Справочник по математическим объектам и их представлению в Scilab.

В среду Scilab встроены типы данных, которые в большинстве случаев являются основой для проведения математических расчетов. Мы будем называть их объектами. Внутренняя структура объектов заранее предопределена внутри среды и скрыта от глаз пользователей. Работу с памятью при обработке объектов среда также берет на себя. В дальнейшем эти предопределенные объекты могут послужить для создания пользовательских объектов.

Переменная в Scilab— это именованный массив всего с одним полем, которое хранит данные некоторого типа.

Среди типов данных можно выделить:

1. Числа
2. Целые числа
3. Вещественные числа
4. Комплексные числа
5. Строки
6. Логические переменные

Чтобы создать переменную, нужно ввести её имя (ВАЖНО! чтобы оно соответствовало правилам, приведённым ниже) и присвоить ей какое-либо значение.

Правила, которым должны удовлетворять имена переменные:

1. Имя переменной может состоять из букв латинского алфавита (верхнего и нижнего регистра) и цифр;
2. Имя переменной не может начинаться с цифры, но может начинаться с символов '%', '_', '#', '!', '$', '?';
3. Регистр в имени играет роль, то есть переменные с именами var,VAR ,Var и т. п. разные;
4. Запрещено совпадение имени переменной с зарезервированными словами, такими как имена объявленных функций, констант и др.;

Для работы с любыми объектами в среде существуют предопределенные **функции**, которые всегда загружаются вместе со средой. Приведённые в таблице функции используются повсеместно.

Имя функции	Ключевые слова для параметров	Аргументы	Назначение
whos	**Необязательные**: **-type** typ **-name** nam	typ — текстовая строка, кодирующая тип данных nam — имена искомых переменных, либо их начальные фрагменты	Выводит список переменных в длинной форме (с указанием типа и размера), если вызывается без аргументов. Выводит список переменных, хранящих определенный тип данных (если указан параметр -type) или/и обладающих определенным именем или фрагментом этого имени (если указан параметр -name).
who()	аргументы перечисляются в скобках	**Необязательные**: 'local' или 'get' — список локальных переменных и констант в коротком виде 'global' — список глобальных переменных в коротком виде 'sorted' — возвращает сортированный список объявленных переменных (локальных и глобальных) с текущими значениями параметров памяти.	Без аргументов возвращает несортированный списо к объявленных переменных (локальных и глобальных) с текущими значениями параметров памяти.
who_user	нет	нет	Работает почти как who('local'), но выводит только переменные, созданные пользователем во время текущей сессии.
typeof()	нет	**Обязательный**: object — имя объекта среды	Возвращает в виде строки тип объекта, чье имя передается в качестве аргумента. В отличие от функции type(), более гибко работает с типами данных. О возвращаемых значениях смотрите в справке по данной функции.

clear	нет	**Необязательные**: Аргументами являются имена объектов, перечисляемых через пробел.	Удаляет из памяти объекты и освобождает имена. Если вызывается без аргуметов, то удаляются все незащищенные объекты текущей сессии.
predef()	нет	'a' или 'all' — защищает все переменные из списка who('get') 'c' или 'clear' — снимает защиту со всех переменных списка who('get'), кроме последних 7 (по умолчанию), некоторых предопределенны х системных переменных и констант n (n>7) — устанавливает n последних переменных в качестве защищенных 'names' — возвращает список защищенных переменных	Функция предназначена для защиты переменных. Вызов без аргументов возвращает количество защищенных переменных. Остальные вариации перечислены в колонке с аргументами.
exists()	нет	**Обязательный**: name — имя переменной (передается строкой, то есть в кавычках) **Необязательный**: where — возможны варианты: 'l' — локальный, 'n' — нелокальный, 'a' — все	Функция проверяет существование объекта с указанным именем. Если объект существует, то функция возвращает 1, в противном случае 0. С помощью флага, можно задавать область поиска. По умолчанию используется вариант 'a'.

Целые числа

В Scilab целые числа возможно создавать только через специальные функции. Во всех остальных случаях числовому значению всегда будет присваиваться вещественный тип данных.

Для хранения целого числа в памяти может быть использовано разное число битов, а именно 8, 16 и 32. От количества используемых битов зависит диапазон целых чисел. Кроме того, имеется возможность включения и отключения знакового бита, что бывает полезно, когда отрицательные целые числа не требуются.

