Flowchart-based programming

Andrea Sterbini – sterbini@di.uniroma1.it

Flowcharts

Flowcharts show the possible execution paths of the program

Every program has a single input and single output (initial edge)

An edge can be a sub-flowchart/component with single IN/OUT

- single-thread execution (but what about fork/join?)
- Flowgorithm flowgorithm.org
- Algobuild algobuild.com
- Raptor raptor.martincarlisle.com (with OOP!)
- Visual Logic visuallogic.org
- PseInt pseint.SF.net (in Spanish)

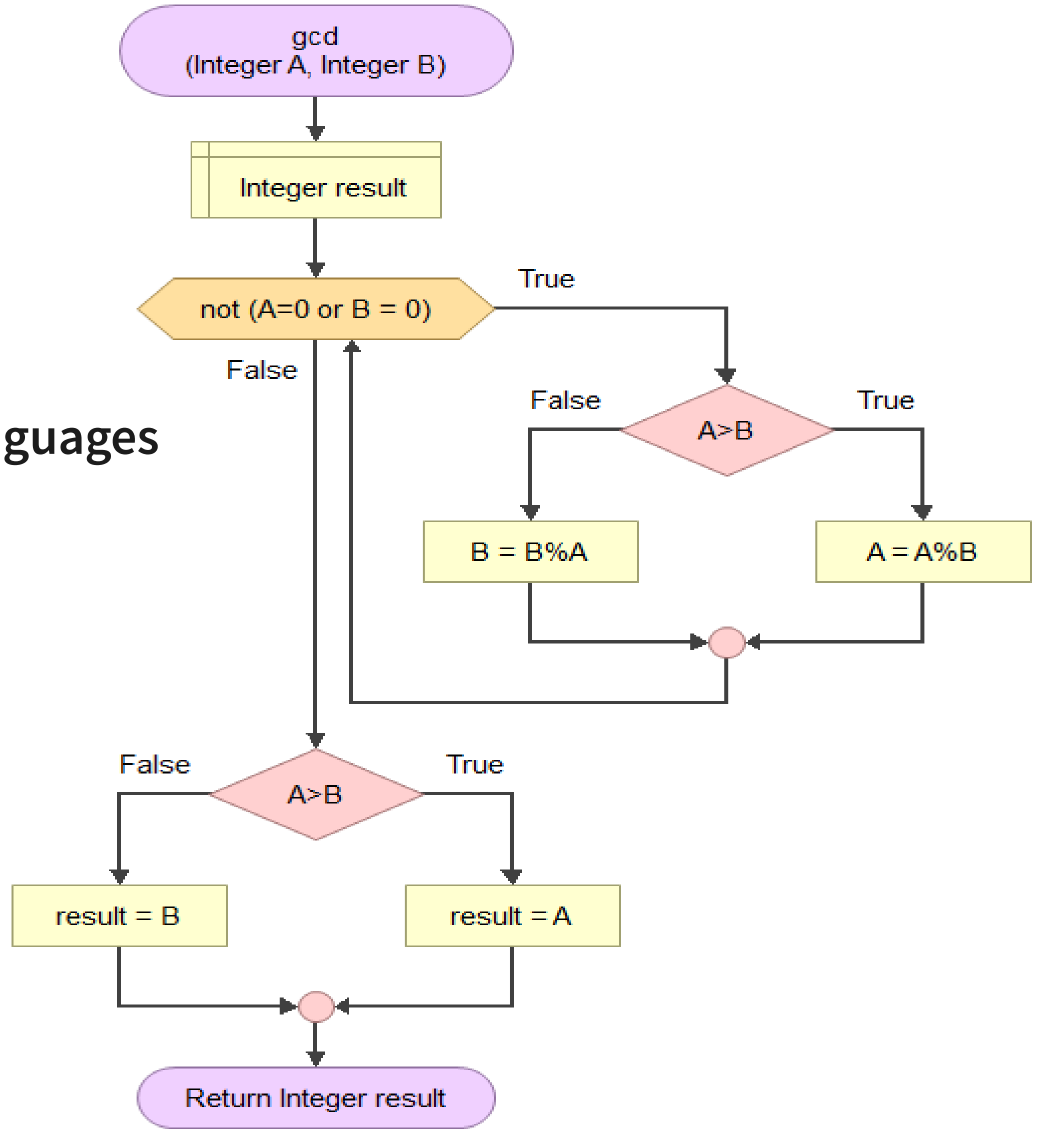
Flowgorithm = Flow-chart + Algorithm

