1. String

```
> # Строки
> restart :
> s1 := "A bird may be known by its song.";
```

$$s1 := \text{"A bird may be known by its song."} \quad (1.1)$$

```
> s1[1];
```

$$\text{"A"} \quad (1.2)$$

```
> length(s1);
```

$$32 \quad (1.3)$$

```
> s1[length(s1)];
```

$$\text{"."} \quad (1.4)$$

2. \n \t

```
> restart :
> s1 := "abc\n def";
> s1;
```

$$\text{"abc}$$
$$\text{def"} \quad (2.1)$$

```
> s1 := "abc\t def" :
> s1;
```

$$\text{"abc} \qquad \text{def"} \quad (2.2)$$

```
>
> s1 := "abc\" def" :
> s1;
```

$$\text{"abc" def"} \quad (2.3)$$

```
> s1 := "A"; whattype(s1);
```

$$s1 := \text{"A"}$$
$$string \quad (2.4)$$

```
> s1 := convert(s1, symbol); whattype(s1);
```

$$s1 := A$$
$$symbol \quad (2.5)$$

```
> s1 := convert(s1, string); whattype(s1);
```

$$s1 := \text{"A"}$$
$$string \quad (2.6)$$

3. StringTools

```
> restart :
> with(StringTools) :
> #Case Conversions
```

CamelCase Capitalize LowerCase OtherCase UpperCase

```
>
> CamelCase("hardwork");# capitalize each word in a string
```

"HardWork" **(3.1)**

```
> Capitalize("hardwork");
```

"Hardwork" **(3.2)**

```
> Capitalize("hard work");
```

"Hard Work" **(3.3)**

```
> UpperCase("hard work");
```

"HARD WORK" **(3.4)**

```
> LowerCase("HARD WORK");
```

"hard work" **(3.5)**

```
> OtherCase("HaRd wOrk");
```

"hArD WoRK" **(3.6)**

```
>
```

> ***#Character Class Tests***

Has	HasAlpha	HasAlphaNumeric	HasASCII
HasBinaryDigit	HasControlCharacter	HasDigit	HasGraphic
HasHexDigit	HasIdentifier	HasIdentifier1	HasLower
HasOctalDigit	HasPrintable	HasPunctuation	HasSpace
HasUpper	HasVowel	IsAlpha	IsAlphaNumeric
IsASCII	IsBinaryDigit	IsControlCharacter	IsDigit
IsGraphic	IsHexDigit	IsIdentifier	IsIdentifier1
IsLower	IsOctalDigit	IsPrintable	IsPunctuation
IsSpace	IsUpper	IsVowel	

> *HasPunctuation*("Hard work!");

true **(3.7)**

> *HasControlCharacter*("\tHard\nwork");

true **(3.8)**

> *IsPrintable*("\n \n \t\t\t")

false **(3.9)**

> *HasVowel*("AEOUI") # *гласные*

true **(3.10)**

> *HasVowel*("QWRTYPSDFGHJKLZCVBNM") ;

false **(3.11)**

> *IsGraphic*("\n\n\t\t");

false **(3.12)**

> *IsGraphic*("ABC DEF");

false **(3.13)**

> *IsGraphic*("@#$%%^^&&&···(((0))_?");

true **(3.14)**

> *IsOctalDigit*("01234567");

true **(3.15)**

> *IsOctalDigit*("01234567 89");

false **(3.16)**

> *IsBinaryDigit*("01");

true **(3.17)**

> *IsBinaryDigit*("01 2");

false **(3.18)**

> *IsHexDigit*("0123456789abcdef");

true **(3.19)**

> *IsHexDigit*("0123456789abcdef QWE");

false **(3.20)**

>

```
> #Combinatorics on Words
```

Border	BorderArray	BorderLength	Fibonacci
IsConjugate	IsDerangement	IsEodermdrome	IsPalindrome
IsPeriod	IsPermutation	IsPrimitive	LyndonFactors
MaximalPalindromicSubstring	MinimumConjugate	MonotonicFactors	Overlap
PatternCanonicalForm	PatternEquivalent	Period	PrimitiveRoot
ThueMorse			

```
> IsPeriod("abababa", 2); #определяет является ли число периодом слова
```

true **(3.21)**

```
> Border("CCsomeTextXX");
```

"" **(3.22)**

```
> Border("CCXXsomeTextCCXX");
```

"CCXX" **(3.23)**

```
> BorderLength("CCsomeTextCC")
```

2 **(3.24)**

```
> IsPalindrome("semitimes");
```

true **(3.25)**

```
> # Строка является перестановкой тогда и только тогда, когда каждый символ в
      строке встречается ровно один раз.
> IsPermutation("abc");
```

