1. Последовательности (Sequences)

- **Создание:** Оператор $
  - x$n=1..10 → последовательность чисел 1, 2, 3, ..., 10
  - x^2$n=1..10 → последовательность квадратов 1, 4, 9, ..., 100
  - x$x="A".."Z" → последовательность букв от A до Z
- **Результат:** Несколько элементов, разделенных запятыми.

2. Списки (Lists)

- **Создание:** Квадратные скобки [ ]
  - [1/n$n=1..10] → список [1, 1/2, 1/3, ..., 1/10]
- **Тип:** list (проверка: whattype(L) и type(L, list))
- **Особенности:** Сохраняют порядок и позволяют дубликаты.

3. Команда seq

- **Синтаксис:** seq(expr, i = m..n)
  - seq(i, i=1..5) → 1, 2, 3, 4, 5
  - [seq(i, i=0..6)] → список [0, 1, 2, 3, 4, 5, 6]
  - [seq(i^2, i=X)] → список квадратов элементов списка X
  - [seq([X[i], Y[i]], i=1..nops(X))] → список пар (точки для графика)
- **Шаг:** [seq(i, i=1..10, 2)] → [1, 3, 5, 7, 9] (с шагом 2)
- **Обратный порядок:** [seq(i, i=10..1, -2)] → [10, 8, 6, 4, 2]
- **Для строк:** seq(i, i="Hello") → "H", "e", "l", "l", "o"

4. Множества (Sets)

- **Создание:** Фигурные скобки { }
  - {n$n=1/3..10/3} → {1/3, 4/3, 7/3, 10/3}
- **Тип:** set (проверка: whattype(S) и type(S, set))
- **Особенности: Не** сохраняют порядок, только уникальные элементы.

5. Команды для работы со списками

- **Длина списка:** nops(L)
- **Выбор элементов:** op(2..5, L) → элементы со 2-го по 5-й
- **Добавление элемента:** L := [op(L), новый_элемент]
- **Удаление элементов:** remove(условие, L)
  - remove(x -> x > 5, L) → удаляет все элементы > 5

- remove(x -> x = 1, L) → удаляет все единицы
- remove(x -> irem(x,2)=0, L) → удаляет чётные (остаток от деления на 2 равен 0)
- remove(x -> iquo(x,5)=1, L) → удаляет элементы, целая часть от деления на 5 равна 1

- **Выбор элементов:** select(условие, L)
- select(x -> irem(x,2)=0, L) → выбирает чётные
- select(x -> x < 8, L) → выбирает элементы < 8
- select(type, L, string) → выбирает все строки
- select(type, L, numeric) → выбирает все числа
- **Объединение списков:** LL := [op(L1), op(L2)]
- **Команда zip:** Попарное применение функции к элементам двух списков
- zip((x,y) -> (x,y), L1, L2) → создает список пар [1,4, 2,5] (если L1=[1,2], L2=[4,5])
- zip((x,y) -> [x,y], L1, L2) → создает список списков [[1,4], [2,5]]
- zip((x,y) -> (x^2, y/10), L1, L2) → применяет функции к элементам
- zip((x,y) -> x*y, L1, L2) → вычисляет попарные произведения
- **Сортировка:** sort(L) (по возрастанию), sort(L, (x,y)->is(x>y)) (по убыванию)

6. Команды для работы с множествами

- **Создание:** s := {1,1,2,1,3} → {1,2,3} (дубликаты автоматически удаляются)
- **Разнородные элементы:** Множества могут содержать числа, строки, символы, константы, даже списки.
- **Преобразование в список:** L := [op(s)]
- **Объединение множеств:** s3 := union(s1, s2) или s3 := {op(s1), op(s2)}
- **Пересечение множеств:** s3 := intersect(s1, s2)
- **Разность множеств:** s3 := minus(s1, s2)

1. Решение алгебраических уравнений

- **Основная команда:** solve(уравнение, переменная)
  - Возвращает точные (символьные) решения.
  - Пример: solve(x^3 + 2*x^2 - x + 12 = 0, x) (1.1-1.3)
- **Численное приближение:** evalf(решение) для получения численных значений корней.
- **Проверка решения:**
  - Подстановка: subs(x = корень, уравнение)
  - Сравнение с допуском: is(значение < epsilon) (1.4-1.5)
- **Примечание:** Результат solve для уравнений высоких степеней может быть громоздким.

2. Решение трансцендентных уравнений

- **Особенность:** Уравнения, содержащие неалгебраические функции (тригонометрические, показательные, логарифмические и т.д.).
- **Метод:**

1. **Визуализация:** Постройте график (plot), чтобы определить количество корней и их приближенное положение.
2. **Численное решение:** Используйте fsolve(уравнение, переменная = интервал_поиска).

- Требует начального приближения или интервала.
- Пример: fsolve(f, x=0), fsolve(f, x=1.5), fsolve(f, x=1.8)

- **Проверка:** Всегда проверяйте найденные корни подстановкой.
- **Примечание:** Команда solve часто не может найти точное решение для трансцендентных уравнений и возвращает неопределенный объект RootOf.