Executable flow-charts

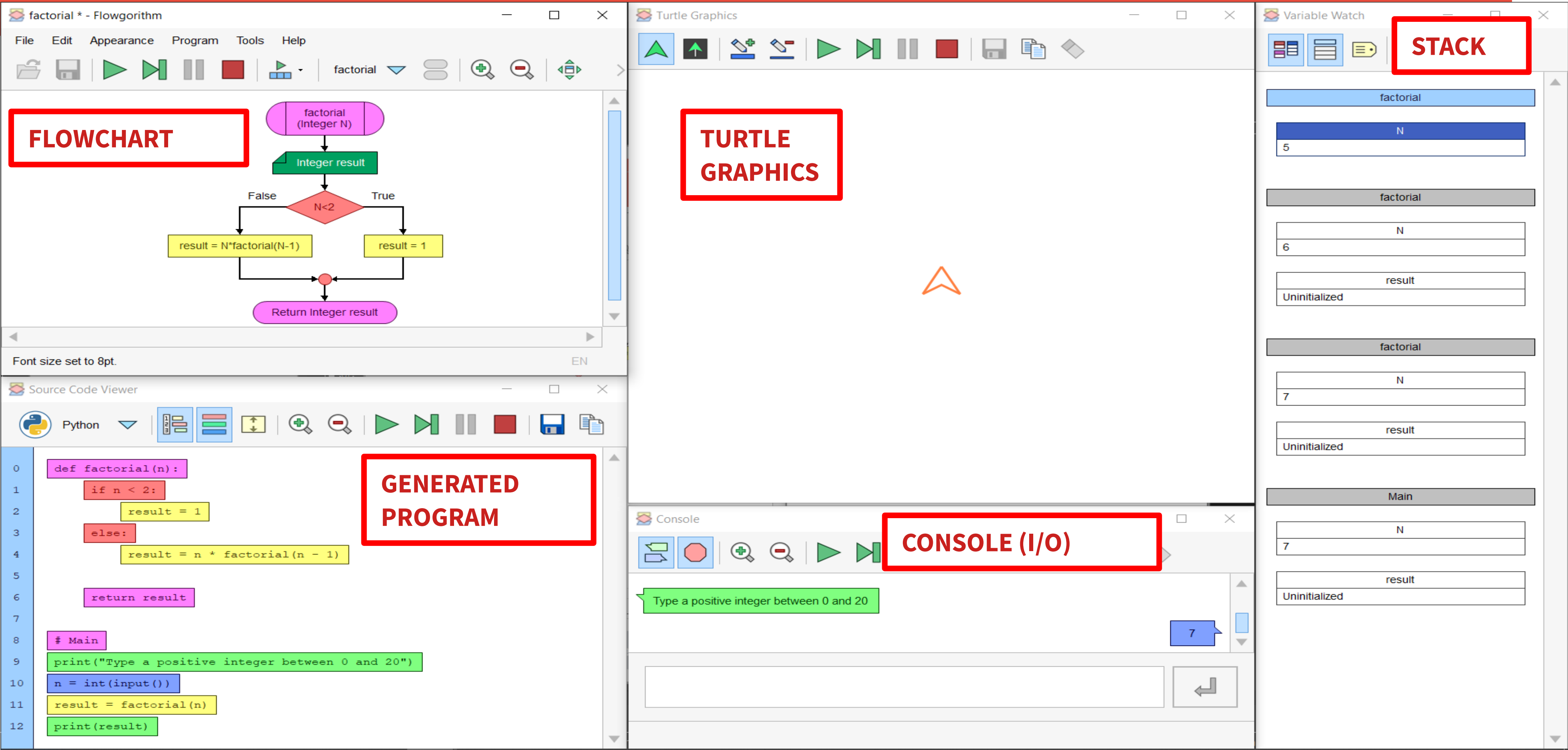
Personalized flow-chart STYLE and COLOURS

Generate your code in many programming languages

MISSING: loading a program source and generating its flow-chart
(BUT there are tools for that)
- code2flow.com
- …



Flowgorithm IDE windows



```
def factorial(n):
    if n < 2:
        result = 1
    else:
        result = n * factorial(n - 1)

    return result

# Main
print("Type a positive integer between 0 and 20")
n = int(input())
result = factorial(n)
print(result)
```

Only simple data types (and 1 dimensional arrays)

