# Types – quick reference

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Literal value example** | **Type name** | **Notes** |
| **Integer** | 42 | Int | Size? |
| **Floating Point** | 1.618  3.0 – must always include a decimal point | Float | An Int may be passed as an argument to a Float parameter |
| **Boolean** | true or false | Bool |  |
| **Single character** | 'q' | Char |  |
| **String** | "Hello" | String | Newline may be embedded as \n |
| **Date** | TBD | Date |  |
| **List** | ["apple", "orange", "pear"]  [1.0, 2.5, 3.0, 0.0, 2.0]  [[3,7,1], [5,5,7], [0,1,0]] | List<of String>  List<of Float>  List<of List<of Int>> | Members of a list must all be of the same type. To create an *empty* list:  new List<of String>() |
| **Dictionary** | ['a':121, 'b':23, 'c':35] | Dictionary<of Char, Int> |  |

## Int

An integer is a whole number i.e. with no ‘fractional’ component.

### Type name

Int

### Defining a literal integer

var meaningOfLife set to 42

### Default value

0

### Constraints

* Maximum value: 253 – 1 which is just over 9 x 1015
* Minimum value: -(253 – 1)

If either limit is exceeded the number will automatically be represented as a Float, with possible loss of precision.

### Notes

* An Int may always be passed as an argument into a method that requires a Float.

## Float

Float is short for ‘floating-point number’ – a number that may have both an integer and fractional part.

### Type name

Float

### Defining literal floating-point value

var a set to 1.618

### Constraints

Since Elan compiles to JavaScript, the constraints on floating point numbers are those of JavaScript:

* Maximum value: just over 1 x 10308
* Minimum value: approx. 5 x 10-324

For greater detail, refer to the official JavaScript documentation

### Notes

* A variable that has been defined as being of type Float may not be passed as an argument into a method that requires an Int, nor as an index into an ArrayList, *even if the variable contains no fractional part*. However, it may be converted into an Int before passing, using the functions floor() (the integer value left by removing any fractional part) or ceiling() (if the Float value *does* have a fractional part, the ‘ceiling’ will the lowest integer greater than the Float value).
* If you wish to define a variable to be of type Float but initialise it with a whole number then add .0 on the end of the whole number, for example: var a set to 3.0.

## Boolean

A Boolean value is either true or false.

### Type name

Boolean

### Defining a literal Boolean

var a set to true

true and false must be written lower-case

### Default value

false

## String

A String represents ‘text’ – a sequence of zero or more characters.

### Type name

String

### Defining a literal string value

var a set to "Hello"

String are always delineated by double-quote marks

### Default value

"" – known as ‘empty string’.

### Notes

* As on most programming languages, strings are *immutable*. When you apply any operation or function with the *intent* of modifying an existing string, the existing string is never modified. Instead the function or operation will return a *new* string that is based on the original, but with the specified differences.
* Strings may be appended using the plus operator, for example print "Hello" + " " + "World".
* A newline may be inserted within a string as \n, for example: print "Hello\nWorld".
* You may insert single-quote marks – ' – within a string.
* Elan strings are automatically interpolated: you may insert the values of variables, or simple expressions within a string, by enclosing them in curly-braces. For example (assuming that the variables a and b are already defined as integers) :  
   print "{a} times {b} equals {a\*b}.”
* It is not *currently* possible to include double-quote marks *within* a string. This is likely to be made possible in a future release.

## Date and Time

Dae and Time are not currently yet implement as standard types in Elan. They are likely to be introduced in a future release.

## Array-list

An ‘array-list’ is the simplest way to represent a collection of data items *of the same type*. It is called an ‘array-list’ because it offers the functionality both of a traditional array, including:

* access elements by index
* create an empty structure of a defined size

and of a traditional list (sometimes referred to as a ‘linked list’), including:

* be able to append to the list, extending it dynamically – starting from an empty list if desired
* be able to find elements within the list
* be able to insert an item between existing members of the list

### Type name

The type name for the array-list must specify the type of the elements, for example:

ArrayList<of String>

ArrayList<of Boolean>

ArrayList<of ArrayList<of Int>>

### Defining a literal array-list

A literal array-list is ‘delimited’ by square brackets, and the elements are separated by commas. The elements may be literal values (all of the same type):

var fruit set to ["apple", "orange", "pair”]

including ‘nested lists’:

var coordinates set to [[3.4, 0.1, 7.8],, 15.3] [1, 0, 1.5], [10, -1.5, 25]]

Or of variables of the same type, or a combination of literal values and variables.

or variables (provided they are all of the same type)

var values set to [x, y, z]

or a mixture of literal values and variables:

### Default value

The default value of any type of array-list is empty. This may be created by:

## var players set to new List<of String>()

### Constraints

* All the members of the list must be of the type specified (either explicitly in the name of the Type, or implicitly in the literal values with which the list was initialised).
* An array-list may be passed as an argument into a procedure, but may not be passed as an argument into a function because the latter may accept only *immutable* types, hence …

## Immutable-list

An ImmutableList has similar capabilities to an ArrayList but – just like a string – is *immutable*. You can still insert, delete, or change elements in an Immutable--Lst, but the methods for these operations do not modify the input list: they return a new list based on the input list but with the specified differences.

