# Types – quick reference

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Literal value** | **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> |  |

## Integer

var a set to 42 – the type of a will be Int

function square(x as Int) return Int

## Float

const phi set to 1.618

var a set to 1.0 - the type of a will be a Float because there is a decimal point in the literal value.

function cubeRoot(x as Float) return Float

Where a parameter is defined as a Float, an Int argument may be supplied and will be automatically converted to a Float. The reverse is not true because that would involve potential loss of information.

## Bool

var a set to true – the type of a will be Int

function markAnswers(a as Bool, b as Bool, c as Bool) return Int

## Char

A single char is defined within single-quotes:

var a set to 'q' – the type of a will be Char

function count(c as Char, within as String) return Int

## String

A literal string is always defined using double-quotes:

var a set to "Hello" – the type of a will be String

function count(c as Char, within as String) return Int

Newlines may be specified within a literal string with \n:

print "Hello\nthere!" – will be printed over two lines.

See also Interpolated Strings

## Date

TBD

## List

A literal list is delimited by square brackets. All items within a list must be of the same type:

## const fruit set to ["apple", "orange", "pear"]

## var scores set to [1.0, 2.5, 3.0, 0.0, 2.0]

To create a new empty list it is necessary to specify the type of elements that it will contain:

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

## var names set to default List<of String>

## Enum

## Array

## Tuple

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