T = Integer, Float, String, Boolean

1 dimensional Array of <T>

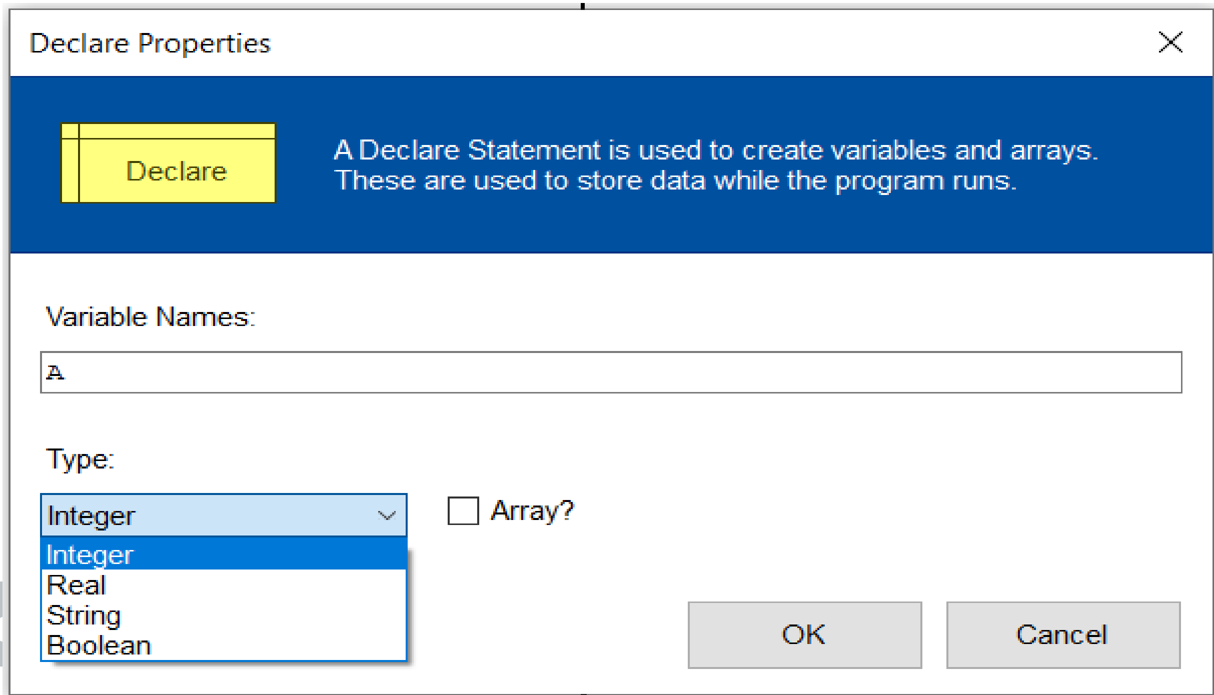
NO bigintegers (like Python) → you must consider range of possible values

NO lists or dynamic arrays
NO heterogeneous arrays
NO multidimensional arrays

NO objects

NO coroutines

NO function objects


Declare Properties

A Declare Statement is used to create variables and arrays. These are used to store data while the program runs.

Variable Names: A

Type: Integer

Array? [ ]

