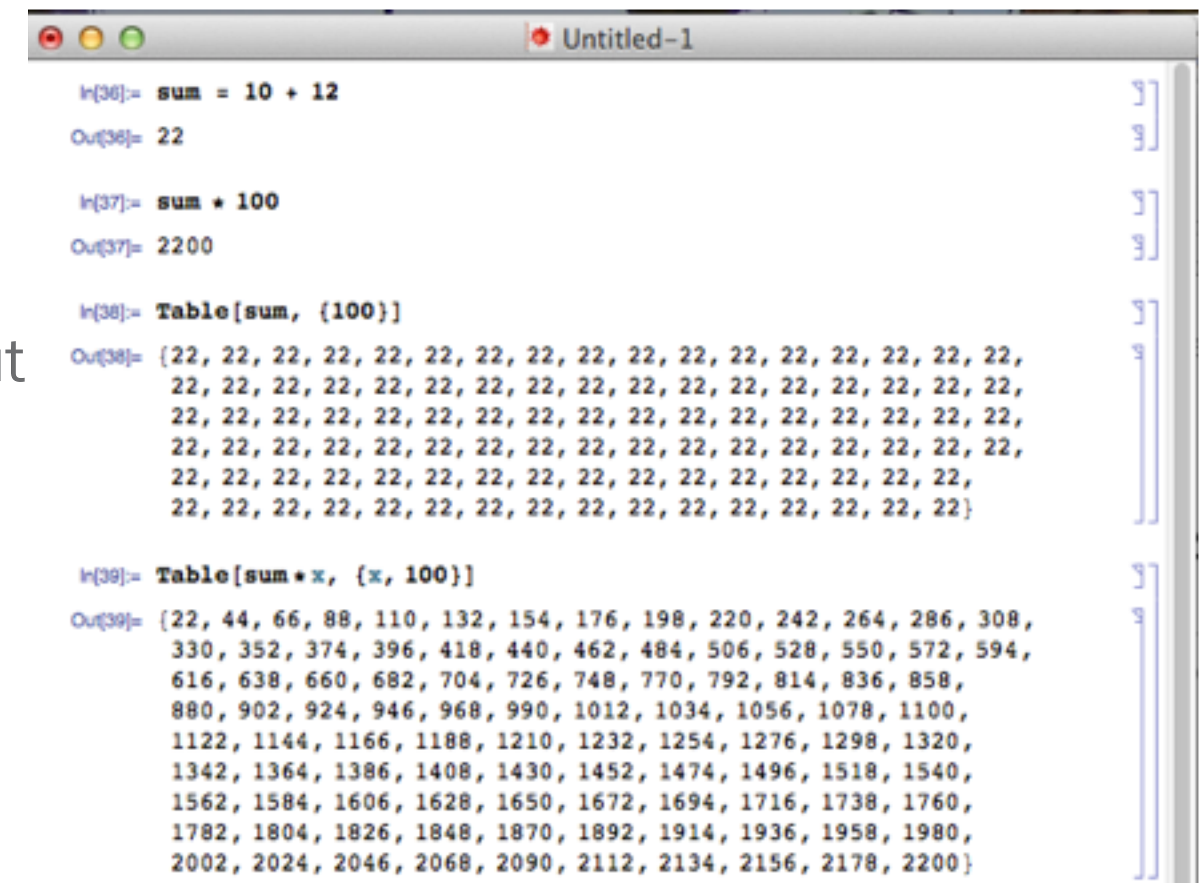
Notebooks are organized into *cells*

Default cells are for calculations, with input entered by you followed by output created by the kernel

Cells must be executed to obtain output: Shift + Enter to execute

Cells may be executed more than once, and the input can be changed between executions

Brackets in the right margin show cell boundaries and distinguish between input and output



Uses for brackets:

1. monitor calculations (bracket is highlighted while the kernel is executing)
2. select entire cell for deletion
3. hide output by double clicking

