# Introduction to Mathematica Workshop

Please register your attendance.

(You could win a cup of coffee from De Fer.)

https://shorturl.at/bZeYE

# Introduction to Mathematica Workshop

# Learning Objectives

- Learn what Mathematica and the Wolfram Language are
- Gain familiarity (hands on) with:
  - Notebook Environment
  - Wolfram Language Syntax
  - Basic WL functions and expressions

## What to keep in mind during this session

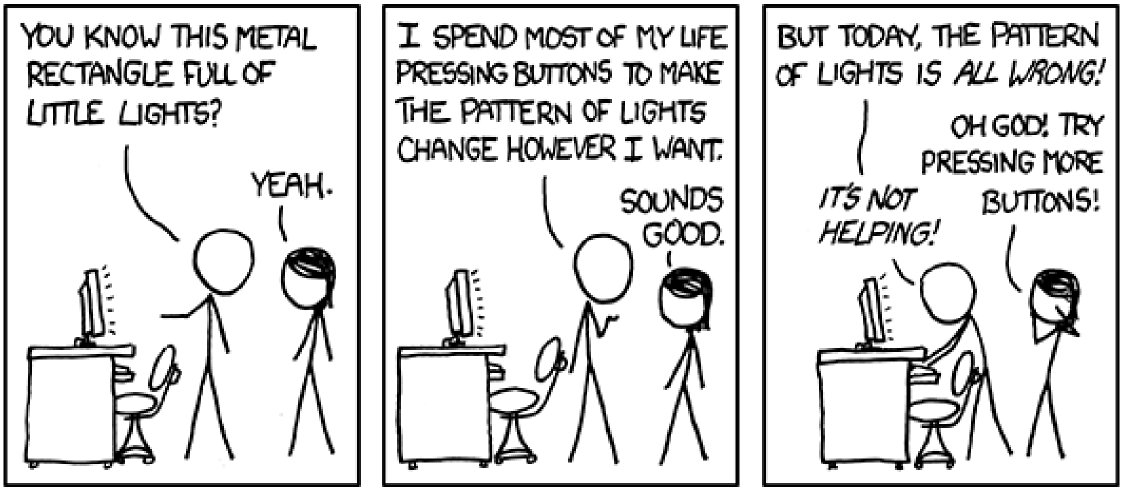
- **Learn by doing**: try to follow and type commands and instructions along with the speaker.
- **Ask questions!** : Don't hesitate to ask any question you have or any issues you are going through while working on your Notebook
- **Try stuff yourself** : During the session, if you have time to explore on your own, try other commands/functions/expressions, **we encourage you to be curious!**

# What is Mathematica?

- **Software** with **built-in libraries** developed for several areas of **technical computing** (ML, Statistics, Symbolic Computation, data manipulation, network analysis, time series analysis, NLP...)
- The **Wolfram Language** is the **programming language** used in *Mathematica*

## What is a programming language?

System of notation for writing computer programs (communicating tasks to the computer)