Statements

DECLARE/ASSIGN variable

INPUT & OUTPUT text or number

IF-THEN-ELSE

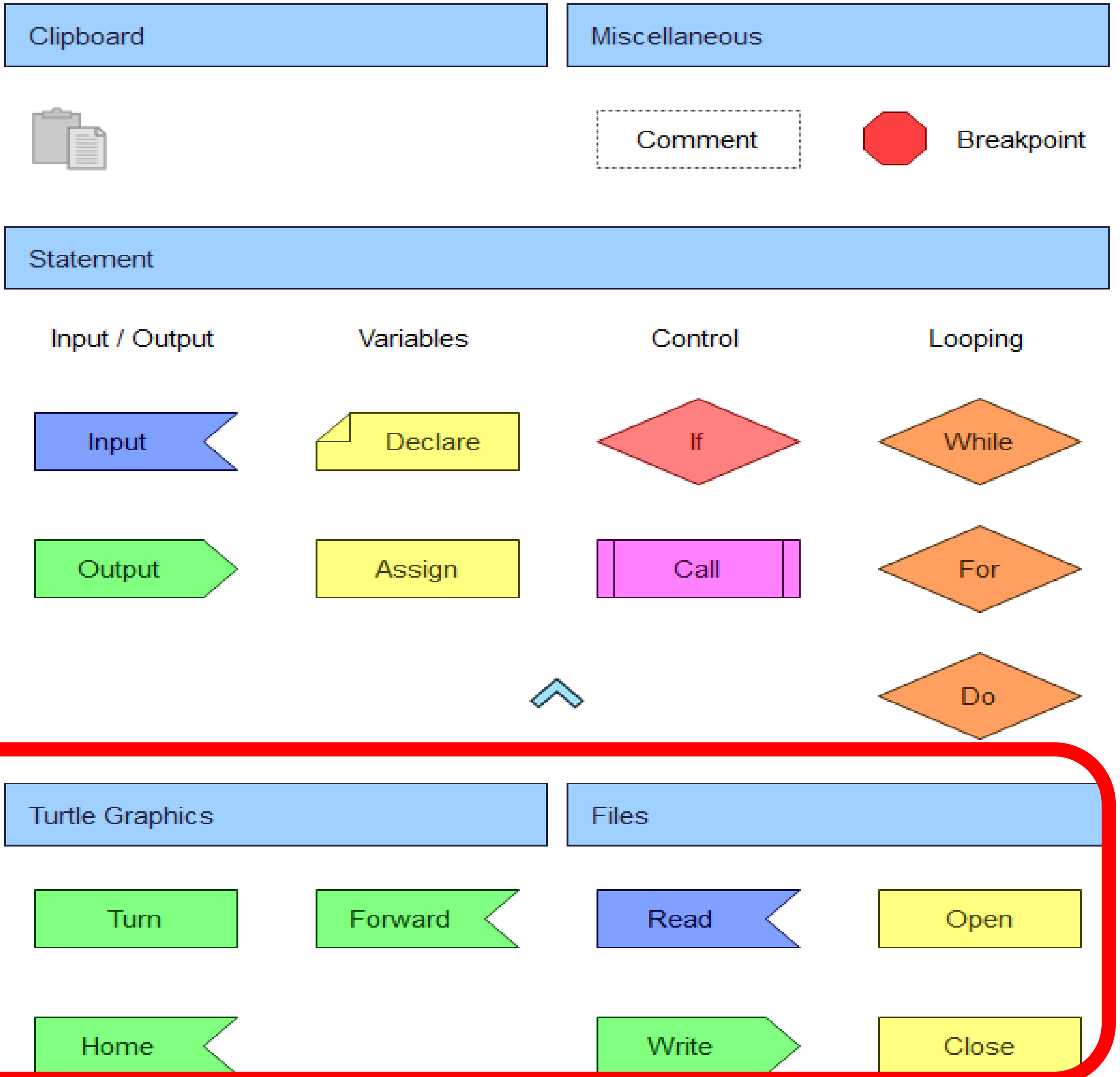
CALL procedure/function

WHILE-do / counted FOR / DO-while
(but NO foreach)

COMMENTS & BREAKPOINTS

TURTLE GRAPHICS (new!!!)

FILES I/O (new!!!)
ONLY 1 open file at a time!!!



Expressions and operators

Function calls

Logic:	and, or, not, comparison
Math:	+, -, *, /, %, ^, sign, trigonometry, log/pow, random, round
String:	concat, len, char(S, i)
Arrays:	size
Conversions:	char, ascii, int, float, str, round

Precedence of operators in expressions as usual

Control flow

Functions?	YES	
args by reference?	NO	(except for arrays like C)
multiple return values?	NO	(single simple types only)
Recursion?	YES	
ONE entry and ONE exit per function/diagram		
NO early return		(use an IF to skip the rest of the code)