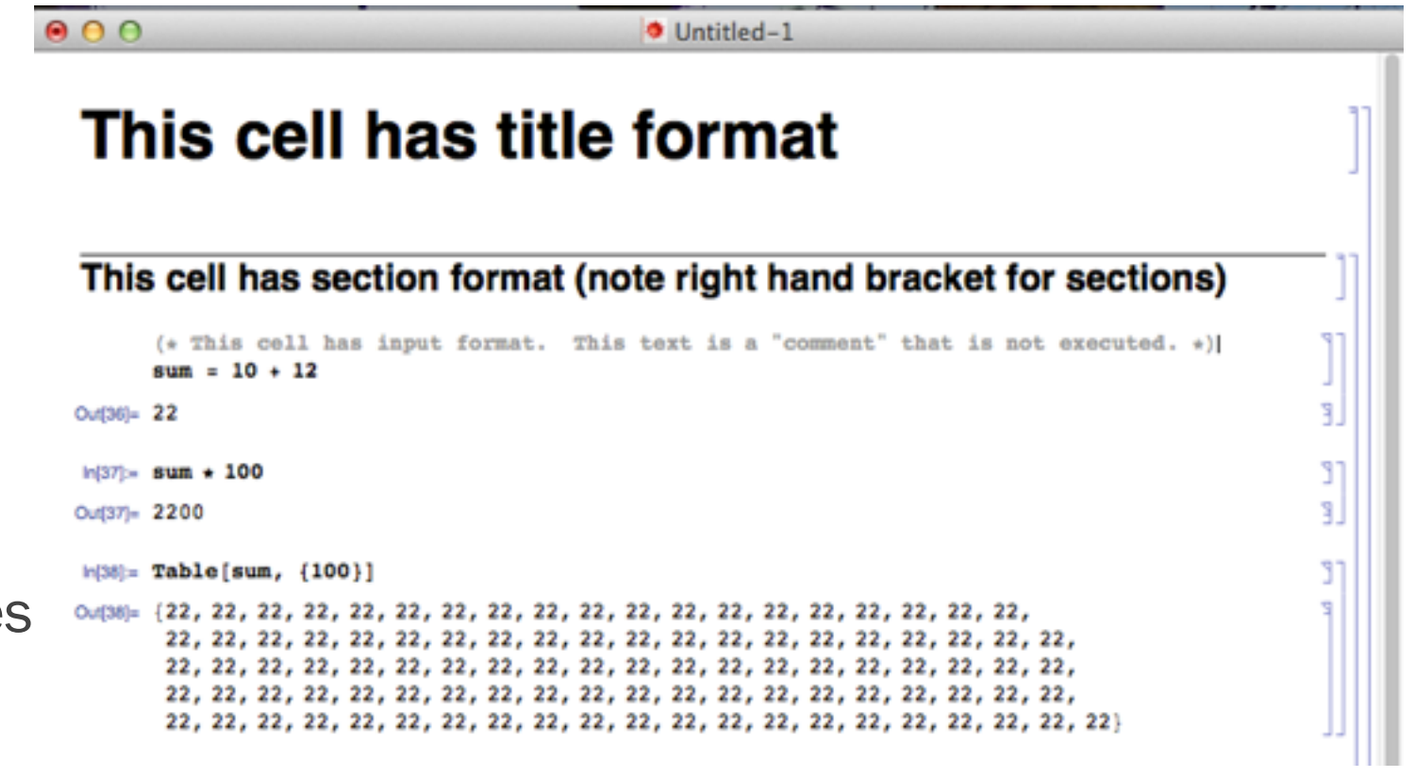
Formatting notebooks

Notebooks can be formatted like a word processor document

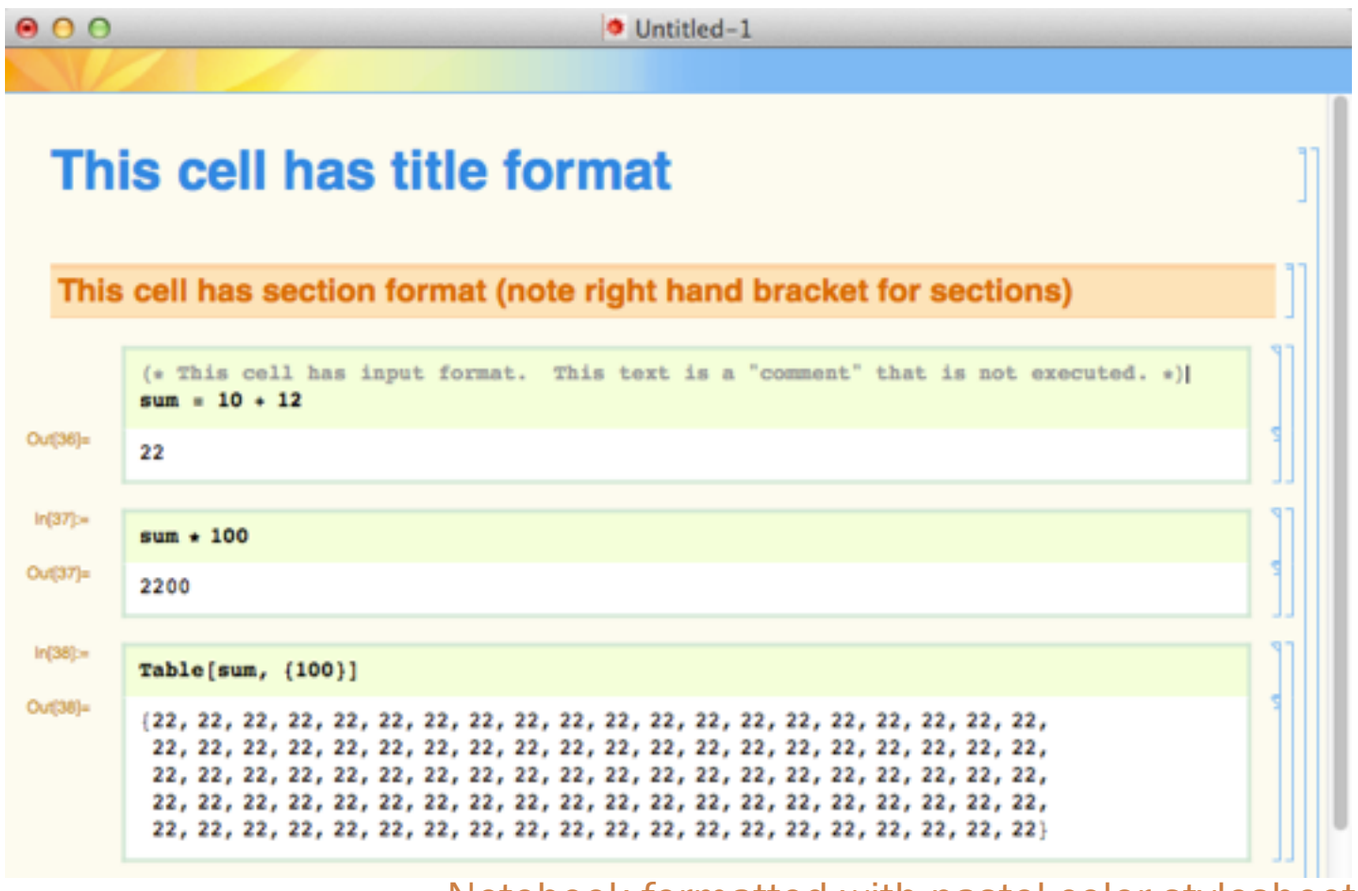
Individual cells can be formatted as titles text, section headings, or input (input is the default)

