Mike Hahn - hahnbytes@gmail.com

Boardalog

Boardalog is a system used for 2-way communication between a teacher and a student, or between a presenter and an audience. Students and audience members are called members. The teacher/presenter and the member each use a Windows computer or a smartphone (a combination which is not allowed is teacher on a Windows computer and member on a smartphone). Any functionality tied to a specific subject, such as math, is written in a new programming language called Jyalog, along with a text markup language called Jyatags. The organization which hosts the teacher/presenter pays $5/year for each member, up to 50 members, and $1/year for the 51st and all subsequent members.

Boardalog in a Nutshell

Boardalog (whiteboard + dialog) is excellent for teaching math and computers. The teacher sits next to, in same room as, or in different location than the student(s). The teacher's smartphone syncs a portion of the student's screen. Alternatively, the teacher's laptop uses desktop sharing software to interact with the student's screen. A chat window takes care of the student's questions and the teacher's instructions. A specialized whiteboard, written in an embedded scripting language, is used to teach math. The teacher's employer pays a subscription fee based on the number of students.

Teaching Locally

Members display the curriculum on a Windows computer or smartphone, running a whiteboard called the JyBoard. Teachers display a window of the member's screen on a smartphone (held in landscape orientation unless the member is also using a smartphone). Bluetooth is used to keep the 2 screens synchronized. Both parties are in the same room or sitting at the same computer.

Teaching Remotely

Teachers can teach remotely using a Windows computer instead of a smartphone. A chat window facilitates communication between teacher and member. The teacher runs desktop sharing software and the member runs the JyBoard, or an always on top chat window. The JyBoard can be ported to Linux and Mac, for remote/local teaching, should a sufficient market exist.

Streamalog

Streamalog for Android is implemented after Boardalog for Android, then Boardalog is ported to iOS, and finally Streamalog is ported to iOS. Streamalog is a smartphone app used by organizations to facilitate user interactions, which uses Jyalog as an embedded scripting language.

JyBoard for Math

The JyBoard supports math being taught, using text in monospaced mode. The most commonly used commands are as follows:

- Use the arrow keys to move the cursor.
- Type underscore(s) to underline the numerator of a fraction.
- Use the special character command (Ctrl+K) to insert special characters such as pi, square root, sum, and integral.
- Use Tab/Shift+Tab to display/undo the next step in the math problem being solved.
- Type question mark (?) to explain the current step or to break the current step down into lower-level steps.
- Click on Help after typing question mark to access the help system.

Miscellaneous commands:

- Use asterisk and slash for multiply and divide.
- Fractions or matrices enclosed in brackets use tall brackets.
- Smart down/up arrow: press it after inserting a character moves the cursor beneath/above that character.
- Functions such as lines and parabolas can be plotted interactively on a graph.

- The default-to-upper-case setting assumes that all letters entered are upper case (use the shift key to enter a lower case letter), so Caps Lock is unnecessary.

Expression Language

Mathematical expressions are encoded (internally) using the Jyalog programming language. Each step in the math problem being solved manipulates this Jyalog expression. Even if the user enters steps in a different order than the default ordering, the simplification logic can handle that. The user can type Tab/Shift+Tab to redo/undo her previous step, as well as to redo/undo the computer's previous step.

Advanced JyBoard Commands

These next 2 paragraphs may be ignored, they are written in computerese. Use Shift+Arrow Key to highlight a rectangular block. Press Insert to insert a row or column of spaces before a highlighted block (insert blank line if no highlight). Press Shift+Insert/Delete to insert/delete an entire row/column when a block is highlighted. Press Enter at end of a line of text: insert blank line, back up on that line to line up with beginning of text on previous line. Press Enter on blank line to back up to line up with beginning of text on a previous line, or insert blank line if already at beginning of line. Press Ctrl+Tab to move forward to line up with beginning of first or next word on a previous line. Press Home to move to beginning of text on current line, press it again to toggle between beginning of line and beginning of text. This usage of Enter, Tab and Home is useful for editing program code with multiple indentation levels. The user doesn't have to memorize these commands: type question mark at any time to access the help system.

Superscripts

Superscripts and subscripts in monospaced mode are handled by employing a vertical offset of half a line per level of superscripting or subscripting. The caret symbol (^) is used as a superscript prefix, double-caret (^^) is used as a subscript prefix, and backslash (\) is used as an escape character (terminate super/subscript with a semicolon). Carets and double-carets cannot be mixed (exception: one level of superscript can be combined with one level of subscript).