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

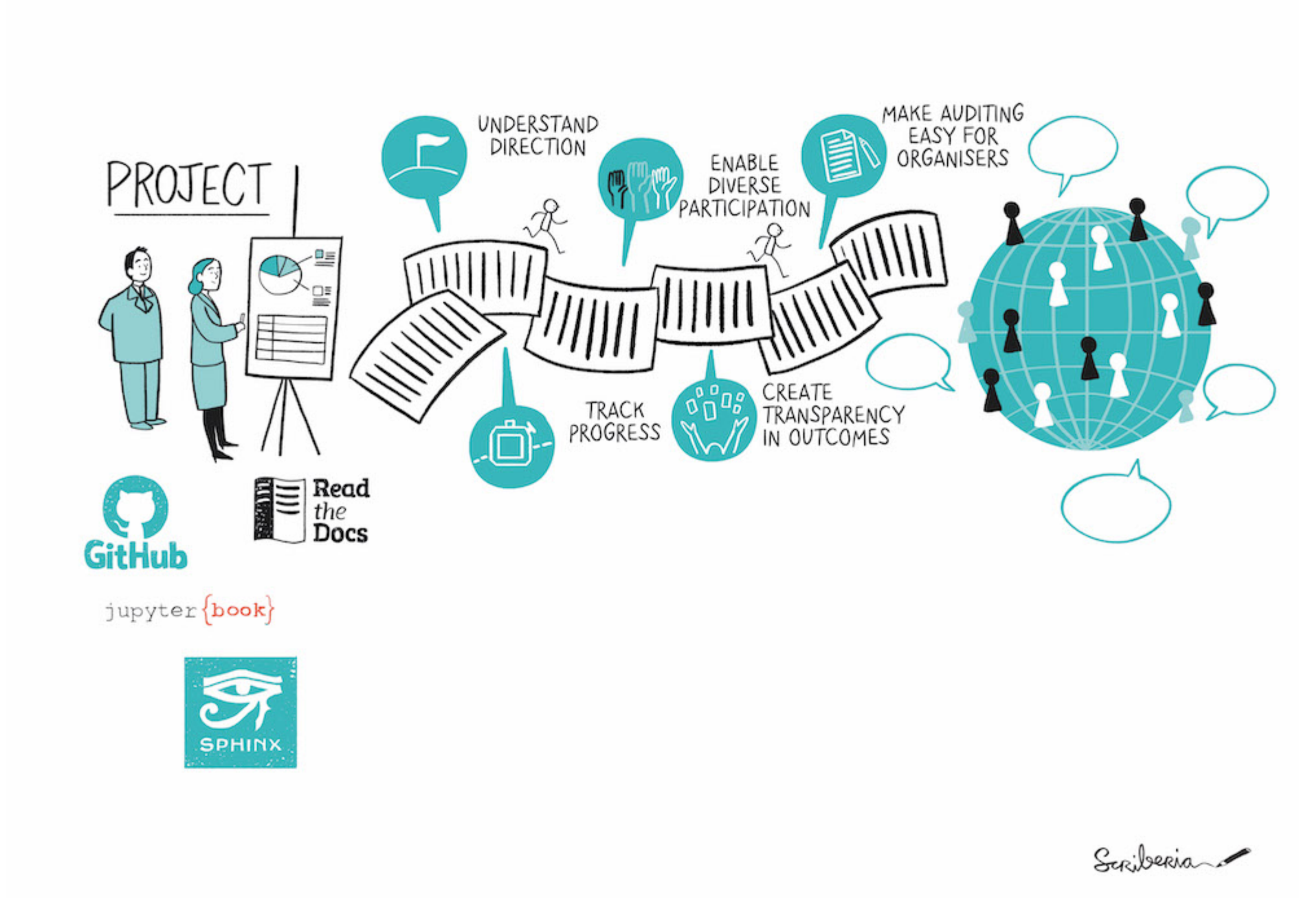
*source:https://xkcd.com/722/

## Command Line Interface (CLI) and Notebook Environment

| | Command Line Interface | Notebook Environment |
|---|---|---|
| Execution | Line-by-line or script | Cell-based |
| Interactivity | Limited, primarily text-based | Highly interactive with in-line visualizations |
| Code organization | Linear, script files | Nonlinear, mix of code and documentation |
| Use cases | scripts, automation, pipelines... | Experimentation, data analysis and reporting |

# The Notebook Environment as Laboratory Notebook

- **Organizes our record of research**
- **Documents hypothesis, experiments and results**
- **Focuses on reproducibility, having a consistent format**


PROJECT
UNDERSTAND DIRECTION
ENABLE DIVERSE PARTICIPATION
MAKE AUDITING EASY FOR ORGANISERS
TRACK PROGRESS
CREATE TRANSPARENCY IN OUTCOMES
GitHub
Read the Docs
jupyter{book}
SPHINX
Scriberia

# Time to open Mathematica!

# Working with Mathematica's Notebook Environment

## Cell Types

Notebooks are organized into **cells** in a hierarchical manner. Different cell types serve different purposes:

- **Section cells**: For organizing content
- **Text cells**: For adding explanatory text
- **Input cells**: For entering Mathematica code
- **Output cells**: Display results of evaluations

To change a cell type, use the Format menu or keyboard shortcuts.

## Evaluation

To evaluate a cell, place your cursor in it and press Shift+Enter. To evaluate the entire notebook, use Evaluation > Evaluate Notebook.