Use Format | Style menu to format individual cells

Use Format | Stylesheet menu to format the whole notebook



Notebook formatted with default stylesheet



Notebook formatted with pastel color stylesheet

Functions

Functions are key to *Mathematica:* functions receive information or data, process it, and return a result

Functions are called by their name, usually composed of complete English words describing what the function does, with no spaces and first letters capitalized

Function names are followed by square brackets, in which one or more arguments is entered:

FunctionName[argument]

For example, the *ListPlot[]* function takes a matrix of x,y values as its argument:

ListPlot[{{1,2},{3,4}}]

Mathematica's help files give descriptions and examples of every function

Options for functions

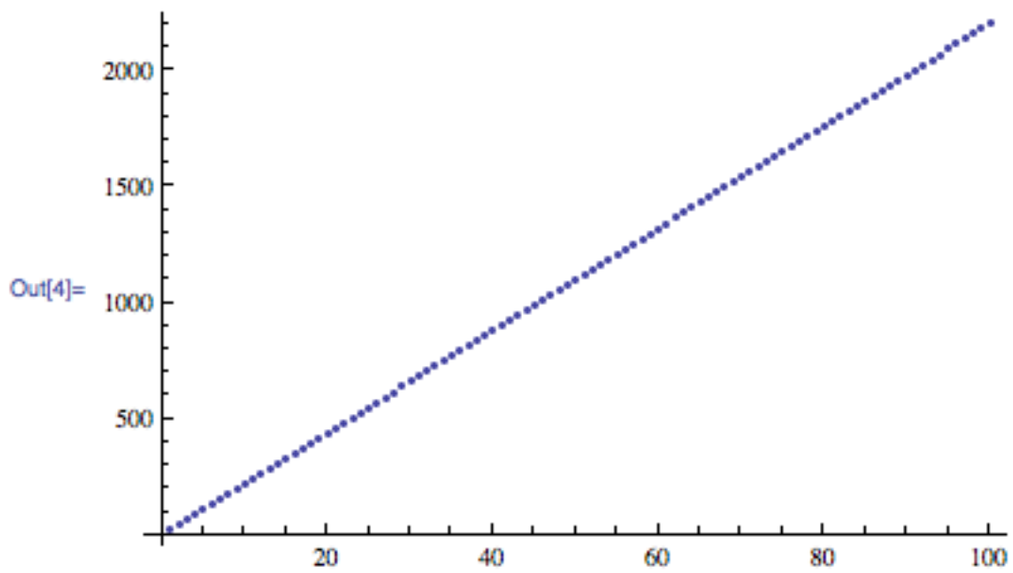
Many functions have options that are entered as arguments

