Программирование в системе Mathematica.
Типы данных. Функции

Типы данных

- Основные классы данных
- Выражения
- Списки и массивы
- Объекты и идентификаторы
- Функции, опции, атрибуты и директивы
- Подстановки
- Функции линейной алгебры

Основные классы данных

Mathematica оперирует с тремя основными *классами данных:*

- *численными данными,* представляющими числа различного вида;
- *символьными данными,* представляющими символы, тексты и математические выражения (формулы);
- *списками* — данными в виде множества однотипных или разнотипных данных.

Каждый из этих классов данных в свою очередь имеет ряд специальных, более частных типов данных. На них мы остановимся более подробно.

Численные данные

Десятичные числа

К наиболее известным типам данных в математике относятся привычные нам *десятичные числа* (DECIMAL). Каждый разряд таких чисел имеет представление, заданное одной из арабских цифр — 0, 1, 2,..., 9. Весовой коэффициент старшего разряда относительно предшествующего равен 10. Количество цифр, представляющих число, может быть, в принципе, любым. Десятичные числа относятся к следующим основным типам.

Обозначение	**Тип чисел**	**Примеры задания**	
Integer	Целочисленные	123	-345
Rational	Рациональные	123/567	-23/67
Real	Вещественные	123.	-123.45610^6
Complex	Комплексные	-3.5 + 0.	56 I

Десятичные числа наиболее распространены в научно-технических расчетах.

Целые числа

Целочисленные данные (Integer) — это целые числа, например 1, 2 или 123, которые представляются системой без погрешности и ограничения разрядности. Более того, арифметические операции над целыми числами система выполняет также без погрешностей и без ограничения числа цифр.

Количество цифр, представляющих большое целое число, ограничено лишь его значением, но не какими-либо фиксированными форматами. *Рациональные* данные задаются отношением целых чисел, например 123/567, и также представляют результат точно. Поэтому система при символьных и численных расчетах всегда старается выдать результат в виде целых или рациональных чисел, там где это возможно:

```
1000000/3000000
```

```
1/3
```

```
(124-1)/(455+1)
```

```
41/152
```

Фактически целые числа произвольной разрядности в системах символьной математики представляются списками отдельных цифр. Особая организация списков повышает компактность представления больших целых чисел. Характерным примером работы с целыми числами большой разрядности является вычисление факториала n!=1*2*3*. . . *n

Числа с произвольным основанием

Для вычисления чисел с произвольным основанием используется конструкция

```
Основание^^Число
```

Число должно быть записано по правилам записи чисел с соответствующим основанием. Если основание больше 10, для обозначения значений чисел используются буквы от a до z. Наиболее известными из чисел с основанием системы счисления, превышающим 10, являются *шестнадцатеричные* числа (HEX — от слова hexagonal). Разряды таких чисел могут иметь следующие значения:

```
HEX 0123456789abCdef

DECIMAL 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Каждый более старший разряд имеет весовой коэффициент относительно предыдущего разряда, равный 16.

Примеры задания шестнадцатеричного и двоичного чисел:

```
16^^123abcde
```

```
305839326
```

```
2^^1010111
```

```
87
```

Для представления чисел с произвольным основанием *n* (до 32) используется функция BaseForm[expr, n], которая возвращает выражение expr в форме числа с основанием *n,* которое указывается как подстрочный индекс.

Примеры использования функции BaseForm:

```
BaseForm[87,2]
```

1010111_2

```
BaseForm[305839326,16]
```

$123abcde_{16}$

Вещественные числа

Численные данные могут быть представлены также десятичными *вещественными* числами, которые могут иметь различную форму, например 123.456, 1.23456 10^2,12345.6 10^-2 и т. д. В общем случае они содержат *мантиссу с* целой и дробной частями и *порядок,* вводимый как степень числа 10. Как правило, вещественные числа в системах символьной математики могут иметь мантиссу с любым, но конечным числом знаков. Пробел между мантиссой и порядком эквивалентен знаку умножения *:

```
23.456*10^100
```

```
2.345бx10^101
```

```
10^-100
```

```
1/
```

```
10000000000000000000000000000
```

```
00000000000000000000000000000
```

```
000000000000000000000000000000000000000000000000
```

```
10.^-100
```

```
1.x 10^-100
```

Как принято в большинстве языков программирования, целая часть мантиссы отделяется от дробной части *точкой,* а не запятой.

Mathematica производит операции с числами изначально как с целыми. Однако установка значка разделительной точки означает, что число должно рассматриваться как вещественное. Например, 1 — целое число, но 1. — уже вещественное число. Для представления выражения expr в форме вещественного числа используется функция N [expr] или N [expr, число_цифр_результата].

Примеры:

```
1/3
```

```
1/3
```

```
1./3 .
```

```
0.333333
```

```
N[1/3]
```

```
0.333333
```

```
N[2*Pi,50]
```

```
6.2831853071795864769252867665590057683943 38
```

Вещественные числа всегда имеют некоторую погрешность представления результатов из-за неизбежного округления и существования так называемого *машинного нуля* — наименьшего числа, которое воспринимается как нуль. В терминах системы Mathematica говорят о приближении числовых данных как об их *аппроксимации,* хотя в отечественной литературе под аппроксимацией чаще подразумевают описание некоторой зависимости между данными достаточно приближенной аналитической зависимостью.

Mathematica имеет две системные переменные, позволяющие вывести максимально и минимально возможные значения чисел, с которыми оперирует система:

```
$MaxMachineNumber
```

```
1.79769x10^308
```

```
$MinMachineNumber
```

```
2.22507x 10^-308
```

Функции IntegerPart [x] и FractionalPart [x] обеспечивают возврат целой и дробной частей вещественного числа x:

```
N[Pi]
```

```
3.14159
```

IntegerPart[Pi]

```
3
```

FractionalPart[Pi]

```
-3.+ Л
```

N[FractionalPart[Pi]]

```
0.141593
```

Еще одна функция RealDigits [x] возвращает список реальных цифр результата и число цифр целой части x:

RealDigits[N[2*Pi]]