В таблице ниже приведены функции для создания целых чисел и диапазоны возможных значений. Во всех случаях функция возвращает целое число, указанное в качестве аргумента, с определенным способом его хранения в памяти.

Имя функции	Тип целого числа	Диапазон	Код представления
int8()	знаковое 8-битное число		1
uint8()	беззнаковое 8-битное число		11
int16()	знаковое 16-битное число		2
uint16()	беззнаковое 16-битное число		12
int32()	знаковое 32-битное число		4
uint32()	беззнаковое 32-битное число		14

Какой тип целочисленной переменной следует применить, пользователь решает сам. Целые числа могут быть использованы как в чисто математических задачах, так и программировании, в качестве счетчиков.

Для работы с целочисленными переменными существует две функции:

1. **iconvert(X, itype)** — меняет представление в памяти в общем случае матрицы из вещественных или целых чисел. Функции при этом следует передавать имя этой матрицы X и код внутреннего представления целого числа itype из последней колонки таблицы выше;
2. **inttype(X)** — возвращает код внутреннего представления в общем случае матрицы целочисленных данных.

Строки

Строковый тип данных образуется заключением символов в одинарные или двойные кавычки.

```
--> a='Scilab', b="is smart"
```

Матрицы

Матрица в Scilab — это двухмерный массив однотипных элементов. Можно понимать матрицу как несколько векторов-строк, записанных столбцом.

Создать матрицу в Scilab можно одним из нескольких способов:

- Матрицу можно создать из составляющих ее элементов;
- Из имеющихся векторов, упорядочив их строками или столбцами;
- Одной из специальных функций.

В общем случае синтаксическая конструкция имеет вид

```
[x11, x12, …, x1n; x21, x22, …, x2n; …; xm1, xm2, …, xmn]
```

Таким образом, вы создаете векторы-строки, которые отделяете точкой с запятой. Запятая в этом случае, как и с вектором, не обязательна.

```
-->A=[1 2; 3 4] // создадим матрицу из составляющих ее элементов
 A  =
   1.   2.
   3.   4.
```

Для работы с матрицами существуют следующие основные функции

Имя функции	Аргументы	Назначение
size(V,[flag])	V — массив; flag — возможны следующие варианты: 'r' или 1 —	Функция возвращает размеры массива в виде вектора, если не указаны флаги.

	возвращает число строк; 'c' или 2 — возвращает число столбцов	
length(V)	V — массив	Функция возвращает число элементов в массиве
max(V,[flag]) **max(V1,V2,…,Vn)**	V — массив	Функция возвращает элемент с максимальным значением в указанном массиве или группе массивов (см. второй вариант вызова).
min(V,[flag])	Аналогично функции **max()**	Функция возвращает элемент с минимальным значением в указанном массиве или группе массивов.

Математическая функция

В Scilab есть предопределенные математические функции, такие как тригонометрические, экспонента и другие. Но что, если вам необходимо определить собственную функцию? Далее мы будем отличать математические функции и программируемые функции. Разница между ними заключается в том, что программируемые функции призваны реализовывать некоторый алгоритм, в то время как математическая функция отражает связь между множеством аргументов и множеством значений функции.

Отметим, что математическую функцию можно объявить через программирование, но так как обычно ее тело состоит из одной строчки, то рациональнее всего объявлять ее через специальную функцию **deff()**. Общий синтаксис имеет вид

```
deff('[Y1,Y2...]=Fname(X1,X2,...)',['Y1=выражение_1';'Y2=выражение_2;...'])
// [Y1,Y2...] — вектор возвращаемых переменных (имена назначаете сами)
// Fname — имя функции, которое вы назначаете сами
// (X1,X2,...) — список из аргументов функции
// 'Y1=выражение_1;Y2=выражение_2;...' — для каждой выходной переменной должно быть определено выражение,
// которое может зависеть от аргументов, а может и не зависеть.
```

Данная конструкция поначалу кажется сложной и неизбежны ошибки при ее вводе. Главное не забывать, что аргументы для функции deff() вы передаете в виде строк, которые затем разбиваются на лексемы и из которых среда строит программируемую функцию.