Options usually have the format *OptionName -> Value*

Find options with *Options[FunctionName]* or in Documentation Center

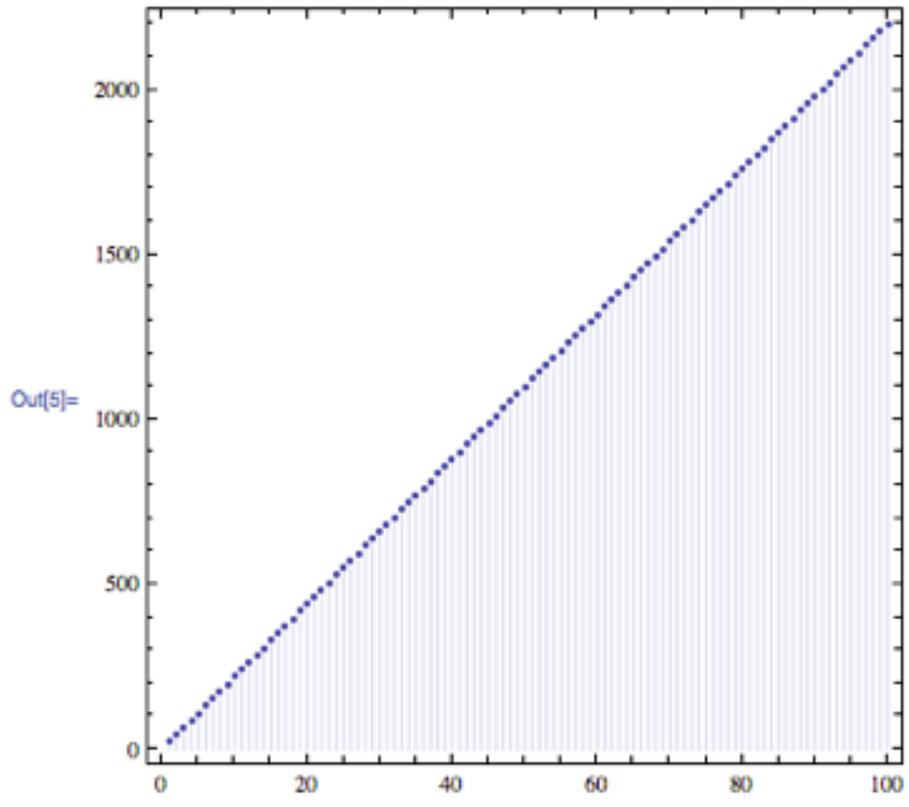
Listplot with no options

```
In[4]:= ListPlot[data]
```



Listplot with three options

```
In[5]:= ListPlot[data, Frame → True, AspectRatio → 1, Filling → Axis]
```