```
{{6, 2, 8, 3, 1, 8, 5, 3, 0, 7, 1, 7, 9, 5, 8, 6}, 1}
```

Есть и множество других функций для работы с вещественными числами. Они будут рассмотрены в дальнейшем. В Mathematica 4 функция RealDigits имеет расширенные формы, например RealDigits [x, b, len, n]. Для получения цифр мантиссы введены функции MantissaExponent [x] и MantissaExpo-nent[x,b].

Комплексные числа

Многие математические операции базируются на понятии *комплексных* чисел. Они задаются в форме

```
z=Re(z)+I*Im(z)
```

или

```
z=Re(z)+i Im (z)
```

где знак I (i) — мнимая единица (квадратный корень из -1), Re (z) — действительная часть комплексного числа, а Im (z) — мнимая часть комплексного числа. Пример задания комплексного числа:

```
2 + I3
```

или

2 + 3*I

Мнимая часть задается умножением ее значения на символ мнимой единицы I. При этом знак умножения * можно указывать явно или заменить его пробелом — в последнем случае комплексное число выглядит более естественным. Функции Re [z] и Im [z] выделяют,

соответственно, действительную и мнимую части комплексного числа z. Это иллюстрируют следующие примеры:

```
Re[3+2*1]
```

```
3
```

```
Im[3+2 I]
```

```
2
```

Большинство операторов и функций системы Mathematica работают с комплексными числами. Разумеется, это расширяет сферу применения системы и позволяет решать с ее помощью различные специальные задачи — например, относящиеся к теории функций комплексного аргумента. Комплексные числа широко используются в практике электро- и радиотехнических расчетов на переменном токе.

Символьные данные и строки

Символьные данные в общем случае могут быть отдельными *символами* (например *a, b,..., z), строками* (strings) и математическими *выражениями* expr (от expression — выражение), представленными в символьном виде.

Символьные строки задаются цепочкой символов в кавычках, например "sssss". В них используются следующие управляющие символы для строчных объектов:

- \n— новая строка (line feed);
- \ t — табуляция.

Это иллюстрируется следующими примерами:

```
"Hello my friend!"
```

```
Hello my friend!
```

```
"Hello\nmy\nfriend!"
```

```
Hello
```

```
my
```

```
friend!
```

```
"Hello\tmy\tfriend!"
```

```
Hello my friend;
```

Следует помнить, что управляющие символы не печатаются принтером и не отображаются дисплеем, а лишь заставляют эти устройства вывода выполнять определенные действия.

Mathematica имеет множество функций для работы со строками, которые будут описаны в дальнейшем.

Выражения

Выражения в системе Mathematica обычно ассоциируются с математическими формулами, как показано в следующей таблице.

Запись на языке Mathematica	**Обычная математическая запись**
2*Sin[x]	2*sin(x)
2 Sin[x]	2*sin(x)
(a +b^2 + c^3) / (3*d - 4*e)	$(a + b^2+c^3)/(3d-4e)$
sqrt(2)	Корень из *2*
Integrate [Sin [x] , x]	Интеграл sin(x) *dx*

Для записи математических выражений используются как операторы, так и функции. Их особенности будут рассмотрены несколько позже. А пока сразу отметим некоторые тонкости синтаксиса системы, используемого при записи арифметических операций:

- знак умножения может быть заменен пробелом;
- встроенные функции начинаются с большой буквы и обычно повторяют свое общепринятое математическое обозначение (за исключением тех, в названии которых есть греческие буквы — они воспроизводятся латинскими буквами по звучанию соответствующих греческих букв);
- круглые скобки () используются для выделения частей выражений и задания последовательности их вычисления;
- параметры функций задаются в квадратных скобках [ ];
- фигурные скобки {} используются при задании списков.

Списки и массивы

Наиболее общим видом сложных данных в системе являются *списки* (lists). Списки представляют собой совокупности однотипных или разнотипных данных, сгруппированных с помощью фигурных скобок:

- { 1 , 2 , 3 } — список из трех целых чисел;
- { a , b , c } — список из трех символьных данных;
- {1, a, x^ 2} — список из разнотипных данных;
- {{a,b},{c,d}} — список, эквивалентный матрице

  a b

  c d

- {x^2+y^2, 2*Sin [x] } — список из двух математических выражений.

Как видно из этих примеров, элементы списков размещаются в фигурных скобках — открывающей { и закрывающей } . Списки могут быть с вложениями из списков — так получаются *многоуровневые* списки (двухуровневый список дает *матрицу).* Позже свойства и возможности списков будут рассмотрены детально. С помощью списков представляются множественные данные — *массивы.*

Объекты и идентификаторы

В общем случае система Mathematica оперирует с объектами. Под ними подразумеваются математические выражения (expr), символы (symbols), строки из символов (strings), упомянутые выше числа различного типа, константы, переменные, графические и звуковые объекты и т. д.

Каждый объект характеризуется своим именем — *идентификатором.* Это имя должно быть уникальным, то есть единственным. Существуют следующие правила задания имен:

- ssss — имя объекта, заданного пользователем;
- Ssss — имя объекта, входящего в ядро системы;
- $Ssss — имя системного объекта.

Итак, все объекты (например функции), включенные в ядро, имеют имена (идентификаторы), начинающиеся с большой буквы (например Plus, Sin или Cos). Идентификаторы относящихся к системе объектов начинаются со знака $. Заданные пользователем объекты следует именовать строчными (малыми) буквами. Разумеется, под символами s...s подразумеваются любые буквы и цифры (но не специальные символы, такие как +, -, * и т. д.).

Объекты (чаще всего это функции), встроенные в систему, принято называть *внутренними* или *встроенными.* Объекты, которые создает пользователь (в том числе используя внутренние объекты), называют *внешними* объектами. К ним, в частности, относятся процедуры и функции, составляемые пользователем, которые детально рассматриваются в дальнейшем.

Функции, опции, атрибуты и директивы

К важному типу объектов принадлежат *функции* — объекты, имеющие имя и список параметров, возвращающие некоторое значение в ответ на обращение к ним по имени с указанием списка конкретных (фактических) значений параметров. В системах Mathematica 2/3/4 встроенные функции задаются в виде

