Typical scripting language

* Types:
  + Primitives: i64, f64, bool
  + Arrays of primitives: [i64], [f64], [bool]
  + Timeserieses of primitives: #i64, #f64, #bool
* Construction of:
  + Primitives
    - Literals
  + Arrays
    - Literals
  + Timeserieses
    - "Literal" s(signal-id: string) and s(signal-id: string, lag: i64)
      * Before the expression evaluation we walk through the code and match a regex that ensures that "s" is invoked with a string literal and (if 2nd parameter present) a integer literal. Everything else gives an error. These literals are collected to know the involved signals and their lags.
* Methods on:
  + Primitives
    - None
  + Arrays
    - forEach(fn: Closure): void
    - map(fn: Closure): [i64]/[f64]/[bool]
    - count(): i64
    - if i64 or f64
      * sum(): f64 (Returns NaN on any NaN summand and 0 on an empty array)
      * product(): f64 (Returns NaN on any NaN factor and 1 on an empty array)
      * max(): f64 (Ignores any NaN argument and returns NaN on an empty array)
      * min(): f64 (Ignores any NaN argument and returns NaN on an empty array)
      * avg(): f64 (Ignores any NaN argument and returns NaN on an empty array)
      * median(): f64 (Ignores any NaN argument and returns NaN on an empty array)
    - if bool
      * all(): bool
      * any(): bool
  + Timeserieses
    - at(time: i64): i64/f64/bool (Returns NaN/0/false if there is no datapoint and no predecessor)
    - slice(time: i64, from: i64, to: i64): [i64]/[f64]/[bool]
    - steps(time: i64, from: i64, to: i64, step: i64): [i64]/[f64]/[bool]
    - times(): #i64
    - if f64
      * trend(time: i64, from: i64, to: i64): f64 (Ignores any NaN argument and returns NaN on an empty array)
      * offset(time: i64, from: i64, to: i64): f64 (Ignores any NaN argument and returns NaN on an empty array)
* Language:
  + Conditionals
  + Loops
* Operators / Functions:
  + any combination of primitives or timeseries as input parameters
  + if at least one of the input parameters is a timeseries the result is too
  + if at least one of the input parameters is NaN the result is too or (if bool) "false"
  + Bool Unary Operators: bool → same:
    - !
  + Bool Binary Operators: bool x same → same:
    - &
    - |
  + Bool Binary Comparison Operators: bool x same → bool
    - ==
    - !=
  + Number Unary Operators: i64 or f64 → same:
    - +
    - -
  + Number Binary Operators: i64 or f64 x same → same:
    - +
    - -
    - \*
    - /
    - ~ (power)
    - % (remainder)
  + Number Binary Comparison Operators: i64 or f64 x same → bool:
    - ==
    - !=
    - >
    - >=
    - <
    - <=
  + Float Unary Functions: f64 → same:
    - General:
      * abs (absolute value)
      * ceil (smallest integer greater or equal to x)
      * floor (greatest integer smaller or equal to x)
      * fract (fractional part)
      * round (next integer, halfway values away from 0)
      * signum (1 for positive, -1 for negative numbers)
      * trunc (integer part)
    - Arithmetics
      * cbrt (cubical root)
      * exp (e power x)
      * exp2 (2 power x)
      * exp\_m1 ((e power x) - 1)
      * ln (logarithm base e)
      * ln\_1p (ln(x+1))
      * log10 (logarithm base 10)
      * log2 (logarithm base 2)
      * recip (1/x)
      * sqrt (square root)
    - Trigonometry
      * acos
      * acosh
      * asin
      * asinh
      * atan
      * atanh
      * cos
      * cosh
      * sin
      * sinh
      * tan
      * tanh
  + Float Binary Functions: f64 x same → same
    - atan2 (four quadrant arctangent)
    - copysign (value from x and sign from y)
    - hypot (hypotenuse length with legs x and y)
    - log (logarithm of x with base y)
    - max (greater number of x and y, if only one argument is NaN the other one is returned)
    - min (smaller number of x and y, if only one argument is NaN the other one is returned)
  + Integer Unary Functions: i64 → same:
    - abs (absolute value)
    - signum (1 for positive, -1 for negative numbers, 0 for 0)
  + Integer Binary Functions: i64 x same → same:
    - div\_euclid (euclidean division)
    - max (greater number of x and y)
    - min (smaller number of x and y)
    - rem\_euclid (euclidean remainder)
  + Conversion Functions:
    - to\_int(x: f64): i64 (Returns 0 on NaN)
    - to\_float(x: i64): f64
    - to\_int(x: bool): i64
    - to\_float(x: bool): f64