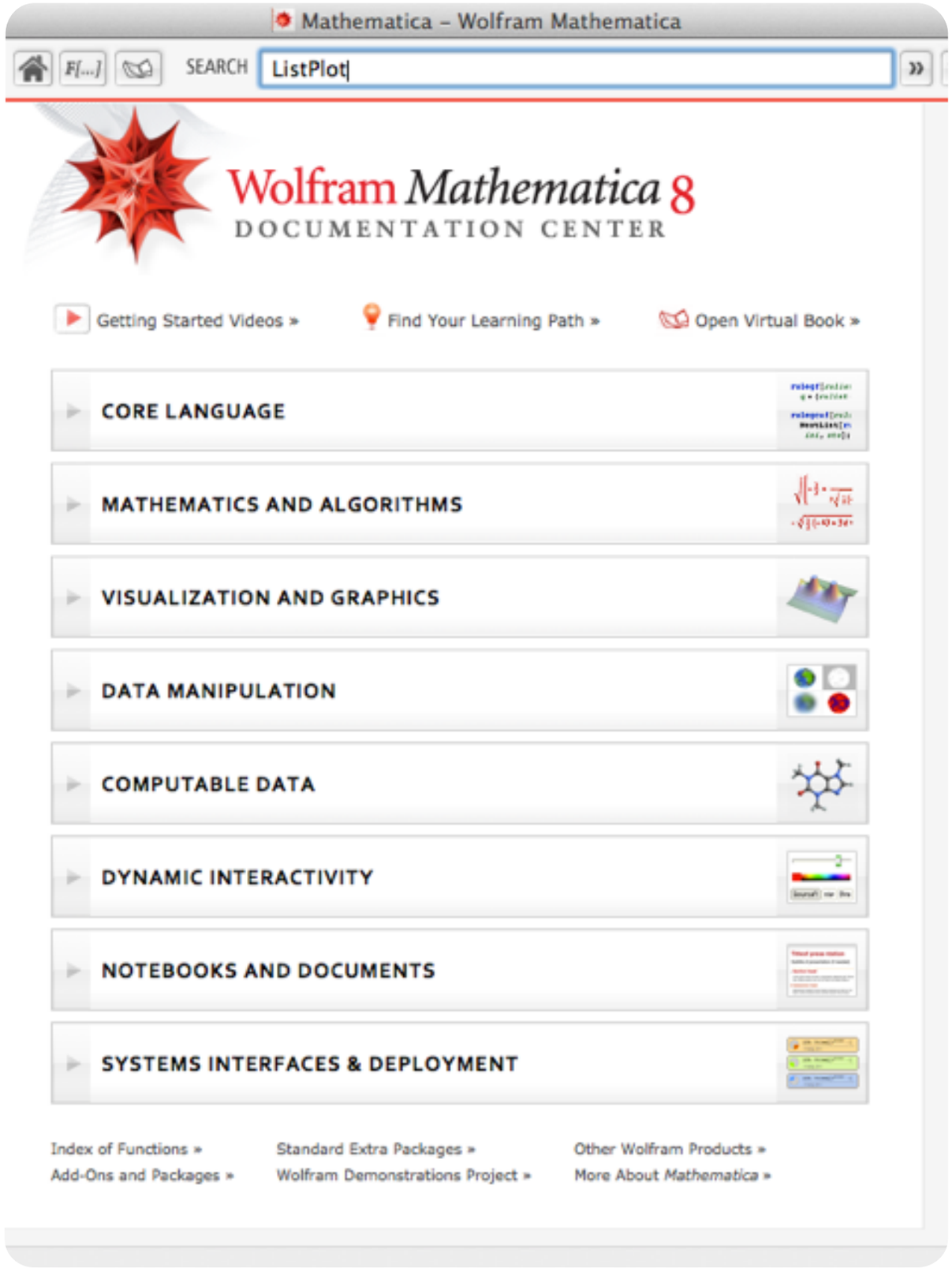
Getting help

Mathematica help files can be browsed or searched from the Documentation Center of the Help menu

Function names are always made of up complete words, no spaces, with the first word capitalized

Search for functions you hope exist: “*Histogram*”, “*LinearRegression*”, “*PrincipalComponents*”, “*GenomeData*”

Note Function Browser and Mathematica Book help buttons at top left of the Documentation Center



Variables

Variables are also key to *Mathematica*, allowing you to store information

Variables do not have brackets or options

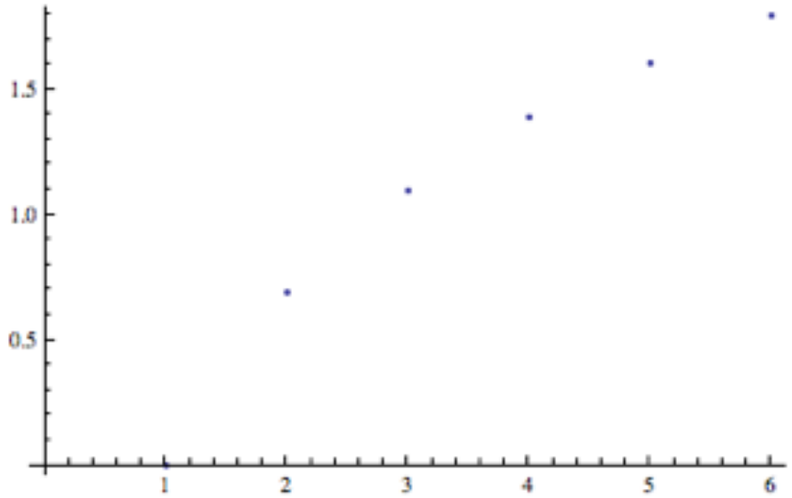
You create variables, giving them a name and putting something into them

Here a variable called *data* is used to store a number, a sequence of numbers, the natural log of a sequence of numbers, and data imported from an Excel file. A variable called *mygraph* is used to store a graphic

You can retrieve what is inside a variable by executing it (the graph is displayed again by executing *mygraph*)

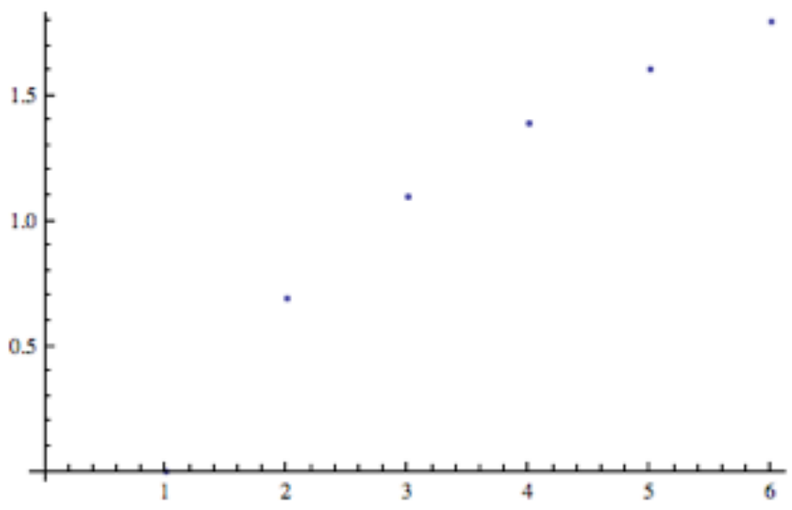
```
In[12]:= data = 1
Out[12]= 1

In[13]:= data = {1, 2, 3, 4, 5, 6}
Out[13]= {1, 2, 3, 4, 5, 6}

In[14]:= data = Log[{1, 2, 3, 4, 5, 6}]
Out[14]= {0, Log[2], Log[3], Log[4], Log[5], Log[6]}

In[15]:= mygraph = ListPlot[data]
```