true **(3.26)**

```
> IsPermutation("abcba");
```

false **(3.27)**

```
> #Строка является примитивной, если ее нельзя записать как собственную степень
      другой строки.
> IsPrimitive("a");
```

true **(3.28)**

```
> IsPrimitive("aaa") ;
```

false **(3.29)**

```
> #является ли строка перестановкой другой строки
> IsDerangement("edit", "tide") ;
```

true **(3.30)**

```
> IsDerangement("foo", "oof");
```

false **(3.31)**

```
> # создать строку Фибоначчи
> Fibonacci(5, "X", "o") ;
```

"XoXXoXoX" **(3.32)**

```
>
```

```
> #Comparisons
```

Compare CompareCI IsPrefix IsSuffix
LeftRecursivePathOrder LexOrder RevLexOrder RightRecursivePathOrder
ShortLexOrder ShortRevLexOrder

```
> IsPrefix("", "abc");
```

true **(3.33)**

```
> IsPrefix("ab", "abc")
```

true **(3.34)**

```
> IsPrefix("AB", "abc")
```

false **(3.35)**

```
> IsSuffix("bc", "abc");
```

true **(3.36)**

```
> IsSuffix("BC", "abc")
```

false **(3.37)**

```
>
```

```
> #Constructors
```

Fill Generate Iota Random
Repeat Tabulate

```
> Fill("a", 5);
```

"aaaaa" **(3.38)**

```
> Repeat("abc", 5);
```

"abcabcabcabcabc" **(3.39)**

```
> #генерировать все строки заданной длины в некотором алфавите
> Generate(2, "ab");
```

["aa", "ab", "ba", "bb"] **(3.40)**

```
> #создать строку, содержащую  символы из заданного диапазона
> Iota(48, 57);
```

"0123456789" **(3.41)**

```
> #Random( len, alphabet )
  #Randomize( seed );
> Random(20, "Ax)");
```

")xA))))xA))))xA)AAAA" **(3.42)**

```
> Random(20, 'binary');
```

"11000000011001101110" **(3.43)**

```
> Random(20, 'upper');
```

"SEQBGBPIQPIGUHKMUGQF" **(3.44)**

```
> Tabulate(n ↦ `if`(n::even, "b", "a"), 10);
```

"abababababab" **(3.45)**

```
> Tabulate(n ↦ Char(n + Ord("A") − 1), 26);
```

"ABCDEFGHIJKLMNOPQRSTUVWXYZ" **(3.46)**

```
> Tabulate(n ↦ `if`(n::prime, "*", "."), 500);
```

```
".**.*.*...*.*...*.*...*.....*.*....*...*.*...*.....*.....*.*....*...*.*....*...*.....*.....*...*.*...*.*...*.....
    .....*...*....*.*.....*.*.....*....*...*.....*.....*.*.....*.*...*.*.....*.....*...*.*...*.....*.
    *.....*.....*.....*.....*.*.....*...*.*.....*.....*...*.*...*.....*.....*.....*.*...*.....*.....
    ..*.....*....*...*.....*.....*...*.....*.....*.*.....*.*.....*...*.....*.....*...*.*...*.....*.....*
    ...*.....*."
```

```
>
```

> ***#Date and Time Procedures***

FormatTime ParseTime

> *FormatTime*( )

"2023-09-18" **(3.48)**

> *FormatTime*("%d-%m-%Y")

"18-09-2023" **(3.49)**

> *dt* := *ParseTime*("%d-%m-%Y", "18-10-2023");

dt := *Record*(*calendar* = "Gregorian", *second* = 0, *minute* = 0, *hour* = 0, *monthDay* = 18, *month* = 10, *year* = 2023, *weekDay* = 4, *weekDayName* = "Wednesday", *yearDay* = 291, *dst?* = *false*) **(3.50)**

>

> ***#Encodings***

null	the null encoding (does nothing)
rot13	classical Caesar cypher on alphabetic characters
rot[n]	classical Caesar cypher on nonzero bytes
alpharot[n]	classical Caesar cypher on alphabetic (letter) characters
base64	base 64 encoding as described in RFC 2045.
percent	percent encoding for URLs

Decode DecodeEntities Encode EncodeEntities
Escape Visible

> *e* := *Encode*("a string", '*encoding*' = '*base64*')

e := "YSBzdHJpbmc=" **(3.51)**

> *Decode*(*e*, '*encoding*' = '*base64*')

"a string" **(3.52)**