## Punctuation symbols

| Punctuation | Usage | Example |
|---|---|---|
| Square Brackets `[]` | - Calling functions<br>- Typing Constellation Object<br>- Download 'currying' syntax (Constellation queries) | `ExperimentHPLC[mySample]`<br>`Object[Instrument, LiquidHandler, "Johnny 5"]` |
| Curly braces `{}` | Defining lists | `{1,2,3}` |
| Quotation marks `" "` | Defining strings | `"This is a string"` |
| Double square brackets `[[]]` | Shortcut syntax to extract values from lists | `MyList[[3]]` |
| Parenthesis `()` | Grouping commands to define order of operations | `1/ (2 + 3)` |

# Data Types

## Strings

```
(* String creation *)
stringExample = "Hello, World!"

(* String length *)
StringLength[stringExample]
(* Output: 13 *)

(* String extraction *)
StringTake[stringExample, 5]
(* Output: "Hello" *)

StringTake[stringExample, -6]
(* Output: "World!" *)

(* String joining *)
StringJoin["Hello, ", "World!"]
(* Output: "Hello, World!" *)
(* Alternative joining syntax *)
"Hello" <> ", " <> "World!"
(* Output: "Hello, World!" *)

(* String splitting *)
StringSplit["apple,banana,cherry", ","]
(* Output: {"apple", "banana", "cherry"} *)

(* String replacement *)
StringReplace["The cat and the dog", "the" -> "a"]
(* Output: "The cat and a dog" *)

(* Case manipulation *)
ToUpperCase["hello"]
(* Output: "HELLO" *)

ToLowerCase["WORLD"]
(* Output: "world" *)

Capitalize["hello world"]
(* Output: "Hello World" *)

(* String patterns and regular expressions *)
StringCases["The rain in Spain", RegularExpression["\\w*ai\\w*"]]
(* Output: {"rain", "Spain"} *)

(* String position *)
StringPosition["Mississippi", "ss"]
(* Output: {{3, 4}, {6, 7}} *)

(* String counting *)
StringCount["Mississippi", "s"]
(* Output: 4 *)

(* String padding *)
StringPadLeft["123", 6, "0"]
(* Output: "000123" *)

StringPadRight["123", 6, "0"]
(* Output: "123000" *)

(* String trimming *)
StringTrim["  Hello, World!  "]
(* Output: "Hello, World!" *)

(* String comparison *)
StringEqual["hello", "Hello"]
(* Output: False *)
```

```
StringContainsQ["Hello, World!", "World"]
(* Output: True *)

(* String conversion *)
ToString[42]
(* Output: "42" *)

ToExpression["2 + 2"]
(* Output: 4 *)
```

## Numerical types

```
(* Integers *)
integerExample = 42

(* Rational numbers *)
rationalExample = 2/3

(* Real numbers (arbitrary precision) *)
realExample = 3.14159265358979323846

(* Complex numbers *)
complexExample = 2 + 3I
```

## Basic Operations

```
(* Addition *)
2 + 3
(* Output: 5 *)

(* Subtraction *)
10 - 7
(* Output: 3 *)

(* Multiplication *)
4 * 5
(* Output: 20 *)

(* Division *)
15 / 3
(* Output: 5 *)
(* Exponentiation *)
2^3
(* Output: 8 *)
```

## Working with variables/expressions

```
(* Assign a value to a variable *)
x = 5

(* Use the variable in calculations *)
y = x * 2
(* y is now 10 *)

(* Clear a variable *)
Clear[x]

(* Define a function *)
myFunction := x^2
```

## Inspecting expressions and functions

We can use the character `?` before a function to inspect it or the `Information` function.

```
?Function
(* OR *)
Information[Function]

(* To inspect the type of the expression *)
Head[Expression]
```

## Data Structures

### Lists

Lists are the most fundamental data structure in Mathematica. They can contain any type of data and are created using curly braces.

```
(* A simple list *)
simpleList = {1, 2, 3, 4, 5}

(* A nested list *)
nestedList = {{1, 2}, {3, 4}, {5, 6}}
```

```
(* A list with mixed data types *)
mixedList = {1, "hello", 3.14, True}
```

## Inspecting, Slicing, and Extracting Information from Lists (Arrays)

In Mathematica, lists serve as the primary data structure for representing arrays. Here's how you can inspect, slice, and extract information from them:

### Inspecting Lists

Length of a list:

```
list = {1, 2, 3, 4, 5};
Length[list]
(* Output: 5 *)
```

Dimensions of a nested list:

```
nestedList = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
Dimensions[nestedList]
(* Output: {3, 3} *)
```

Information about a list:

```
Information[list]
(* Shows type and other details about the list *)
```

### Slicing Lists

Extracting a single element (indexing starts at 1):

```
list[[3]]
(* Output: 3 *)

(* To extract the last element *)
list[[-1]]
(* Output: 5 *)
```

Extracting a range of elements:

```
list[[2;;4]]
(* Output: {2, 3, 4} *)
```

Extracting elements with a step:

```
list[[1;;-1;;2]]
(* Output: {1, 3, 5} *)
```

Extracting from a nested list:

```
nestedList[[2, 3]]
(* Output: 6 *)
```

## Extracting Information from Lists

First and last elements:

```
First[list]
(* Output: 1 *)

Last[list]
(* Output: 5 *)
```

Take first or last n elements:

```
Take[list, 3]
(* Output: {1, 2, 3} *)

TakeLargest[list, 2]
(* Output: {5, 4} *)
```

Select elements based on a condition:

```
Select[list, EvenQ]
(* Output: {2, 4} *)
```

Apply a function to each element:

```
Map[#^2 &, list]
(* Output: {1, 4, 9, 16, 25} *)
```

```
Map[Function[x, x^2], List]
(* Output: {1, 4, 9, 16, 25} *)
```

Transpose a nested list:

```
nestedList = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
Transpose[nestedList]
(* Output: {{1, 4, 7}, {2, 5, 8}, {3, 6, 9}} *)
```

## Associations

Associations are similar to dictionaries in other languages. They store key-value pairs.

```
assoc = <|"name" -> "John", "age" -> 30, "city" -> "New York"|>

(* Accessing values *)
assoc["name"]
```

# Loops and List Generation

Mathematica offers several ways to create loops and generate lists. While traditional loops exist, Mathematica often uses functional programming constructs that are efficient and idiomatic.

## Traditional Loops

### For Loop

```
For[i = 1, i <= 5, i++,
  Print[i]
]
```

### While Loop

```
i = 1;
While[i <= 5,
  Print[i]; i++
]
```

### Do Loop

```
Do[
  Print[i],
  {i, 1, 5}
]
```

## Functional Approaches to Looping

### Table

`Table` is often used instead of traditional loops to generate lists:

```
(* Generate a list of squares *)
squaresList = Table[i^2, {i, 1, 5}]
(* Output: {1, 4, 9, 16, 25} *)

(* Generate a 2D list *)
matrix = Table[i + j, {i, 1, 3}, {j, 1, 3}]
(* Output: {{2, 3, 4}, {3, 4, 5}, {4, 5, 6}} *)
```

### Range

`Range` is useful for creating lists of evenly spaced numbers:

```
(* Create a list from 1 to 10 *)
Range[10]
(* Output: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} *)

(* Create a list from 0 to 1 in steps of 0.1 *)
Range[0, 1, 0.1]
(* Output: {0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.} *)
```

### Map and Apply

`Map` (shorthand `/@` ) applies a function to each element of a list:

```
(* Square each element in a list *)
Map[#^2 &, Range[5]]
(* Output: {1, 4, 9, 16, 25} *)

(* Shorthand notation *)
```

```
#^2 & /@ Range[5]
(* Output: {1, 4, 9, 16, 25} *)
```

### NestList

`NestList` applies a function repeatedly, creating a list of results:

```
(* Generate a list of powers of 2 *)
NestList[2# &, 1, 5]
(* Output: {1, 2, 4, 8, 16, 32} *)
```

## Generating Special Lists

### Array

`Array` is useful for creating lists with a specific function:

```
(* Create a list of squares *)
Array[#^2 &, 5]
(* Output: {1, 4, 9, 16, 25} *)
```

### ArrayReshape

`ArrayReshape` can be used to create multi-dimensional lists:

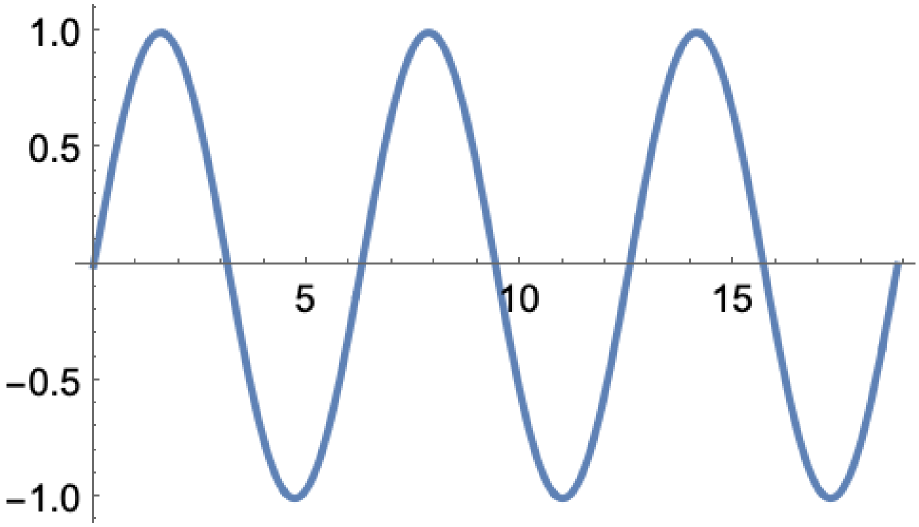
```
ArrayReshape[Range[12], {3, 4}]
(* Output: {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}} *)
```

# Plotting

Mathematica has built-in functions for plotting data and/or expressions (functions)

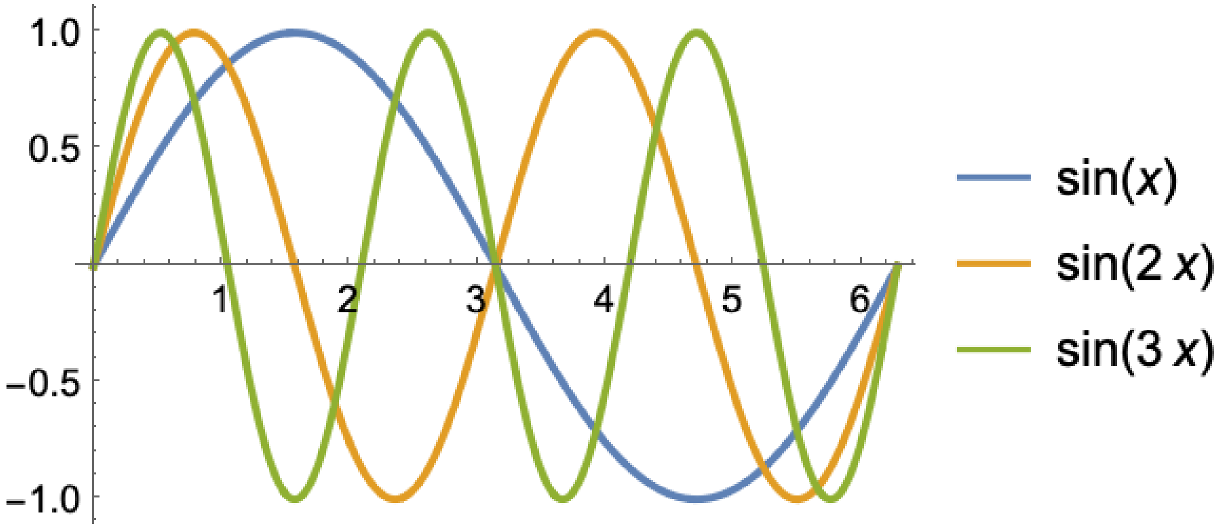
```
Plot[myFunction,{x, x_min, x_max}]
(* Example, plotting sin(x) in the range {0,2Pi} *)

Plot[Sin[x], {x, 0, 6Pi}]
```