Out[15]=




```
data = Import["/Users/pdavidpolly/Documents/Stat Data.xls"];

In[16]:= mygraph
```

Out[16]=



Parts of variables

When a variable has more than one item stored, you can get specific parts using double square brackets after the variable name

data returns all the items in *data*

data[[1]] returns only the first item in *data*

data[[1;;3]] returns items 1 to 3

For more examples look at the Documentation Center under the function *Part[]* and under the tutorial *GettingPiecesOfLists*

```
In[22]:= data = {10, 20, 30, 40, 50, 60}
Out[22]= {10, 20, 30, 40, 50, 60}

In[23]:= data
Out[23]= {10, 20, 30, 40, 50, 60}

In[24]:= data[[1]]
Out[24]= 10

In[25]:= data[[3 ;; 5]]
Out[25]= {30, 40, 50}
```

Lists, Matrices, and other Multidimensional data

You will often work with “lists”, which is Mathematica’s term for any group of several items

Some lists have only one element (scalar), some have a long row of elements (vector), some have columns and rows of data (matrix or array)

```
In[12]:= data = 1
Out[12]= 1

In[26]:= data = {10, 20, 30, 40, 50, 60}
Out[26]= {10, 20, 30, 40, 50, 60}

In[27]:= data = {{1, 2}, {1, 3}, {3, 1}}
Out[27]= {{1, 2}, {1, 3}, {3, 1}}

In[28]:= data = {{{1, 2}, {1, 3}, {3, 1}}, {{1, 2}, {1, 3}, {3, 1}}}
Out[28]= {{{1, 2}, {1, 3}, {3, 1}}, {{1, 2}, {1, 3}, {3, 1}}}
```

You can get columns, rows, or elements from the list using the double square bracket system

```
In[30]:= data[[1, 2 ;; 3]]
Out[30]= {{1, 3}, {3, 1}}
```

See Documentation Center under:

1. ListsOverview
2. HandlingArraysOfData

Special formatting tags

You can control the display of output in many ways by putting special tags at the end of a line of input

semicolon (;) prevents output from being displayed

//N forces numbers to be displayed in decimal form

//MatrixForm displays tables of data in rows and columns

```
In[34]:= data = {10, 20, 30, 40, 50, 60}
Out[34]= {10, 20, 30, 40, 50, 60}

In[36]:= data
Out[36]= {10, 20, 30, 40, 50, 60}

In[37]:= data;

In[38]:= data = Log[{1, 2, 3, 4, 5, 6}]
Out[38]= {0, Log[2], Log[3], Log[4], Log[5], Log[6]}

In[39]:= data
Out[39]= {0, Log[2], Log[3], Log[4], Log[5], Log[6]}

In[40]:= data // N
Out[40]= {0., 0.693147, 1.09861, 1.38629, 1.60944, 1.79176}

In[41]:= data = {{1, 2}, {1, 3}, {3, 1}}
Out[41]= {{1, 2}, {1, 3}, {3, 1}}

In[42]:= data
Out[42]= {{1, 2}, {1, 3}, {3, 1}}

In[43]:= data // MatrixForm
Out[43]//MatrixForm=
1  2
1  3
3  1
```

Importing and exporting data

Mathematica has an extensive range of file types that can be imported and exported: text files, Excel files, Word files, PDFs, Illustrator, JPEG, etc.

Import[FilePath]

Export[FilePath, "type"]

Note the helpful file path chooser found on the Insert menu

```
In[49]:= data =
           Import[
            "/Users/pdavidpolly/Documents/Lectures/G562 Geometric Morphometrics/Sample
              Data/CaumulAndPolly2005.xls"];

In[50]:= Export[
          "/Users/pdavidpolly/Documents/Lectures/G562 Geometric Morphometrics/mygraph.pdf",
          "PDF"]

Out[50]= /Users/pdavidpolly/Documents/Lectures/G562 Geometric Morphometrics/mygraph.pdf

In[51]:= mypic =
           Import[
            "/Users/pdavidpolly/Documents/Lectures/G562 Geometric Morphometrics/Example
              Images/JPG/DSCN4141.JPG"];

In[52]:= mypic

Out[52]=
```

Simple graphics

ListPlot[]