NO break		(use an IF to skip the rest of the code)
Multiple assignments?	NO	
Concurrency/multi threading?	NO	
Events?	NO	
Exceptions?	NO	(errors are shown but you cannot catch them)

Programming style

PROCEDURAL/SEQUENTIAL?	YES	
FUNCTIONAL?	NO	no functions as arguments
STRUCTURED?	YES	
DECLARATIVE?	NO	
EVENT-BASED?	NO	
CONCURRENT?	NO	
MODULARIZATION?	YES	by function/procedure
ANALYSIS		
TOP-DOWN?	YES	
BOTTOM-UP?	NO	
OBJECT-ORIENTED?	NO	no objects

Debug support

Step-by-step execution		(both flow-chart AND generated code)

NOTE: the generated code is NOT executed (only shown)

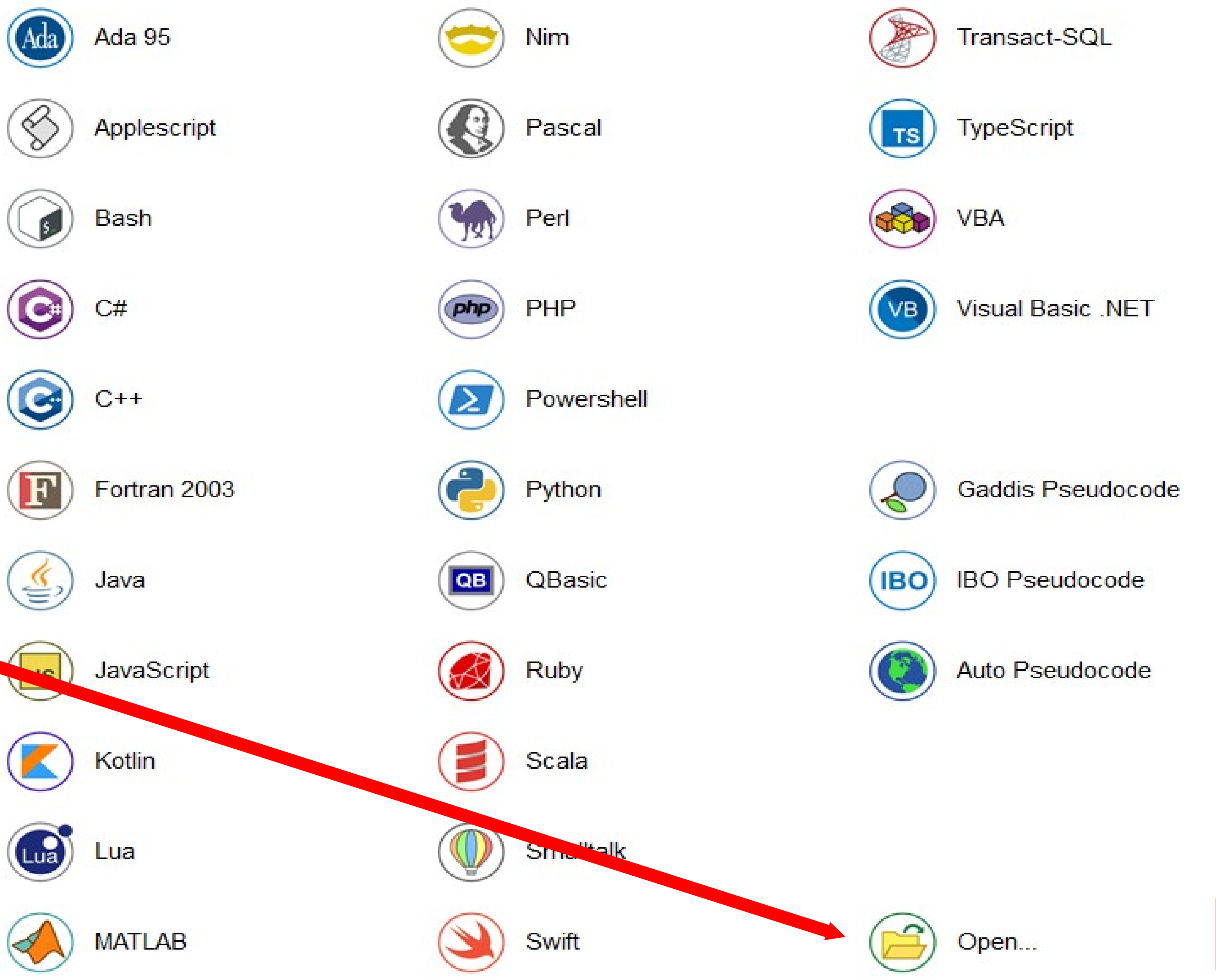
View Variables content		(both simple values and arrays)
Show the Stack content		(good to understand recursion)
Breakpoints		(then step by step)
Assertions?	NO	(add if-then by hand)
Exceptions?	PARTIAL	(some errors are generated, but cannot be handled)