Implementation Steps

1. Develop foundation of Jyalog code execution - ***almost done!***
2. Approach Progress Place: teaching computers
3. Approach West Neighbourhood House: teaching math
4. Develop rest of Jyalog code execution
5. Release Jyalog as console-based compiler on GitHub
6. Implement GUI: monospaced mode
7. Release Jyalog/GUI on GitHub
8. Write Jyatags design specs
9. Develop Jyatags
10. Integrate Jyalog with Jyatags
11. Jyalog/Jyatags: Jyalog Runtime Environment (JYRE)
12. JYRE for Linux is open source
13. Implement monospaced JyBoard in Java: teaching math/computers
14. Expand upon JyBoard using Jyalog and Jyatags
15. Implement Android app: Streamalog image collection manager
16. Use Specialisterne to hire contract Android programmer on spectrum:
    - they find tech jobs for those on autism spectrum
17. Port monospaced JyBoard to Android
18. Port JYRE to Android
19. Make pitch to DMZ tech incubator
20. Begin search for angel investor
21. Develop Jyalog code editor
22. Develop foundation of Streamalog chat
23. Add speech-to-text
24. Add text-to-speech
25. In case speech/text is problematic, abandon Streamalog development
26. Implement Jyalog version of image collection manager
27. Implement Jyberland
28. Expand code editor to Jyalog SDK
29. Implement Keyboard Aid (bells and whistles of editor)
30. Develop WYSIWYG Jyatags screen editor
31. Develop monetizing functionality
32. Perform beta testing
33. Launch websites
34. Purchase Google AdWords advertising
35. Without angel investor, do not renew contract of Android programmer
36. Use Specialisterne to hire remote iOS programmer on spectrum
37. Implement iOS app: Streamalog image collection manager
38. Convert monospaced JyBoard to iOS
39. Convert JYRE to Swift/iOS
40. Convert JyBoard to iOS
41. Convert foundation of Streamalog chat to iOS
42. Add speech-to-text for iOS
43. Add text-to-speech for iOS
44. Implement Jyalog/iOS version of image collection manager
45. Convert Jyberland to iOS
46. Perform beta testing
47. Launch iOS versions

About Us

I am Mike Hahn, the founder of Boardalog.com. I was previously employed at Brooklyn Computer Systems as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at Fred Victor on Tuesday afternoons, where for 5 years I taught math, computers, and literacy, and became a volunteer math/computer tutor at West Neighbourhood House. I quit that job in mid-2019. I have a part-time job working for a perfume store. My hobbies are reading and I often go for walks. I don't read books very often, but on March 19, 2021 I started reading a biography of Steve Jobs which my brother gave me. I read the CBC news website, news/tech articles on my Flipboard app, and miscellaneous articles on my phone (same screen as my Google web page). I visit my brother once a month or more. For almost 30 years I was depressed on and off (I'm a rapid cycler), but it largely vanished after I ramped up development of my previous Aljegrid project in early March 2021.

Jyalog

Jyalog (implemented in Java) is an open source Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes, also statements are separated by semicolons). Jyalog source files have a .JYG extension. Jyatags files (the sister language of Jyalog, a text markup language) have a .JYTG extension. Jyalog boasts an ultra-simple Lisp-like syntax unlike all other languages.

Special Characters

( )	grouping
-	word separator
;	end of stmt.
:	dot operator
"	string delimiter
\	escape char.
#	comment
_	used in identifiers
$	string prefix char.
{ }	block comment

Op Characters

+ - * / %

= < >

& | ^ ~ ! ?

Differences from Python

- Parentheses, not whitespace
- Operators come before their operands
- Integration with Jyatags
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("hedron" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data
- Adds cons, car and cdr functionality

Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [*text*]
- Repeats zero or more times: [*text*]…
- Repeats one or more times: <symbol>…
- Pipe separates alternatives: *opt1* **|** *opt2*
- Comments in *italics*

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon. Typing space after hyphen at end of identifier converts hyphen to underscore. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Jyatags

Jyatags is a simplified markup language used to replace HTML. Mock JSON files using Jyatags syntax have a .JYTJ extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Jyatags code can be embedded in the Jyalog echo statement. Jyatags syntax, where asterisk (*) means occurs zero or more times, is defined as follows:

Tags:
- [tag]
- [tag (fld val)*: body]
- [tag (fld val)*| body |tag]

Body:
- text
- [(fld val)*: text]*

Call: (Jyalog code)
- [expr: <expr>]
- [exec: <stmt>...]
- [jyg: <path>]

Jyalog Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space):

<source file>:
- do ([<imp>]... [<def glb>] [<def>]... [<class>]...)