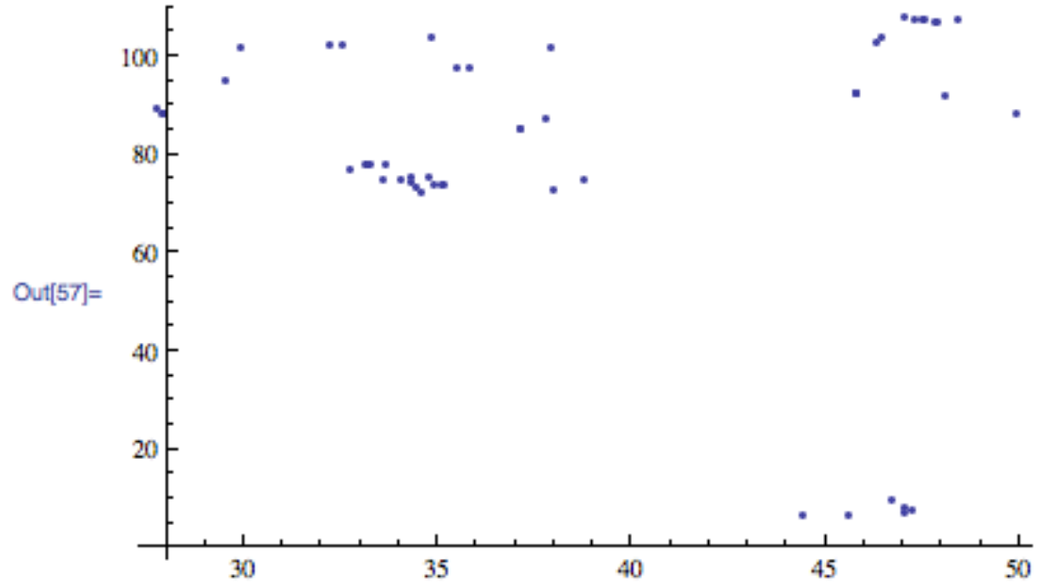
Plot[]

Histogram[]

BarChart[]

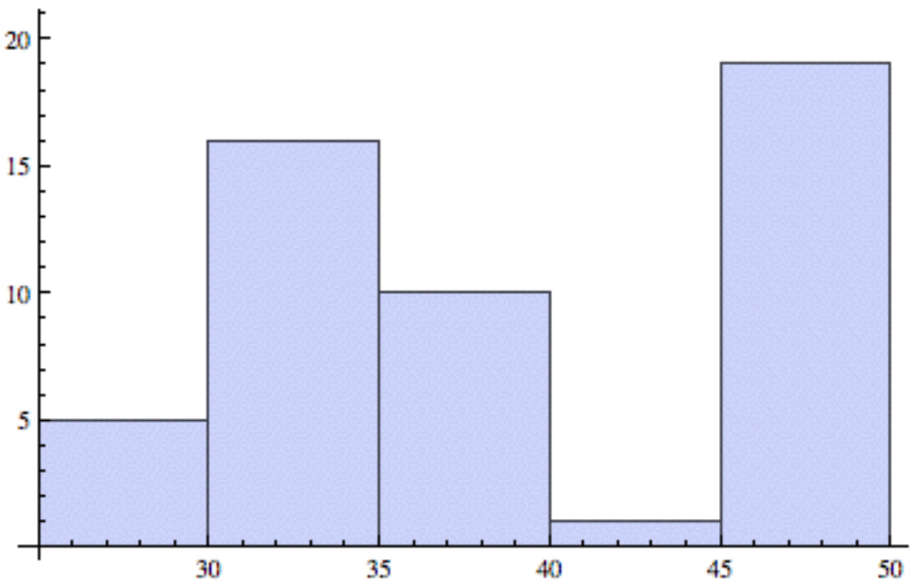
```
In[57]:= ListPlot[data[[1, 2 ;;, 6 ;; 7]]]
```

Out[57]=




```
In[58]:= Histogram[data[[1, 2 ;;, 6]]]
```

Out[58]=



Loops: programming structure for repeating things

Use Table[], Map[], or Do[] to carry out repeated tasks

Table[*lines to be repeated* , *{iterator}]*

where the lines to be repeated consist of other Mathematica functions or lists of functions separated by semicolons

iterator is a special construction that creates a temporary counting variable and specifies number of times to repeat

Simple: {10} (repeats 10 times)
With variable: {x,10} (repeats while incrementing x from 1 to 10 in steps of 1)
Full: {x,1,10,1} (repeats while incrementing x from 1 to 10 in steps of 1)
Full: {x,10,2,-2} (repeats while incrementing x backward from 10 to 2 in steps of 2)

```
In[2]:= Table["hello", {10}]
Out[2]= {hello, hello, hello, hello, hello, hello, hello, hello, hello, hello}

In[3]:= Table[x, {x, 10}]
Out[3]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In[5]:= Table[x, {x, 1, 10, 1}]
Out[5]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In[6]:= Table[x, {x, 10, 1, -1}]
Out[6]= {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

In[7]:= Table[x, {x, 10, 1, -2}]
Out[7]= {10, 8, 6, 4, 2}

In[10]:= min = 7;
         max = 14;
         bin = 2;
         Table[x, {x, min, max, bin}]
Out[13]= {7, 9, 11, 13}
```

Conditional statements

Is equal?	==
Is unequal?	!=
Greater than?	>
Less than?	<
And	&&
Or	||

If[statement is true, then this, or else this]

myage = 65.5;
If[myage > 50, Print[“my age is older”], Print[“my age is not older”]

If[myage > 55 && myage < 65, Print[“my age is in the bin”], Print[“my age is outside the bin”]