IDE support

Refactoring	PARTIAL	(cut/paste flowchart into new functions)

Code is generated by templates

Code generation from flow-charts to many programming languages (custom also)



Ada 95
Applescript
Bash
C#
C++
Fortran 2003
Java
JavaScript
Kotlin
Lua
MATLAB

Nim
Pascal
Perl
PHP
Powershell
Python
QBasic
Ruby
Scala
Smalltalk
Swift

Transact-SQL
TypeScript
VBA
Visual Basic .NET
Gaddis Pseudocode
IBO Pseudocode
Auto Pseudocode
Open...

Example template: Python

A section with some global info (keywords, ext, case-sensitive ...)

The template contains required imports and definitions for some missing functions (you can extend it if you like)

Types are mapped to corresponding Python types

Diagram elements map to corresponding templates

Each Flowgorithm expression operator or intrinsic function is mapped to the corresponding Python one (with precedence levels)

Functions definition and call templates

DEMO

Literate programming / Documentation?

Program properties:

- Title, Author, Description
- BUT: they are NOT present in the generated code!!!

Comments in the flow-chart

NO free text

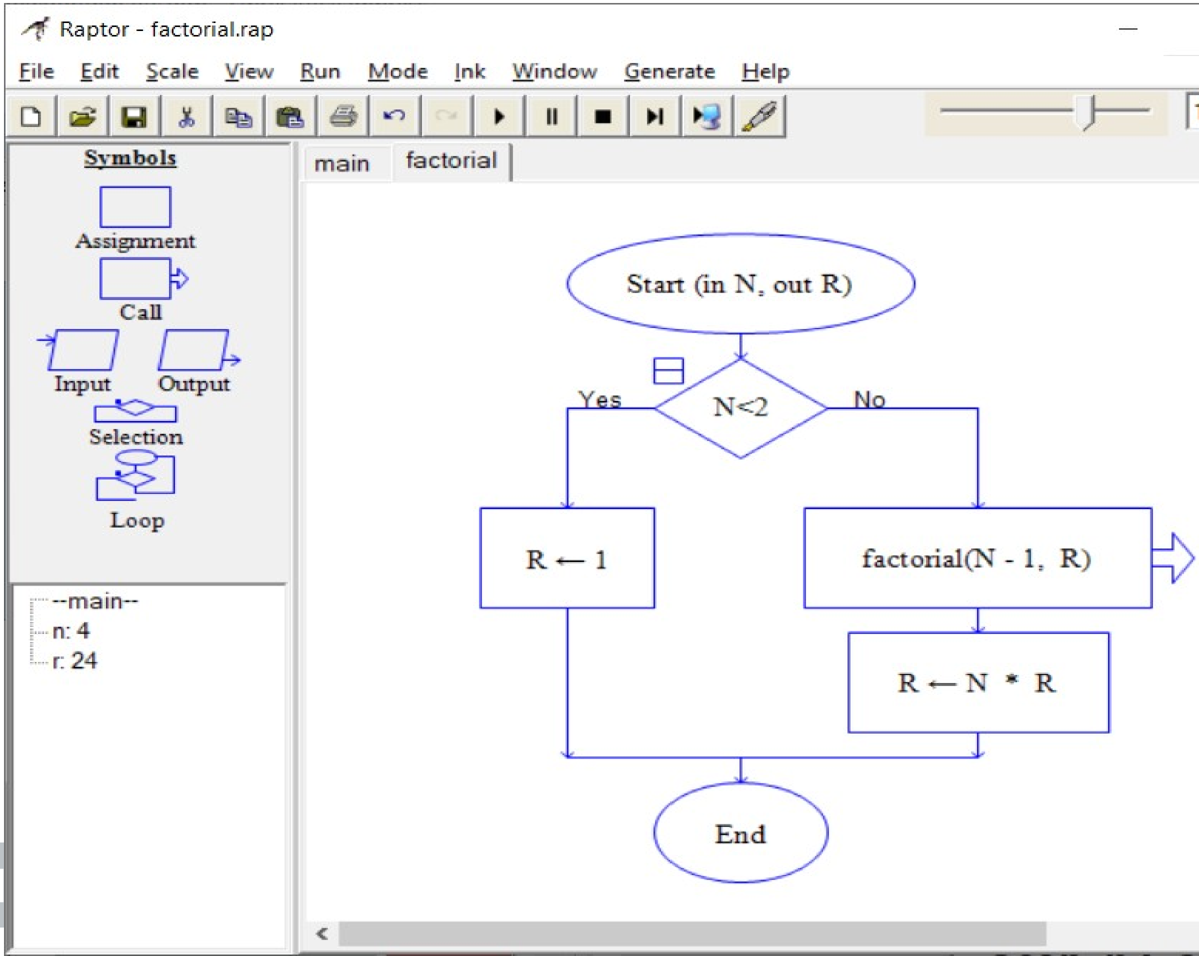
Examples

DEMO

(segue)

Raptor

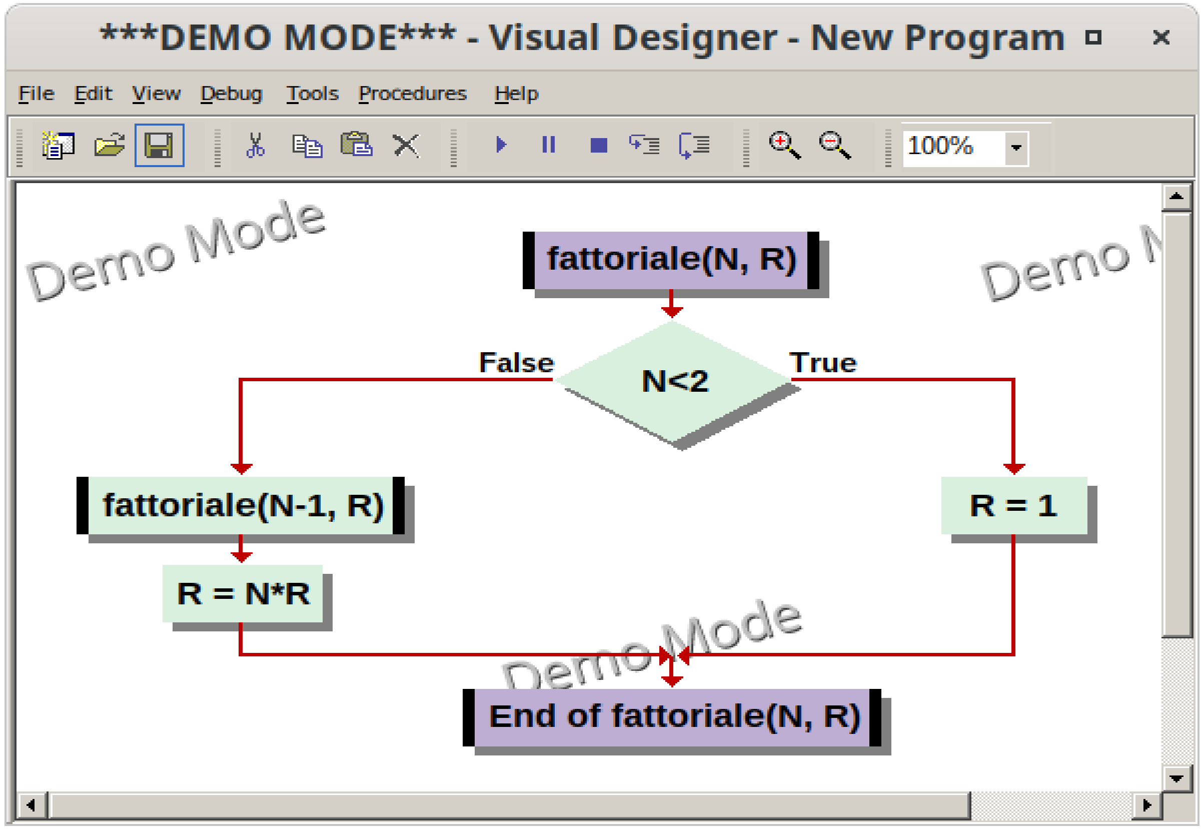
Procedures	YES
(with IN/OUT args)	
Recursion	YES
Functions	NO?
(procedures + OUT args!!!)	
<u>OOP</u>	<u>YES</u>
<u>Sub-charts</u>	<u>YES</u>
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	YES
Ada, C#, C++, Java, VBA	



```
--main--
n: 4
r: 24
```

