Types and Variables

- Local variables don’t have to be declared, but do it anyway!
- Global variables must be declared.
- An variable that has *not* been declared will automatically be treated as a local variable.
- Icon is dynamically typed. This means that
  - You don’t need to declare the types of variables.
  - A variable may contain different types of data at runtime.

```icon
local X
X := "hello" # String
X := 5 # Integer
X := 6.7 # Real
```
... You won’t get type errors at compile-time, but you will get them at run-time:

```
procedure main(args)
    t := "hello" + 4.5
end
```

Run-time error 102
File t.icn; Line 6
numeric expected
offending value: "hello"
Trace back:
main()
{"hello" + 4.5} from line 2
Types and Variables...

- `type(V)` will return the *name* (a string) of the type of `V`:

  ```
  record complex(a,b)
  t := "hello"
x := type(t)  # x="string"
t := [5,6,7]
x := type(t)  # x="list"
t := complex(4,5)
x := type(t)  # x="complex"
  ```

- Some data types are automatically converted to the required type. For example, a string (consisting entirely of digits) can be converted into a number, explicitly or implicitly:

  ```
  write(5 + "6")  # implicit
  write(5+integer("6"))  # explicit
  ```
Icon Statements

while e1 do e2  Evaluate e2 until e1 fails.
until e1 do e2  Same as while not e1 do e2.
repeat e  Evaluate the expression e repeatedly.
break  Jump out of the most closely nested loop.
next  Jump to the beginning of the most closely nested loop.

{ ... }  Compound statement.

a := b  Assignment. Repeated assignments (a := b := c) are also OK.
If-expression. If \( e_1 \) succeeds then evaluate and return the value of \( e_2 \), otherwise evaluate and return \( e_3 \). The else-part is optional.

Similar to repeated if-expression: if \( e_1 \) then \( s_1 \) else if \( e_2 \) then \( s_2 \) else... else \( s_3 \). The default-part is optional. \( e_1, e_2,... \) can be arbitrary expressions of arbitrary type, not just scalar constants as in Pascal.
Expressions

There are fundamental differences in the way Pascal & Icon statements are executed:
1. Icon statements are expressions that return values.
2. Icon expression either succeed or fail.

Failure doesn’t necessarily mean that something has gone wrong, rather, it means that there is no value to return. `numeric("pi")` fails because "pi" cannot be converted to number.

Success & Failure

\[ i + j \] Succeeds and returns the value \( i + j \).

\[ i < j \] Succeeds if \( i < j \), in which case \( j \) is returned. Fails otherwise.
Expressions...

Example: numeric

\[
\text{numeric}(x) \quad \text{Converts } x \text{ to a number.}
\]

\[
\text{numeric}("3.14") \quad \text{Returns 3.14.}
\]

\[
\text{numeric}("\text{pi}") \quad \text{Fails.}
\]

- All Icon variables have a special null value initially.
Expressions...

Success & Failure

stop(s) Write s and terminate.

expr1 | expr2 Generate the values from expr1, then from expr2.

x | y Generate the variables x and y.

0 | 1 Generate the values 0 and 1.

if i = (0 | 1) then write("ok") Write "ok" if i is 0 or 1.

every i := (0 | 1) do write (i) First write 0 then 1.
Expressions...

Success & Failure...

every (x | y) := 0  x := 0;  y := 0

x := p() | stop("error")  If p() fails, then stop and write "error".

j := i + 10  This fails if i = null.

/x  Succeeds (and produces null) if x = null. Fails otherwise.

/x := 0  Assign 0 to x if x = null.

/\x  Succeeds and produces x if x \neq null. Fails otherwise.

/\x := 0  if x \neq null then x := 0.