Working with Strings

Strings are entities of characters, as opposed to numbers. You can manipulate strings in Mathematica as well as numbers. For example:

mytext = “Species”;

You can combine strings by joining them with the StringJoin[] function or <> (which does the same thing):

```
In[16]:= StringJoin[mytext, " Name"]
Out[16]= Species Name
```

```
In[17]:= mytext <> " Name"
Out[17]= Species Name
```

You can create a list of labels using Table[] and ToString[], the latter of which converts numbers to strings so they can be joined to other strings:

```
In[18]:= Table[mytext <> " " <> ToString[x], {x, 5}]
Out[18]= {Species 1, Species 2, Species 3, Species 4, Species 5}
```

mySQL and Mathematica

Functions for interacting with SQL server are found in the DatabaseLink package.

First, load the package:

```
<<DatabaseLink`
```

Next, set up a database link and give it a name (“conn” in this example):

```
conn = OpenSQLConnection[]
```

This function will open a Connection Tool window, where you can create a new connection to your mySQL database. Click “New” and follow the steps to create the connection:

1. give it a short name, e.g. ‘mySQL’
2. specify System Level;
3. choose “MySQL(Connector/J)” from the database type menu;
4. Specify “localhost” or “129.0.0.1”, port 3306, username “root”, password for your mySQL (probably “root” by default). [test it to make sure it works];
5. store password if you wish

Close the connection again…

```
CloseSQLConnection[conn]
```

Using your connection again

Once you have created a connection, you can use it again without going through the configuration steps:

```
In[22]:= conn = OpenSQLConnection["mySQL", Catalog → "Felidae", Username → "root",
           Password → "root"]
         CloseSQLConnection[conn]

Out[22]= SQLConnection[mySQL, 2, Open, Catalog → Felidae]

Out[23]= {}
```

This opens the database (aka, “Catalog”) named “Felidae” using the stored connection named “mySQL”.

Put semicolons at the end of the line to suppress the output:

```
In[24]:= conn = OpenSQLConnection["mySQL", Catalog → "Felidae", Username → "root",
           Password → "root"];
         CloseSQLConnection[conn];
```

Always close the connection when you are finished with it.

Doing something with your connection

There's no use opening a database connection, unless you do something with it. We'll use the connections to load data into Mathematica using an SQL query. The SQLExecute[] function allows you to send an SQL statement to the database and get back the results. Store the results in a variable so you can use them:

```
In[32]:= conn = OpenSQLConnection["mySQL", Catalog → "Felidae", Username → "root",
            Password → "root"];
          data = SQLExecute[conn, "Select * FROM Occurrences;"];
          CloseSQLConnection[conn];
```

The entire occurrences table should now be stored in the variable data. We can check the first line to make sure:

```
In[35]:= data[[1]]

Out[35]= {1, 11797, Carnivora, Felidae, Vishnufelis, sp., H. O'Regan, Pilgrim, 1932, 4195,
           Ramanagar, India, Jammu and Kashmir, Ramanagar, 32.8167, 75.366667000000010,
           Neogene, Miocene, Null, Null, Null, 10., 10., 9.99999999999999999, Chinji, Null}
```

Getting labels or specialized tables

```
In[41]:= conn = OpenSQLConnection["mySQL", Catalog → "Felidae", Username → "root",
            Password → "root"];
          GenusLabels = SQLExecute[conn, "Select DISTINCT Genus FROM Occurrences;"];
          CloseSQLConnection[conn];

In[44]:= GenusLabels

Out[44]= {{Vishnufelis}, {Machairodus}, {Felis}, {Metailurus}, {Panthera},
          {Lynx}, {Miracinonyx}, {Puma}, {Megantereon}, {Vinayakia}, {Leopardus},
          {Pseudaelurus}, {Nimravides}, {Adelphailurus}, {Pratifelis}, {Homotherium},
          {Smilodon}, {Dinofelis}, {Xenosmilus}, {Caracal}, {Acinonyx},
          {Therailurus}, {Paramachaerodus}, {Miomachairodus}, {Epimachairodus},
          {Viretailurus}, {Herpailurus}, {Smilodontidion}, {Neofelis}, {Prionailurus}}

In[45]:= Flatten[GenusLabels]

Out[45]= {Vishnufelis, Machairodus, Felis, Metailurus, Panthera, Lynx, Miracinonyx, Puma,
          Megantereon, Vinayakia, Leopardus, Pseudaelurus, Nimravides, Adelphailurus,
          Pratifelis, Homotherium, Smilodon, Dinofelis, Xenosmilus, Caracal,
          Acinonyx, Therailurus, Paramachaerodus, Miomachairodus, Epimachairodus,
          Viretailurus, Herpailurus, Smilodontidion, Neofelis, Prionailurus}
```

Note that by default each line of data comes back as a subgroup defined by curly brackets. The subgroups are useful when several variables are found in each line, but annoying when you just want a list of names. Flatten[] gets rid of them.