3. Решение тригонометрических уравнений

- **Переменная _EnvAllSolutions:** Ключевая настройка.
  - _EnvAllSolutions := false (по умолчанию): возвращает только главные решения.
  - _EnvAllSolutions := true: возвращает **все решения** в общем виде с использованием параметра (например, _Z~).
- **Пример:** solve(sin(x)=0, x) при _EnvAllSolutions:=true вернет π * _Z~.
- **О параметре:** Команда about(_Z~) покажет, что такое _Z~ .

4. Решение уравнений с кусочными функциями

- **Задание функции:** f := piecewise(условие1, выражение1, условие2, выражение2, ..., выражение_по_умолчанию)
- **Решение:** solve(f, x) пытается найти корни для всех "кусков" функции.
- **Визуализация:** График (plot) помогает понять поведение функции и найти корни.

5. Поиск целых решений

- **Целые решения:** Можно получить, вызвав isolve.

6. Решение систем уравнений

- **Синтаксис:** solve({уравнение1, уравнение2, ...}, {переменная1, переменная2, ...})
- **Результат:** Множество решений.
- **Визуализация:** plots:-implicitplot полезен для графического отображения решений.

7. Решение неравенств

- **Синтаксис:** solve(неравенство, переменная)
- **Результат:** Интервал или объединение интервалов.
- **Визуализация:** Постройте график функции, чтобы увидеть, где она положительна/отрицательна.

8. Пакет **RootFinding Особенно полезен для сложных уравнений и систем**

- **Назначение:** Расширенные возможности для поиска корней.
- **Команды:**
  - HasRealRoots(уравнение): Проверяет наличие действительных корней.
  - NextZero(функция, точка): Находит следующий корень функции, начиная с заданной точки. Полезно для поиска нескольких корней.
  - Isolate: Изолирует корни полинома.

9. Решение систем неравенств

- **Синтаксис:** solve({неравенство1, неравенство2, ...}, переменная)
- **Результат:** Область решений, часто описываемая системой неравенств.
- **Численная оценка:** evalf можно применить к результату, чтобы получить численные границы.
- **Визуализация:** Постройте графики всех функций, входящих в систему, чтобы увидеть область пересечения.

Итог:

- **solve** - основной инструмент для **точного** решения.
- **fsolve** - основной инструмент для **численного** решения.
- **plot** - **критически важен** для визуализации и понимания задачи перед решением.
- **_EnvAllSolutions** управляет полнотой решений тригонометрических уравнений.
- **Пакет RootFinding** предоставляет специализированные и мощные инструменты для сложных случаев.

1. Базовые операции со строками

- **Тип данных:** string (проверка: whattype(s))
- **Доступ к символам:** s[1] - первый символ, s[length(s)] - последний символ
- **Длина строки:** length(s)
- **Спецсимволы:** \n - новая строка, \t - табуляция
- **Преобразование типов:**
  - convert(s, symbol) - строка → символ
  - convert(sym, string) - символ → строка

2. Пакет StringTools

- **Подключение:** with(StringTools);
- **Регистр символов:**
  - UpperCase(s) - ВЕРХНИЙ РЕГИСТР
  - LowerCase(s) - нижний регистр
  - Capitalize(s) - Первая Буква Заглавная
  - CamelCase(s) - ВерблюжийРегистр
  - OtherCase(s) - иНвЕрТнЫй РеГиСтР

3. Проверка свойств символов (Character Class Tests)

- **Наличие категорий:** HasAlpha(s), HasDigit(s), HasPunctuation(s)
- **Проверка принадлежности:** IsAlpha(c), IsDigit(c), IsHexDigit(c)
- **Специальные проверки:**
  - HasVowel(s) - содержит ли гласные
  - IsPrintable(s) - все ли символы печатные
  - IsGraphic(s) - содержит ли графические символы
  - HasControlCharacter(s) - содержит ли управляющие символы

4. Комбинаторика слов (Combinatorics on Words)

- **Палиндромы:** IsPalindrome(s) - проверка симметричности
- **Периодичность:** IsPeriod(s, n) - проверка периодичности
- **Перестановки:**
  - IsPermutation(s) - все ли символы уникальны
  - IsDerangement(s1, s2) - является ли одна строка перестановкой другой
- **Примитивность:** IsPrimitive(s) - нельзя представить как степень другой строки
- **Границы:** Border(s), BorderLength(s) - поиск границ строки
- **Генерация:** Fibonacci(n, s1, s2) - генерация строки Фибоначчи

5. Сравнение строк (Comparisons)

- **Префиксы и суффиксы:** IsPrefix(s1, s2), IsSuffix(s1, s2)
- **Сравнение:** Compare(s1, s2), CompareCI(s1, s2) (без учета регистра)

6. Генерация строк

- Repeat(s, n) - повторение строки
- Random(n) - случайная строка
- Iota(n) - последовательность символов
- Fill(c, n) - заполнение символом
- Generate(n, proc) - генерация по правилу
- Tabulate(n, proc) - табулирование

7. Работа с датами и временем

- FormaTime(format) - форматирование текущей даты
- ParseTime(format, s) - разбор строки с датой

8. Кодирования (Encodings)

- **ROT13:** Encode(s, encoding=rot13) - шифр Цезаря
- **Base64:** Encode(s, encoding=base64) - base64 кодирование
- **URL:** Encode(s, encoding=percent) - URL-кодирование
- **Декодирование:** Decode(s, encoding=...)