Visual Logic

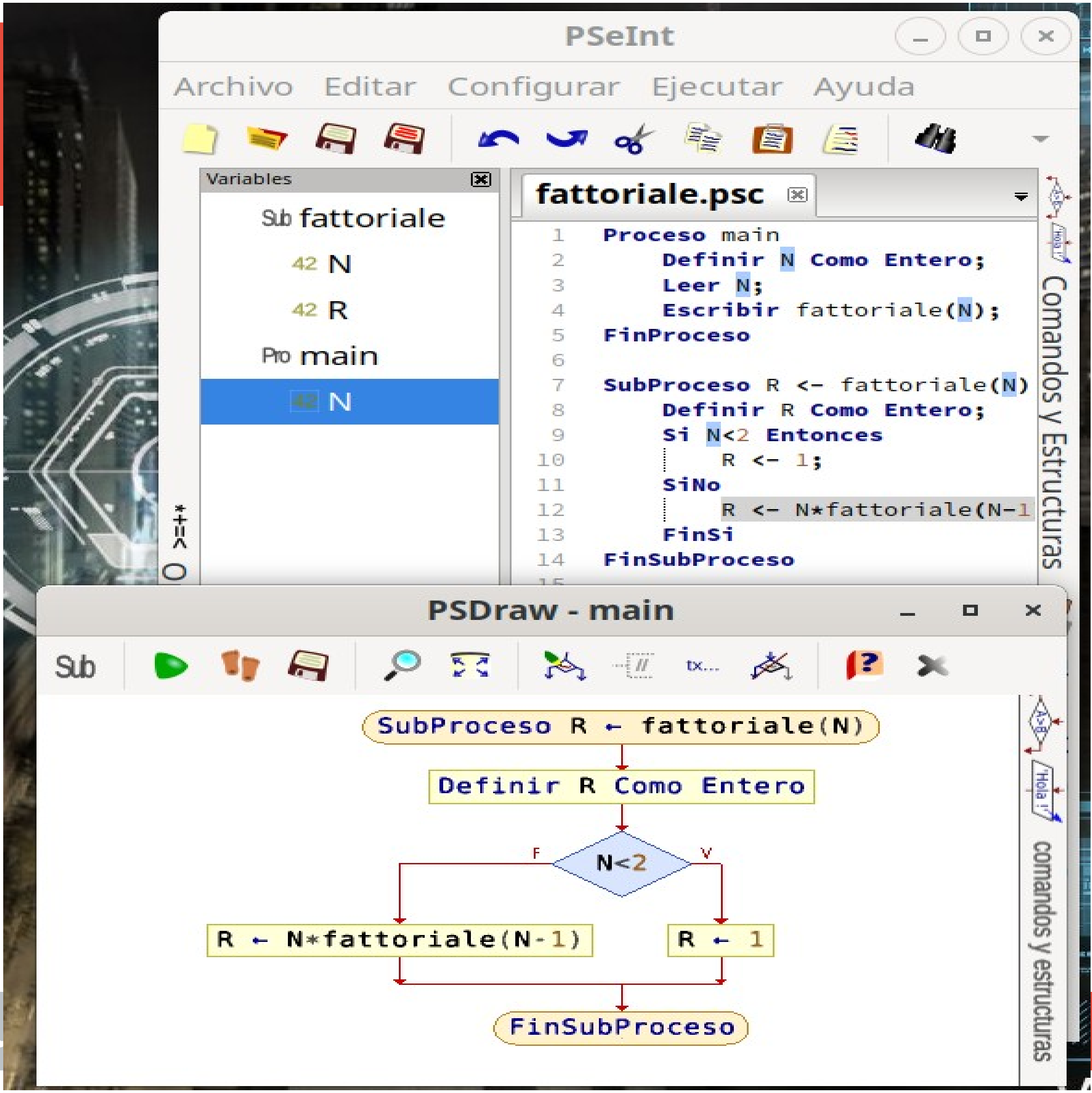
Procedures (with IN/OUT args)	YES
Recursion	YES
Functions (procedures + <u>OUT args!!!</u>)	NO?
OOP	NO
Sub-charts	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation Visual Basic + Pascal	YES



PseInt (Spanish only)

Procedures	YES
Recursion	YES
Functions	YES
OOP	NO
Sub-charts	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	YES

- C, C++, C#, Java, JavaScript, MatLab
- Pascal, PHP, Python 2/3
- Qbasic, Visual Basic ...