<imp>:
    <import stmt> **;**

<import stmt>:
    import <module>...
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    (**:** <name><name>...)
    (as <name><name>)
    (as (**:** <name><name>...) <name>)

<mod list>:
    <id as>...

<id as>:
    <mod id>
    (as <mod id><name>)

<mod id>:
    <mod name>
    <class name>
    <func name>
    <var name>

<rel module>:
    (**:** [<num>] [<name>]...)
    <name>   *// ?*

<cls typ>:
    class
    iclass

<hedron>:
    hedron
    ihedron

<class>:
- <cls typ><name> [<base class>] [<does>] [<vars>] [<ivars>] do (<def>…) **;**
- abclass <name> [<base class>] [<does>] [<vars>] [<ivars>] do (<anydef>…) **;**
- <hedron><name> [<does>] [<const list>] do ([<abdef>]... [<defimp>]...) **;**
- enum <name><elist> **;**
- ienum <name><elist> **;**

<does>:
    (does <hedron name>...)

<hedron name>:
<base class>:
    <name>
    (**:** <name><name>…)

<const list>:
    (const <const pair>...)

<const pair>:
    (<name><const expr>)

<def glb>:
    gdefun [<vars>] [<ivars>] do <block> **;**

<def>:
- <defun> (<name> [<parms>]) [<vars>] [<gvars>] [<dec>] do <block> **;**

<defimp>:
- defimp (<name> [<parms>]) [<vars>] [<gvars>] [<dec>] do <block> **;**

<abdef>:
    abdefun (<name> [<parms>]) [<dec>] **;**

<defun>:
    defun
    idefun

<anydef>:
    <def>
    <abdef>

<vars>:
(var [<id>]...)

<ivars>:
(ivar [<id>]...)

<gvars>:
(gvar [<id>]...)

<parms>:
[<id>]... [<parm>]... [(* <id>)] [(** <id>)]

<parm>:
(<set op><id><const expr>)

<dec>:
(decor <dec expr>...)

<block>:
([<stmt-semi>]…)

<stmt-semi>:
<stmt> **;**

<jump stmt>:
<continue stmt>
<break stmt>
<return stmt>
return <expr>
<raise stmt>

<raise stmt>:
raise [<expr> [from <expr>]]

<stmt>:
<if stmt>
<while stmt>
<for stmt>
<switch stmt>
<try stmt>
<asst stmt>
<del stmt>
<jump stmt>
<call stmt>
<print stmt>
<bool stmt>

<call expr>:

- (<name> [<arg list>])
- (**:** <colon expr>… <name>)
- (**:** <colon expr>… (<method name> [<arg list>]))
- (**::** <colon expr>… <name> else <expr>)
- (**::** <colon expr>… (<method name> [<arg list>]) else <expr>)
- (call <expr> [<arg list>])

<call stmt>:

- <name> [<arg list>]
- **:** <colon expr>… (<method name> [<arg list>])
- call <expr> [<arg list>]

<colon expr>:
<name>
(<name> [<arg list>])

<arg list>:
[<expr>]... [(<set op><id><expr>)]...

<dec expr>:
<name>
(<name><id>...)
(**:** <name><id>...)
(**:** <name>... (<id>...))

<dot op>:
dot **| :**

<dotnull op>:
dotnull **| ::**

<del stmt>:
del <expr>

<set op>:
set | =

<asst stmt>:
<asst op><target expr><expr>
<set op> (tuple <target expr>...) <expr>
<inc op><name>

<asst op>:
set | addset | minusset | mpyset | divset |
idivset | modset |
shlset | shrset | shruset |
andbset | xorbset | orbset |
andset | xorset | orset |
= | += | -= | *= | /= |
//= | %= |
<<= | >>= | >>>= |
&= | ^= | '|=' |
&&= | ^^= | '||='

<target expr>:
<name>
(**:** <colon expr>… <name>)
(slice <arr><expr> [<expr>])
(slice <arr><expr> all)
(<crop><cons expr>)

<arr>: *// string or array/list*
<name>
<expr>

<if stmt>:
- if <expr> do <block> [elif <expr> do <block>]…
  [else do <block>]

<while stmt>:
    while <expr> do <block>
    while do <block> until <expr>

<for stmt>:
- for <name> [<idx var>] in <expr> do <block>
- for (<bool stmt>**;** <bool stmt>**;** < bool stmt>) do <block>

<try stmt>:
- try do <block> <except clause>… [else do <block>] [eotry do <block>]
- try do <block> eotry do <block>

<except clause>:
    except <name> [as <name>] do <block>

<bool stmt>:
    quest [<expr>]
    ? [<expr>]
    <asst stmt>

<switch stmt>:
    switch <expr><case body> [else do <block>]

<case body>:
    [case <id> do <block>]**...**
    [case <dec int> do <block>]...
    [case <str lit> do <block>]**...**
    [case <tuple expr> do <block>]**...**

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<paren stmt>:
    (<stmt>)

<qblock>:
    (quote [<paren stmt>]...)