Plotting more than one function

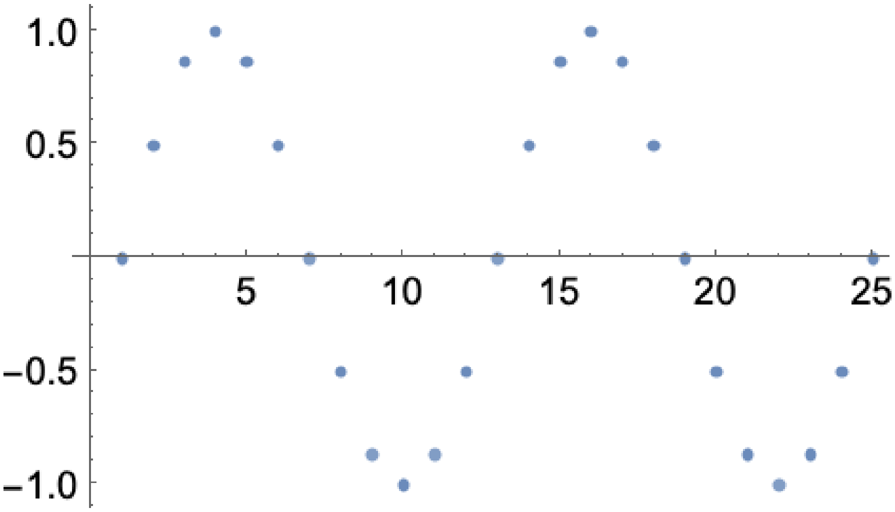
```
Plot[{Sin[x], Sin[2 x], Sin[3 x]}, {x, 0, 2 Pi}, PlotLegends -> "Expressions"]
```



We can also plot data

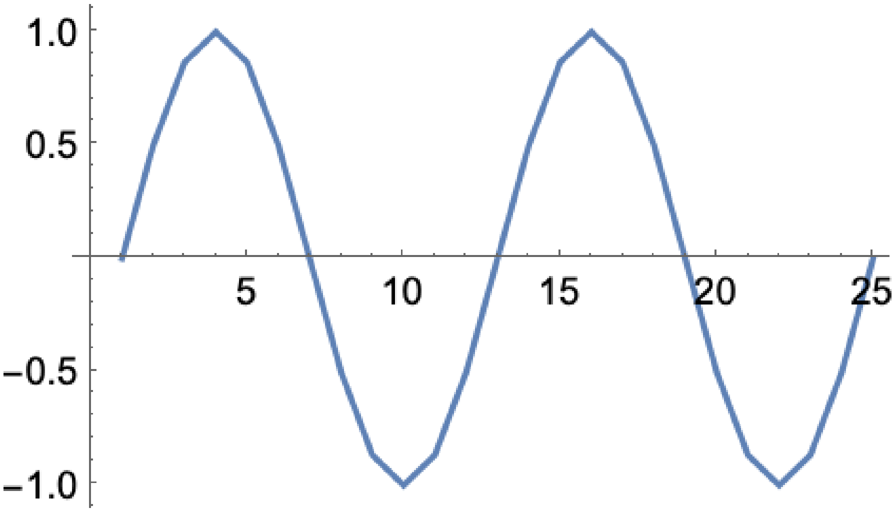
```
(* define the data*)
sdata = Table[1. Sin[2 i], {i, 0, 2 Pi, Pi/12}]

(* Plot sdata *)
ListPlot[sdata]
```



If we want a line connecting the data we can use

```
ListLinePlot[sdata]
```



# Loading, Manipulating, and Saving Data

## Loading Data

Mathematica can import data from various file formats:

```
(* Import CSV file *)
data = Import["path/to/your/file.csv", "CSV"]

(* Import JSON file *)
jsonData = Import["path/to/your/file.json", "JSON"]
```

```
(* Import from a URL *)
webData = Import["https://example.com/data.csv", "CSV"]
```

## Manipulating Data

Mathematica provides powerful functions for data manipulation:

```
(* Select specific columns from CSV data *)
selectedData = data[[All, {1, 3, 5}]]

(* Apply a function to each element *)
squaredData = Map[#^2 &, data]

(* Filter data *)
filteredData = Select[data, #age > 30 &]

(* Group and aggregate data *)
groupedData = GroupBy[data, #category &, Mean]
```

## Saving Data

You can export data in various formats:

```
(* Export to CSV *)
Export["path/to/output.csv", data, "CSV"]

(* Export to JSON *)
Export["path/to/output.json", jsonData, "JSON"]

(* Export a plot *)
plot = Plot[Sin[x], {x, 0, 2 Pi}]
Export["path/to/plot.pdf", plot]
```