```
Идентификатор_Функции [o1, o2, o3, ...]
```

где ol, o2, o3... — объекты (параметры, опции, математические выражения и т. д.). Список входных параметров задается необычно — в квадратных скобках. В числе входных параметров могут быть специальные объекты — *опции.* Они задаются в виде

```
Имя_опции->Значение_опции
```

Значением опции обычно является то или иное слово. Например, в функции построения графиков

```
Plot [sin[x] , {x, 0,20} ,Axes->None]
```

опция Axes->None указывает на то, что отменяется вывод координатных осей (Axes). Функция Options [name] выводит для функции с идентификатором name список всех возможных для нее опций. Некоторые функции, например Sin, могут вообще не иметь опций, другие, такие как Solve, могут иметь целый «букет» опций:

```
Options [Sin]

Options [Solve]

{InverseFunctions -> Automatic, MakeRules -> False,

Method -> 3, Mode -> Generic, Sort -> True,

VerifySolutions -> Automatic, WorkingPrecision -> 00}
```

В последнем случае характер возвращаемого функцией результата может сильно зависеть от значений опций. Назначение каждой опции мы рассмотрим в дальнейшем. В этой главе они нам пока не понадобятся.

Каждый объект может характеризоваться некоторой совокупностью своих свойств и признаков, называемых *атрибутами.* Функция Attributes [name] возвращает список всех атрибутов функции с именем name, например:

```
Attributes [Sin]

{bistable, NumericFunction, Protected}

Attributes [Solve]

{Protected}
```

Как видите, для функции синуса характерны три атрибута:

- bistable — указывает на применимость в списках и таблицах;
- NumericFunction — указывает на отношение к числовым функциям;
- Protected — указывает на то, что слово Sin защищено от какой-либо модификации.

Кроме того, в Mathematica 2/3/4 имеется понятие *функций-директив.* Эти функции не возвращают значений, а указывают, как в дальнейшем будут выполняться функции, работа которых зависит от директив. Синтаксис функций-директив тот же, что и у обычных функций.

Применение опций и директив делает аппарат функций более гибким и мощным, поскольку позволяет задавать те или иные свойства функций и условия их выполнения. Это особенно важно при использовании функций в задачах графики и символьной математики.

Константы

Константы являются типовыми объектами системы, несущими заранее предопределенное численное или символьное значение. Это значение не должно меняться по ходу вычисления

документа. К численным'константам относятся любые *числа,* непосредственно используемые в математических выражениях или программных объектах, например процедурах и функциях. Так, числа 1 и 2 в выражении 2*Sin [1] являются численными константами. Константы-числа не имеют идентификаторов. Идентификатором, в сущности, является само число. Его представление и хранится в памяти.

Имеется также ряд *именованных* констант, которые можно рассматривать как функции без аргумента, возвращающие заранее заданное значение. Имена констант (и других объектов, например функций и переменных) представляются их *идентификаторами* — непрерывной строкой символов, отождествляемой с именем. В системе Mathematica большинство идентификаторов имеют естественный математический смысл и начинаются с большой буквы. Например, E — это основание натурального логарифма.

Используются следующие встроенные именованные константы:

- Complexlnf inity — комплексная бесконечность, которая представляет величину с бесконечным модулем и неопределенной комплексной фазой.
- Degree — число радиан в одном градусе, которое имеет числовое значение Pi/180.
- E- основание натурального логарифма с приближенным числовым значением 2 . 71828....
- EulerGamma — постоянная Эйлера с числовым значением 0.577216....
- GoldenRatio — константа со значением (l+Sqrt[5]) /2, определяющая деление отрезка по правилу золотого сечения.
- I — представляет мнимую единицу Sqrt [-1].
- Infinity — «положительная» бесконечность (со знаком «минус» дает «отрицательную» бесконечность).
- Catalan — константа Каталана 0 . 915966....
- Pi — число, имеющее значение 3 .14159... и равное отношению длины окружности к ее диаметру.

Константы, имеющие значение, дают его в виде вещественного числа:

```
{N [Degree], N[E], N[Pi]}
```

```
{0.0174533, 2.71828, 3.14159}
```

```
{N[EulerGamma],N[GoldenRatio],N[Catalan]}
```

```
{0.577216, 1.61803, 0.915966}
```

Константы в описываемой системе используются вполне естественно, так что от дальнейшего их описания можно воздержаться.

Переменные

Переменными в математике принято называть именованные объекты, которые могут принимать различные значения, находящиеся в определенном множестве допустимых значений. Подобно этому, переменными в системе Mathematica являются именованные объекты, способные в ходе выполнения документа неоднократно принимать различные значения — как численные, так и символьные. При этом символьные значения переменных, в отличие от обычных языков

программирования, могут представлять собой как исполняемые математические выражения expr, так и некоторые обобщенные классы функций и объектов. Например, переменная может представлять графический объект, такой как изображение трехмерной поверхности, или звуковой объект, при активизации которого исполняется звук. Значением переменных могут быть также множественные объекты — списки.

Имена переменных называют их *идентификаторами.* Они должны быть уникальными, то есть не совпадать с именами директив, атрибутов, опций и функций в ядре системы. Имена переменных должны начинаться с буквы. Общеприняты, скажем, имена x и y для функциональной зависимости y (x) или представления графиков, f — для функций. Желательно назначать именам переменных смысловые значения, например xcoordinaate или ycoordinate для координат точки. Все сказанное об идентификаторах объектов справедливо и для идентификаторов переменных, поскольку переменные — распространенные виды объектов.

Особенности применения переменных

В отличие от переменных в математике, каждая переменная в системе Mathematica, как и в любой системе программирования, всегда отождествляется с некоторой физической областью памяти, в которой и хранится значение переменной. Для уменьшения объема памяти применяются различные способы компактного размещения информации. Надо помнить, что и имя переменной занимает определенную область памяти. Распределение памяти под переменные — динамическое. Это означает, что местоположение ячеек памяти и объем памяти под ту или иную переменную не фиксированы, а меняются в ходе выполнения задачи.

Заранее объявлять тип переменной не требуется. Он определяется операцией присваивания переменной некоторого значения. Такой подход упрощает построение программ и естественен при использовании переменных в обычной математической литературе.

Без особых на то указаний переменные в системе Mathematica являются глобальными. Это означает, что после определения переменной ее значение можно изменить в любом месте документа или программы. Переменная появляется как действующий объект только после ее первого определения или задания. Определения переменных выполняются с помощью операции присваивания, вводимой знаком равенства:

```
var = value
```

Здесь var — имя переменной, value — ее значение. Ниже представлены основные операции по присваиванию переменным значений:

- x = value — переменной x присваивается вычисленное значение value;
- x = y = value — вычисленное значение value присваивается переменным x и y;
- x:=value — присваивание переменной x невычисленного значения value;
- x =. — с переменной x снимается определение.

Примеры (комментарий ln[...] опущен):

- g = Plot[Sin[x],{x,0,20}] — переменной д присваивается значение в виде графического объекта;

- y = 1 + x^2— переменной y присваивается символьное значение в виде математического выражения (1 + x^2);
- z= {1, 2, x, a + b}— переменной z присваивается значение в виде списка, содержащего четыре элемента.

Различие в присваивании переменным значений с помощью знаков «=>> и <<:=» иллюстрируют следующие примеры:

```
a=12;

b=a

12

c:=a

c

12

a=15;

b

12

c

15
```

Как видите, после первоначальных присваиваний b=a и c: =a обе переменные, b и c, имеют значение 12. Однако после присваивания переменной a нового значения (15) переменная b, которой было присвоено *вычисленное* значение a, остается равной 12, а переменная c, которой было присвоено невычисленное значение a, становится равной 15.

Особо обратите внимание на то, что возможно снятие с переменной определения с помощью символов «=.» или функции Clear [var]. В символьной математике это очень полезная возможность, поскольку нередко переменные с одним и тем же именем в разных частях программы могут иметь разный смысл и представлять объекты, требующие значительных затрат памяти.

Более того, эти объекты сохраняются даже при использовании команды New при переходе к подготовке нового документа. Поэтому рекомендуется всякий раз удалять определения переменных, как только их использование завершается. Это предотвращает возникновение конфликтов между одноименными переменными и освобождает память.

Переменные могут быть *локальными,* то есть действующими только в пределах объекта, в котором они объявлены. Таким объектом может быть функция или процедура со списком входных параметров. Такие объекты мы рассмотрим позже.

Оценивание переменных и операции присваивания

Специфику математических выражений в системе Mathematica составляет возможность их оценивания и изменения в соответствии с заложенными в ядро системы правилами математических преобразований. В итоге после изменения значение выражения, которое присваивается переменной, может быть совсем иным, чем до оценивания. Поэтому в целом для определения переменных используют описанные ниже конструкции.

Основная функция Set [Ihs, rhs] имеет аналогичные по действию упрощенные операторы:

- Ihs = rhs — вычисляет правую часть rhs и присваивает ее значение левой части Ihs. С этого момента Ihs замещается на rhs всюду, где бы этот идентификатор ни появился;
- {11, 12, ...} = {rl, г2, ...} — вычисляет ri и назначает полученные результаты соответствующим 11.

Функция задержанного присваивания SetDelayed[lhs,rhs] может быть заменена аналогичным по действию оператором Ihs : =rhs, который назначает правой части rhs роль *отложенного* значения левой части Ihs. При этом rhs содержится в невычисленной форме. После этого, когда появляется идентификатор Ihs, он заменяется на значение rhs, вычисляемое каждый раз заново.

При задержанном (отложенном) присваивании вывода нет, тогда как при обычном немедленном присваивании lhs=rhs значение rhs вычисляется немедленно и результат выводится в строку вывода.

Функция присваивания верхнего уровня UpSet [Ihs, rhs] применяется в виде lhs^{A}=rhs. При этом левой части Ihs присваивается значение правой части rhs, причем это значение связывается с символами, которые появляются на первом уровне вложенности в Ihs.

И, наконец, функцию отложенного присваивания верхнего уровня UpSetDelayed[lhs, rhs] может заменить оператор lbs^ :=rhs. При этом величина rhs выполняет роль отложенного значения Ihs, и связывается это присваивание с символами, которые появляются на первом уровне вложенности в Ihs.

Отметим еще одну важную конструкцию SetOptions [s, namel->valuel, name2->value2, . . .], которая устанавливает для символа s указанные опции, определяемые по умолчанию.

Применение различных типов операций присваивания способствует большей *гибкое*™ системы. Различия между этими операциями на первый взгляд несущественны, но они принципиальны, и это станет понятно после более детального знакомства с символьными преобразованиями и приобретения практики работы с системой.

Имеются также *системные* переменные, значениями которых являются данные о системе и ее работе, например версия применяемой операционной системы, текущая дата, время в данный момент, машинная точность вычислений и т. д. Многие из таких переменных имеют отличительный знак $ перед своим именем. Такие переменные более подробно будут рассматриваться в дальнейшем.

Операторы и функции

Операторы и функции являются основными кирпичиками в построении математических выражений, которые вычисляются или преобразуются системой Mathematica. Кроме того, это важнейшие элементы языка программирования системы. В данном разделе мы познакомимся с этими объектами.

Арифметические операторы

Математические выражения в системе Mathematica записываются с помощью операторов и функций. *Операторы* (от слова operator — исполнитель) являются элементами записи математических выражений, указывающими на то, какие действия производятся над символьными или числовыми данными. Когда эти данные используются совместно с операторами, их называют *операндами.*

Выражения, составленные из операторов, операндов и функций, способны возвращать результат своего вычисления. К примеру, если вычисляется сумма 2+3, то знак «+» является оператором, числа 2 и 3 — операндами, а вся запись 2+3 — выражением. Сами по себе операторы не возвращают какого-либо значения.

Существуют общепринятые приоритеты выполнения операций, например, в первую очередь выполняются сложение и вычитание, затем умножение и деление и далее другие операции. С помощью круглых скобок можно изменять последовательность выполнения действий, например, в выражении (2+3) M вначале будет вычислено 2+3, а затем уже результат будет умножен на число 4. В сомнительных случаях применение скобок особенно желательно, например 2^2+3 даст 7, а 2^ (2 + 3) даст 32.

Ниже перечислены основные операторы для выполнения арифметических операций (x, y и z — операнды, задающие данные, над которыми выполняются вычисления):

x+y+z Сложение

x-y-z Вычитание

x*y*z или x y z Умножение

x/y Деление

x^y Возведение x в степень y

Expr //N Дает приближенное (с установленной точностью и формой) значение выражения expr

Полезно отметить, что знак пробела является арифметическим оператором умножения, если по обе стороны от него стоят операнды.

Как уже отмечалось, при выполнении вычислений особая роль принадлежит символам «%». Эти символы как сами по себе, так и в качестве аргументов функций используются для указания на применение результата предшествующих операций:

- % — возвращает результат последней операции;
- %% — возвращает результат предпоследней операции;

- %...% — возвращает результат операции, выполненной в строке, отстоящей от конца на число повторений символа «%»;
- %n — возвращает результат операции в строке n.

Нетрудно заметить, что применение этих символов облегчает выполнение последовательных вычислений.

Как уже отмечалось, для представления арифметических выражений expr в виде вещественного результата используется функция N[expr,m]. Можно также задать вычисление любого выражения в численном виде, используя выражение expr //N:

```
1/3+2/7
```

```
13/21
```

```
1/3+2/7 //N
```

```
0.619048
```

Таким образом, используя функцию N[expr,m] или вывод с помощью символов //N, можно организовать вычисления в режиме калькулятора, находясь в среде оболочки системы.

Если x имеет вещественное значение, то функция

```
MantissaExponent[x]
```

возвращает список, содержащий мантиссу и порядок приближенного вещественного числа x. Примеры:

```
123.456 10^10
```

1.23456×10^{12}

```
MantissaExponent[%]
```

```
{0.123456, 13}
```

Функции пользователя

Хотя в систему входят многие сотни встроенных функций (начиная от элементарных и кончая специальными математическими функциями и системными функциями), нередко требуется расширить ее вводом новых функций, действие которых задается пользователем. Такие функции принято называть *функциями пользователя*. Функции пользователя — простейшие программные объекты, необходимые даже в том случае, когда пользователь не желает углубляться в тонкости программирования системы. Их цель — расширение системы и ее обучение работе с новыми функциями.

Для задания, опознавания и уничтожения функций пользователя используются следующие конструкции:

- f (x_) := x^3 — отложенное задание функции пользователя с именем f;
- f (x_) = x^3 — немедленное задание функции пользователя с именем f;
- ?*f* — вывод информации о функции f;
- Clear [f] — уничтожение определения функции f.

В обозначениях вида x_ знак _ применяется для создания так называемых *образцов,* задающих локальные переменные в теле функции — в нашем примере это x. При этом в самом теле функции переменные обозначаются как обычно, без знака образца. Он лишь указывает на особый статус переменных в ограниченном пространстве программы — в теле функции. Так, если вместо x_ будет подставлено число 2, то f (2) будет возвращать 2 в степени 3. Вне тела функции значение переменной x не изменяется. Переменная x может быть и неопределенной: x_ определяет переменную x только для тела функции. Более подробно создание образцов будет описано в дальнейшем.

Mathematica позволяет записать введенные пользователем функции с их определениями на магнитный диск с помощью оператора

```
Save["filename", fl, f2, ...]
```

После этого функция пользователя становится *внешней* функцией. При этом для ввода таких функций в текущий документ (notebook) достаточно вызвать файл с именем filename:

```
<<filename
```

Рекомендуется создавать файлы с типовым расширением .m. Такие файлы входят в пакеты расширений системы. Имя файла нужно задавать по общепринятым для MS-DOS правилам, то есть при необходимости указывать логическое имя дисковода и путь к файлу, например, так:

```
<<D: \MAT\myfunc .m
```

Создание внешних функций по существу означает возможность расширения системы и ее адаптации к решению типовых задач конкретного пользователя. Как уже отмечалось, в систему входит мощная библиотека внешних расширений, и каждый пользователь может пополнить ее своими собственными библиотеками расширений.

Функции пользователя могут быть рекурсивными, то есть допускать в своем теле обращение к самим себе. Это связано с тем, что функция становится объявленной сразу же после задания своего имени со списком параметров. Рекурсия — мощный прием программирования, но злоупотреблять им не стоит. Многие рекурсивные алгоритмы более эффективно реализуются без рекурсии, с применением средств процедурного программирования, например циклов.

Логические операторы

Логическими принято называть операции, отражающие чисто логическое соответствие между данными. В обиходном языке эти связи выражаются утверждениями типа «да» или «нет». Например, на вопрос «Сын вырос выше отца?» мы можем ответить «да» или «нет». В математике (да и в информатике) принято характеризовать логическое соответствие утверждениями True («Верно», «Истина» или «Да») и False («Неверно», «Ложь» или «Пет»). Слова True и False

являются символьными константами, отражающими результаты логических операций и в системе Mathematica.

Для осуществления логических операций используются следующие логические *операторы:*

Равенство (например, a == b)

! = Неравенство

> Больше (например, b > a)

>= Больше или равно

< Меньше

<= Меньше или равно

Возможны следующие формы применения операторов сравнения:

```
a == b == c

a != b != c

x < y < z
```

и т. д.

Результатом вычисления этих выражений является выдача логических значений True или False.

Логические функции

Основные логические функции над логическими данными p, q и т. д. задаются следующим образом:

Not[p] или !p Логическое отрицание

And[p, q,...] или p && q &&... Логическое умножение — операция «И»

Or[p,q,...] или p || q | |... Логическое сложение — операция «ИЛИ»

Отметим еще ряд логических операторов и функций:

Equal[Ihs, rhs] Greater[x,y] или x > y Возвращает True, если Ihs и rhs тождественны

Greater[xl,x2,x3] или xl > x2 > x3 Возвращает True, если x оказывается больше y, иначе возвращает False

GreaterEqual[x,y] или x >= y Возвращает True, если xi образуют строго убывающую последовательность, иначе возвращает False

GreaterEqual[xl,x2,x3] или xl>= x2 >= x3 Возвращает True, если x больше или равно y, иначе возвращает False

Negative[x] NonNegative[x] Positive[x] Возвращает True, если xi образуют невозрастающую последовательность, иначе возвращает False

SameQtlhs,rhs] или Ihs === rhs Возвращает True, если x оказывается отрицательным числом, иначе возвращает False

Xor[el, e2,...] Возвращает True, если x — неотрицательное число, иначе возвращает False

Возвращает True, если x — положительное число, иначе возвращает False. Возвращает значение True, если выражение Ihs тождественно rhs, иначе False. В отличие от Equal, сравнивает *форму представления* операндов, а не их значения.

Является логической функцией XOR (исключающее «ИЛИ»). Возвращает True, если нечетное количество из ei имеют значение True, а остальные False. Возвращает False, если четное количество ei имеют значение True, а остальные False

Подстановки

Важное значение в числовых и символьных преобразованиях имеют операции *подстановки или правила* (rules). Их смысл заключается в замене одного объекта или его части другим объектом или частью другого объекта. Например, часто возникает необходимость вычислить значение математического выражения при замене некоторой переменной ее конкретным численным значением. Для этого достаточно вместо этой переменной подставить нужное численное значение.

Куда менее тривиальной является замена переменной ее символьным значением в виде математического выражения. При этом исходное выражение может в ходе решения задачи превратиться в совершенно новое выражение, поскольку после подстановки система может провести над исходным выражением достаточно сложные математические преобразования. Говорят, что в этом случае ячейка, содержащая выражение (а точнее — само выражение), *оценивается* и *изменяется* по ходу решения задачи. Операции подстановки обычно вводятся с помощью комбинации символов «/ .»:

- expr / . x -> value — в выражение expr вместо переменной x подставляется ее значение value;
- expr /. {x -> xvalue,. y -> yvalue} — в выражение expr вместо переменных x и y подставляются их значения xvalue и yvalue.

Примеры:

```
1+x^3/.x->1+z
```

$1 + (1+ z)^3$

```
x^2+2*x+3/.x->2
```

```
11
```

Обратите внимание на то, что в результате подстановки в первом примере вместо переменной x оказалось математическое выражение (1 + z). Второй пример иллюстрирует подстановку на место переменной x ее численного значения.

В целом для операций подстановок используют следующие обозначения:

- lhs -> rhs — прямая подстановка lhs в rhs;
- lhs :> rhs — отложенная подстановка (RuleDelayed), которая преобразует lhs в rhs, вычисляя rhs не сразу, а только при *использовании* правила подстановки.

Ниже приведены еще два примера на использование операций подстановки:

```
p:=1+x^2+3*x^3
```

```
p/.x->1+y
```

```
1+ (1 + y)^2+3 (1 + y)^3
```

```
{f[1],f[2],f[3]}/.f[n_]->n^2
```

```
{1, 4, 9}
```

```
f[n_]:=n^2
```

```
f[4]+f[y]+f[x+y]
```

```
16+y^2+(x+y)^2
```

В первом примере подстановка произведена в математическое выражение, а во втором — в список.

Подстановки — мощный и необычайно гибкий инструмент системы Mothe-matica. С их помощью можно задать даже новые математические закономерности и произвольные соотношения (к примеру, можно задать абсурдное правило, что 2 + 2 = 5). Эти необычные возможности мы рассмотрим в дальнейшем.

Работа со списками и массивами

Списки относятся к данным множественного типа. Они имеют большое значение при обработке массивов данных и служат основой для создания векторов и матриц. В этом разделе мы познакомимся со свойствами списков, их созданием (генерацией) и использованием.

Списки и их свойства

Часто математические или иные объекты содержат множество данных, которые желательно объединять под общим именем. Например, под объектом с именем *M* можно подразумевать квадратную матрицу размером 10x10 с общим числом элементов, равным 100. Человека с именем *Victor,* например, можно характеризовать целым списком разных данных — символьными фамилией, именем и отчеством, целочисленным годом рождения, вещественным ростом, объемом груди и т. д.

Для объединения данных могут использоваться *списки* (list). Mathematica имеет обширные возможности работы с объектами-списками, содержащими не только однотипные, но и разнотипные элементы. В частности, элементами списков могут быть числа, константы, переменные, выражения и даже сами списки. Списки используются для конструирования более частных типов данных — массивов, матриц и векторов.

На языке данной системы список — это совокупность произвольных данных, указанных в фигурных скобках, например: {1, 4, 2, 7, 9} или {a, b, c, d, e, sin[x], ln[y], "string"}

Возможно задание списков в списке, например, так:

```
{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
```

Такой список представляет матрицу

1 2 3

4 5 6

7 8 9

Однако, чтобы вывести список в такой матричной форме, надо использовать после списка выражение //MatrixForm (рисунок 3.1).

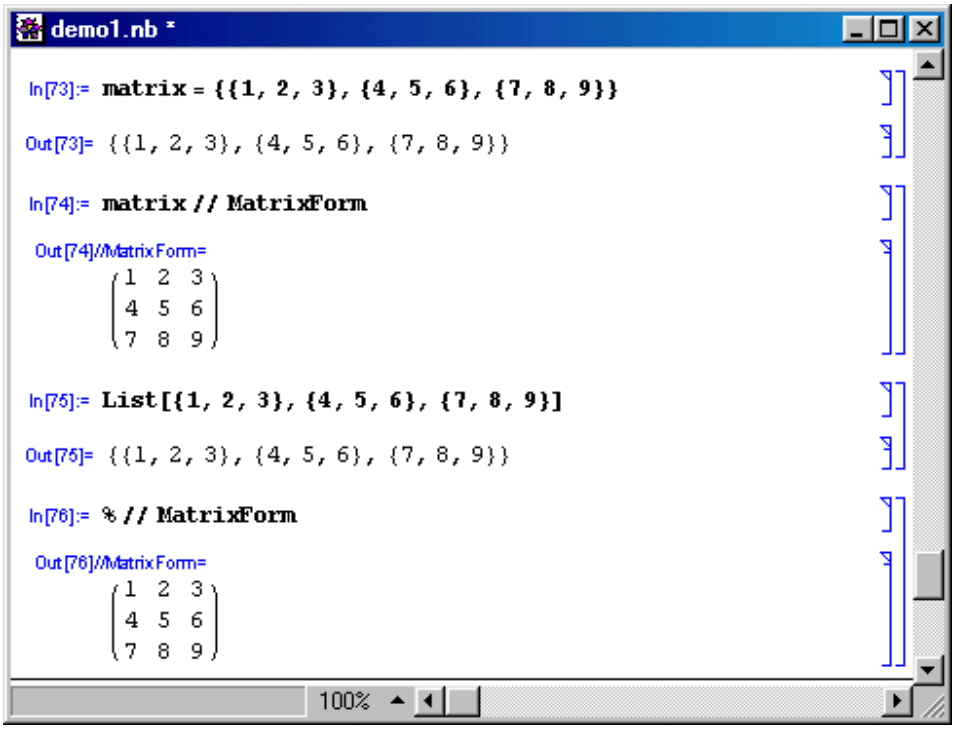

Рисунок 3.1 - *Примеры задания и вывода матрицы*

На рисунке 3.1 показан еще один способ задания списка или матрицы — с помощью функции List:

- List [a, b, c,...] — создает список {a, b,, c,...};
- List [{a,b, c,...}, {d,e, f,...}, {i, k, 1,...}] — создает список — матрицу { {a,b, c,...}, {d,e, f,...}, {i, k, 1,...} }.

Списки можно составлять, непосредственно задавая объекты в соответствии с описанным синтаксисом. Однако можно и генерировать некоторые виды списков, таких как таблицы. Списки могут быть объектами присваивания переменным, например

```
V:={1, 2, 3, 4, 5}
```

Списки характеризуются *размером,* который представляет собой произведение числа элементов списков по каждому направлению *(размерности).* Например, одномерный список является *вектором* и характеризуется числом элементов по единственному направлению. При этом вектор может быть вектором-строкой или вектором-столбцом. Двумерный список представляет *матрицу,* имеющую m строк и п столбцов. Ее размер равен mxn. Если m=n, то матрица называется квадратной. Трехмерный список можно представить в виде параллелепипеда, содержащего mxnxp элементов. Списки большей размерности трудно наглядно представить, но они вполне возможны. Напомним, что имена векторов и матриц в данной книге обозначены жирными символами, например, V для вектора и M для матрицы.

Генерация списков

Для генерации списков с элементами, являющимися вещественными и целыми числами или даже целыми выражениями, часто используется функция Table, создающая таблицу-список:

- Table [expr, {imax}] — генерирует список, содержащий imax экземпляров выражения expr;
- Table [expr, {i, imax}] — генерирует список значений expr при 1, изменяющемся от 1 до imax;
- Table [expr, {i, imin, imax}] — генерирует список значений expr при i, изменяющемся от imin до imax;
- Table [expr, {i, imin, imax, di}] — использует шаг приращения i, равный di;
- Table[expr, {i, imin, imax}, {j, jmin, jmax},...] —возвращает вложенный список. Самым внешним является список по переменной i.

Применяется также функция Range, которая предназначена для создания так называемых числовых списков, значения которых равномерно распределены в некотором заданном диапазоне:

- Range [imax] — генерирует список числовых элементов {1, 2, ..., imax};
- Range [imin, imax] — генерирует список числовых элементов {imin, ...,
- Range [imin, imax, di] — генерирует список числовых элементов от imin до imax с шагом di.

Выделение элементов списков

Для выделения элементов списка list используются двойные квадратные скобки:

- list [[i]] — выделяет i-й элемент списка;
- list [[{ i, j , --.}]] — выделяет i-й, j-й и т. д. элементы списка.

Для выделения заданного i-го элемента списка list используется также функция Part [list, i]. При i>0 отсчет номеров элементов идет с начала списка, а при i<0 — с его конца. Это правило поясняют следующие примеры:

```
L:={1,2,3,a,b,c}

{Part[L,2],Part[L,5],Part[L,6]}

{2, b, c}

{Part[L,-2],Part[L,-5],Part[L,2]}

{b, 2, 2}
```

Функция Part может использоваться для выбора заданного элемента выражения из списка. В этом случае вместо i надо указать три числа — номер выражения как элемента списка, уровень выражения и порядковый номер извлекаемого из выражения объекта.

Функция Select используется для выделения элементов списка, удовлетворяющих заданному критерию:

- Select [list, crit] — выбирает все элементы ei списка list, для которых функция критерия crit [ei] имеет значение True;

- Select [list, crit, n] — выбирает первые n элементов, для которых critfei] есть True.

Вывод элементов списков

Для вывода элементов списка используются следующие функции:

- MatrixFormflist] — выводит список в форме массива (матрицы);
- TableForm [list] — выполняет вывод элементов списка list в виде таблицы.

С этими функциями используются следующие опции:

- TableAlignments — указывает, каким образом должно выравниваться содержимое списка в каждой размерности (слева, по центру или справа);
- TableDepth — устанавливает максимальное количество уровней, выводимых в табличном или матричном формате;
- TableDirections — указывает, как следует располагать последовательные (соседние) размерности — в виде строк или столбцов;
- TableHeadings — задает подписи (labels) для каждой размерности таблицы или матрицы;
- TableSpacing — устанавливает количество пробелов, которое следует оставлять между соседними строками или столбцами.

Обратите внимание на то, что эти опции используются как для функции TableForm, так и для функции MatrixForm, используемой для вывода матриц. Вообще, векторы и матрицы являются разновидностью списков.

В большинстве случаев опции для функций MatrixForm и TableForm не используются. Точнее, они установлены по умолчанию.

Функции выявления структуры списков

Списки относятся к данным сложной структуры. Поэтому при работе с ними возникает необходимость контроля за структурой, иначе применение списков может привести к грубым ошибкам, как явным, сопровождаемым выдачей сообщения об ошибке, так и неявным. Последние могут привести к серьезным просчетам.

Для выявления структуры списков используется ряд функций:

- Count [list, pattern] — возвращает количество элементов в списке list, которые соответствуют образцу pattern;
- Dimensions [list] — возвращает список размеров списка по каждой размерности;
- FreeQ [list, form] — возвращает True, если список list не содержит form;
- Length [list] — возвращает число элементов одномерного списка list или число размерностей в случае многомерного списка;
- MatrixQ [list] — проверяет, является ли список матрицей, и дает True, если это так, и False в противном случае;
- MemberQ [list, form] — проверяет, есть ли form в списке, и возвращает True, если это так, и False в противном случае;
- Position [list, form] — возвращает номер позиции form в списке;
- TensorRank[list] — находит ранг списка, если он является тензором;

- VectorQ [list] — проверяет, является ли список вектором, и дает True, если это так, и False в противном случае.

Функции с буквой Q в конце имени являются тестирующими и возвращают логические значения True или False. Остальные функции возвращают численные значения соответствующего параметра списка.

Изменение порядка расположения элементов в списке

Помимо добавления в список новых данных имеется возможность изменения порядка расположения элементов в списке. Она реализуется следующими операциями:

- Flatten [list] — выравнивает (превращает в одномерный) список по всем его уровням;
- Flatten [list, n] — выравнивает список по n его уровням;
- Flatten [list, n, h] — выравнивает выражения с заголовком h no n уровням;
- FlattenAt [list, n] — выравнивает подсписок, если он оказывается n-м элементом списка list. Если n отрицательно, позиция отсчитывается с конца;
- Sort [list] — сортирует элементы списка list в каноническом порядке;
- Sort[list,p] — сортирует согласно функции упорядочения p;
- Reverse [list] — возвращает список с обратным порядком расположения элементов;
- RotateLeft [list] — возвращает список после однократного поворота влево;
- RotateLeft [list, n] — возвращает список после n-кратного поворота влево;
- RotateRight [list] — возвращает список после однократного поворота вправо;
- RotateRight [list, n] — возвращает список после n-кратного поворота вправо;
- Transpose [list] — осуществляет транспозицию (смену строк и столбцов) для двумерного списка;
- Transpose [list, n] — осуществляет транспозицию n-мерного списка. Ниже приведен ряд примеров на использование этих функций.

Ввод (In)	**Вывод (Out)**
13={{1,2,3},{4,5,6},{7,8,9}};	{1,2,3,4,5,6,7,8,9}
Flatten [13]	
FlattenAt[13,l]	{1,2,3,{4,5,6},{7,8,9}}
Sort[{l,5,3,4,2}]	{1,2,3,4,5}
Reverse[{l,2,3,4}]	{4,3,2,1}
RotateLeft[{1,2,3,4,5}, 2]	{3,4,5,1,2}
RotateRight[{l,2,3,4,5} ,2]	{4,5,1,2,3}
12={{a,b},{c,d}};	
TableForm[12]	a b c d
TableFormf Transpose [12]]	a c d b

Изменение порядка расположения элементов в списке полезно при реализации некоторых алгоритмов. К примеру, сортировка списка ускоряет выполнение статистических расчетов и уменьшает их погрешности.

Комбинирование списков и работа с множествами

Иногда возникает необходимость комбинирования нескольких списков. Для этого используются следующие функции:

- Complement [list, listl, list2, ...] — возвращает список list с элементами, которые не содержатся ни в одном из списков listl, Iist2, ...;
- Intersection [listl, list2,...] (пересечение множеств) —возвращает упорядоченный список элементов, общих для всех списков listi;
- Join[list1, list2,...] — объединяет списки в единую цепочку (выполняет конкатенацию). Join может применяться к любому множеству выражений, имеющих один заголовок;
- Union [listl, Iist2, ...] (объединение множеств) — удаляет повторяющиеся элементы списков и возвращает отсортированный список всех различающихся между собой элементов, принадлежащих любому из данных списков listi. Функция обеспечивает теоретико-множественное объединение списков;
- Union [list] — возвращает отсортированный вариант списка list, из которого удалены все повторяющиеся элементы.

Комбинирование списков позволяет создавать сложные структуры данных из более простых структур. Это может быть полезно при построении очередей, деревьев и иных структурных построений. Кроме того, приведенные функции обеспечивают основные операции со множествами. Функцию Union удобно использовать при решении нелинейных и алгебраических уравнений для удаления повторяющихся решений.

Создание массивов

Совокупность данных образует *массив* (Array). Массивы могут быть одномерными (один список), двумерными и многомерными (два и более списка). Одномерные массивы в математике называют *векторами,* двумерные — *матрицами.* В общем случае массив характеризуется *размерностью* (числом измерений) и *размером* — произведением числа элементов по всем размерностям. Mathematica позволяет создавать многомерные массивы — число элементов в них ограничено лишь объемом памяти компьютера.

Для задания массивов используются следующие функции:

- Array [f, n] — генерирует список длиной п с элементами f [1], f [2], ..., f[n];
- Array [f, {nl, n2, ...}] — генерирует массив размером n1x n2 x... в виде вложенных списков с элементами f [i I, i2,...] (аргумент функции i k меняется от 1 до nk);
- Array[f, dims, origin] — генерирует список с размерностью dims, используя спецификацию индекса origin;
- Array [f, dims, origin, h] — использует заголовок h, а не List, для каждого уровня массива.