<quest>:
    quest | ?

<inc op>:
    incint | decint | ++ | **--**

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>**...**)
    <cons expr>
    <tuple expr>
    <list expr>
    <dict expr>
    <venum expr>
    <string expr>
    <bytes expr>
    <target expr>
    <call expr>
    <cast>

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | **-**

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=

<shift op>:
    shl | shr | shru |
    << | >> | >>>

Note: some operators delimited with single quotes for clarity (quotes omitted in source code)

<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ | '||'

```
<multi op>:
      mpy | add | strdo | strcat |
      and | xor | andbitz | xorbitz |
      or | orbitz |
      * | + | % | + |
      && | ^^ | & | ^ |
      '||' | '|'

<const expr>:
      <literal>
      <keyword const>

<literal>:
      <num lit>
      <str lit>
      <bytes lit>

<cons expr>:
      ( cons <expr><expr> )
      ( <crop><expr> )
<tuple expr>:
      ( tuple [<expr>]… )
      ( <literal> [<expr>]… )
      ( )

<list expr>:
      ( jist [<expr>]… )

<dict expr>:
      ( dict [<pair>]… )

<pair>:
      // expr1 is a string
      ( : <expr1><expr2> )
      ( : <str lit><expr> )

<venum expr>:
      ( venum <enum name> [<elist>] )
      ( venum <enum name><idpair>... )

<elist>:
      <id>...
      <intpair>...
      <chpair>...

<intpair>
      // integer constant
      <int const>
      ( : <int const><int const> )

<chpair>
      // one-char. string
      <char lit>
      ( : <char lit><char lit> )

<idpair>
      <id>
      ( : <id><id> )
```

```
<cast>:
      ( cast <literal><expr> )
      ( cast <class name><expr> )

<print stmt>:      // built-in func
      print <expr>…
      println [<expr>]…
      echo <expr>…

<lambda>:
      ( lambda ( [<id>]... ) <expr> )
      ( lambda ( [<id>]... ) do <block> )
      ( lambdaq ( [<id>]... ) do <qblock> )
      // must pass qblock thru compile func
```

No white space allowed between tokens, for rest of Jyalog Grammar

```
<white space>:
      <white token>...

<white token>:
      <white char>
      <line-comment>
      <blk-comment>

<line-comment>:
      # [<char>]... <new-line>

<blk-comment>:
      { [<char>]... }

<white char>:
      <space> | <tab> | <new-line>

<name>:
•     [<underscore>]… <letter> [<alnum>]…
      [<hyphen-alnum>]… [<underscore>]…

<hyphen-alnum>:
      <hyphen><alnum>…

<alnum>:
      <letter>
      <digit>
```

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

```
<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L

<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…

<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    null
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<str lit>:
    " [<str item>]... "

<str item>:
    <str char>
    <escaped str char>
    <str newline>

<str char>:
    any source char. except "\", newline, or
    end quote

<str newline>:
    \ <newline> [<white space>] "

<escaped char>:
```

\\	*backslash*
\"	*double quote*
\}	*close brace*
\a	*bell*
\b	*backspace*
\f	*formfeed*
\n	*new line*
\r	*carriage return*
\t	*tab*
\v	*vertical tab*
\ooo	*octal value = ooo*
\xhh	*hex value = hh*

```
<escaped str char>:
    <escaped char>
```

\N{name}	*Unicode char. = name*
\uxxxx	*hex value (16-bit) = xxxx*

```
<crop>:
    c <crmid>... r

<crmid>:
    a | d
```

Not implemented: string prefix and bytes data type (rest of grammar)

```
<str lit>:
    [ $ <str prefix>] <quoted str>

<str prefix>:
    r | R

<quoted str>:
    " [<str item>]... "

<bytes lit>:
    $ <byte prefix><quoted bytes>

<byte prefix>:   // any case/order
    b | br

<quoted bytes>:
    " [<bytes item>]... "
```

```
<bytes item>:
    <bytes char>
    <escaped char>
    <str newline>

<bytes char>:
    any ASCII char. except "\", newline, or
    end quote
```