### Type name

The type name for the immutable-list must specify the type of the elements, for example:

ImmutableList<of String>

ImmutableList <of Boolean>

ImmutableList <of ImmutableList<of Int>>

### Defining a literal immutable-list

Like an array-list but delimited by curly-braces rather than by square brackets, for example:

var fruit set to {"apple", "orange", "pair”}

### Constraints

* Like an ArrayList the members of an ImmutableList must all be of the same type.
* Unlike an ArrayList, an ImmutableList may be passed as an argument into a function (and also to a procedure).

## Enum

An enum – short for ‘enumeration’ – is the simplest form of ‘user-defined type’ , specifying a set of values, each defined as a name, such that a variable of that type must always hold one of those values.

### Type name

The name given to an enum (see below), which must begin with a capital, is used as the Type name, when passing a value to or from a procedure of function.

### Defining an enum

Example

enum Status

incomplete, ready, running, stopped, invalid

end enum

* The name must begin with a capital letter, which may be followed by any combination of lower-or-upper-case letters, the underscore ­\_ symbol, or decimal digits (though it is not common practice to use decimal digits in enum names).
* The values that an instance of that type may take, are separated by commas. Each value must take the same form as a variable name i.e. start with a lower-case letter followed by any combination of lower-or-upper-case letters, the underscore ­\_ symbol, or decimal digits.

### Using an enum

The value is specified by the type name for the specified enum, followed by a dot and the value name, for example:

var x set to Status.ready

### Notes

* Enums are *read-only* – once they have been defined it is not possible to add, remove, or update the values.
* *Internally*, enum values are held as integers, with the first named value represented as 0. It is possible to use an enum value *as though it were an integer* – for example as an index into an array-list, or to compare two values of the same Enum type using the standard comparison operators (is, is not, >, etc).

## Tuple

A tuple is a way of holding values of *different* types together as one data structure.

### Constraints

Creating a new tuple

Accessing element from a tuple

Deconstructing

Tuple type

## Func

## Class

See also Class

## Default values

Every type in Elan – whether pre-defined (such as Int or String) or user-defined (see class and enum) – has a default value

# Variables

Variables are defined using the var

Variable names must commence with a lowercase letter, followed by any combination of lower-case or upper-case letters, numeric digits, and the underscore.

Variable names are type-sensitive.

Variables may only be defined within the main, a function, or a procedure – and are thereby scoped to that construct. (There is no such thing as a ‘global variable’ in Elan - although there are global constants).

## Variable definition

### Var

## Re-assigning a variable

### set

## Scope of variables

# Expressions

## Operators

### Arithmetic operators

+, - , \*, /, \*\*

Precedence and brackets

### Logical operators

is, not, and, or, xor, >, <, >=, <=

### Function call

### New Instance

#### With clause

# Global – constructs

What this term means and what they are

Illustrated by what’s visible in the global selector

# Main

Any program that you wish to *run* (as distinct from being a library intended for use within another program) must have a main.

The main can be defined anywhere within the file, but the convention is to define it at the top of the file

There may only be one main within a file, so if a main already exists, the main option no longer shows up as an option in the new code selector.

[what a main can contain]

# Function

## Defining a function

### Return statement

## Rules & patterns

Must return a value of the type specified in the signature

May have only a single return statement, which must be the last statement in the function – it is auto-created by the Function frame and may not be moved from the last position, nor deleted.

When the function is created the return statement takes the form: return result.

The result keyword references a variable, scoped to the function wherein it is used, defined (behind the scenes) to the type that the function must return and initialised to the default value for that type.

The result may be assigned to a new value within the function. [example]

It is not necessary to make any use of the result keyword, it is just a convenience. You may write, for example:

[function defining and writing own result]

And many functions may simply return the result of evaluating an expression [example]

Functions may not cause any side effects – such as creating output, or modifying a reference passed into the function as a parameter – and they may not depend upon any external information such as a on an input, on today’s date, or a random number generated inside the function – everything the function needs must be passed in as a parameter.

For this reason, the input, print, and external statements are not offered as options by the new code selector within a function body.

For the same reason

Example of invalid code:

* Attempting to set a parameter
* Attempting to pass in an array
* Attempting to call a procedure method on a class

## Using a function

May only be called as part of an expression e.g.

In a var, set, return

In the condition for a selection or iteration statement

As an argument or index defined in-line

# Procedure

## Example

## Rules

Can include input, output, or external dependencies

If the type passed in is a *reference type* and *mutable* (e.g. an Array or an of a user-defined class) then any change made to that parameter inside the procedure can be observed outside the procedure

Example – in-line sort

## Calling a procedure

### call statement

# Constant

## Examples

Numeric values

Literal lists or dictionaries

## Rules

Must define a literal value or data structure – cannot make use of functions or other constants

Always declared at global level.

## Using a constant

Example

Option to use the global qualifier to disambiguate from a variable.

# Class

See also Class Members

## Class Members

See also Class

# Enum

# Test

## Assert statement

# Comments

Comments must start with a # (hash symbol).

Comments must be defined on their own line.

For multi-line comments, each line must start with the #

# Statements

## Selection (conditional statements)

### switch

#### case clause

#### default clause

### if

#### else clause

#### else if

## Iteration (loops)

### for

### each

### repeat

### while

## Handling errors

### try

#### catch

### throw

## Input / output

### print

### input

### external

print and input are both examples of e