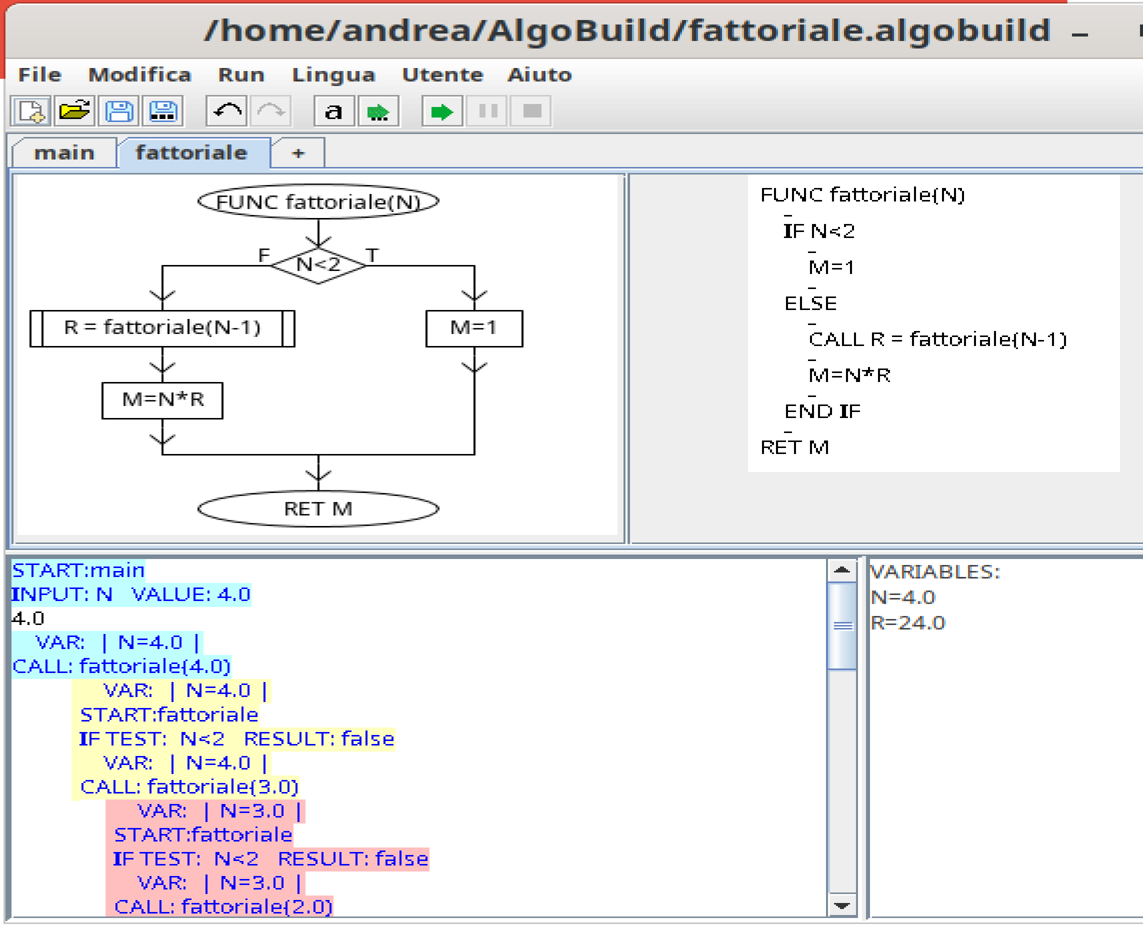
```
Proceso main
    Definir N Como Entero;
    Leer N;
    Escribir fattoriale(N);
FinProceso

SubProceso R <- fattoriale(N)
    Definir R Como Entero;
    Si N<2 Entonces
        R <- 1;
    SiNo
        R <- N*fattoriale(N-1
    FinSi
FinSubProceso
```

AlgoBuild

Functions	YES
Recursion	YES
Simple data types	
- numbers, strings, 1D arrays	
Complex types	NO
OOP	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	NO

Nice tracing of recursion with indentation



```
FUNC fattoriale(N)
  IF N<2
    M=1
  ELSE
    CALL R = fattoriale(N-1)
    M=N*R
  END IF
RET M
```

```
START:main
INPUT: N   VALUE: 4.0
4.0
  VAR:  | N=4.0 |
CALL: fattoriale(4.0)
        VAR:  | N=4.0 |
      START:fattoriale
      IF TEST:  N<2   RESULT: false
        VAR:  | N=4.0 |
      CALL: fattoriale(3.0)
          VAR:  | N=3.0 |
        START:fattoriale
        IF TEST:  N<2   RESULT: false
          VAR:  | N=3.0 |
        CALL: fattoriale(2.0)
```

```
VARIABLES:
N=4.0
R=24.0
```

Demo

DEMO