Mike Hahn – hahnbytes@gmail.com

Jovelyst

Jovelyst is an open source programming language used for developing dual user tutor/student Android apps. It is also used (potentially) for developing single and dual player Android games. The Jovelyst game/app editor runs on Android, Windows and Linux. The students pay the tutors by credit card or PayPal, and the tutors pay 10 percent to Jovelyst.com and 5 percent to the lesson-writers.

Both game players, as well as both the tutor and the student, must be at the same physical location since the respective Android devices communicate using Bluetooth. Some tutors choose to use the web-based whiteboard, interacting with students remotely. Jovelyst rhymes with novelist and is implemented in Java.

Game Engine

The Jovelyst game engine has the lowest priority of all Jovelyst-related projects. If and when it becomes operational, Jovelyst will have a freemium business model. Subscribers pay $10 per year and have access to web-based versions of the dual player games. They also have access to paid games and games which include in-game purchases. Yet another feature accessible by subscribers is the ability to interact remotely with volunteer tutors. Those tutors use a web-based whiteboard. Finally, subscribers can deploy games/apps to the Jovelyst app store.

Join Our Team

Jovelyst is a completely open source project. If Jovelyst makes any money, most of the profits will be allocated to funding a website for consumer/survivors called Psyvaspace.org. Therefore your labour, as a Jovelyst team member, will help support Psvyaspace members.

Jovelearn

Jovelearn.com is a website linking tutors with those needing instruction. Jovelyst is used to develop the dual user tutor/student Android apps and remote (web-based) whiteboard. Jovelearn takes a cut of 10 percent of fees charged by tutors, and the lesson-writers get 5 percent. Monospace/Rich-Text Modes (see below) are used to format text.

Subjects

Jovelearn subjects include math, science, literacy, and computers. Levels of difficulty range from grade school to university undergraduate level. If the student is under 18 and meets with the tutor in the student's home, a parent must be in the room or nearby.

Monospace Mode

In monospace mode, all body text rendered to the screens of Jovelearn end-users is in a mono-spaced, typewriter-style font. Every character takes up 2 square cells: an upper cell and a lower cell. Superscripts and subscripts are handled by employing a vertical offset of one square cell. Header text is also mono-spaced, and each character takes up 2 oversized square cells.

Additional Formatting

The grid of characters can be subdivided into panels, which can themselves be subdivided into more panels, and so on. Any panel can contain zero or more text boxes, which may overlap each other. Vertical grid lines each take up one square cell per row of square cells. Horizontal grid lines are displayed in the same pixel row as underscore characters. Any row of square cells containing a horizontal grid line which is 2 pixels wide is taller by exactly one pixel. The following bracket characters: ( ) [ ] { } can be oriented vertically or horizontally, taking up a single column or row of at least 2 square cells, respectively. Widgets such as check boxes, radio buttons, and combo box arrows take up 4 square cells (2 by 2). Images, animations, and diagrams are contained in canvas objects, which can appear anywhere panels can appear.

Rich-Text Mode

In rich-text mode, a given header or paragraph of body text can consist of a single variable-width font. Paragraphs have before/after spacing, left/right indent, and line spacing (single, double, 1.5, etc.). Panels have margins on all 4 sides. Beginner app-writers start off with monospace mode, and then advance to rich-text mode. In both rich-text and monospace (web-based) modes, text is rendered to the HTML5 canvas object. Some features like form fields and submit buttons use normal HTML.

Windows Client

The Windows Client enables the tutor to control a red pointer on the student's Windows screen (equipped with Bluetooth) using the tutor's Android device. Instructions appear in always-on-top windows on the Windows screen as well as on the Android device. This setup facilitates mastery of Windows applications such as MS Office.

Jovelyst Integration

Jovelyst on the Windows Client displays text/graphics in the always-on-top windows. Jovelyst on the Android device displays text/graphics and enables the tutor to interact with the lesson player and control the red pointer on the Windows screen.

Lesson Player

- Default background color: white
- Default color of content text: black
- Color of tutor text: gray
- Color of student text: blue
- Page Up/Down: scroll a page at a time
- Ctrl+Home/End: top/bottom
- F6: toggle user
- Click: move text cursor, display red mouse pointer on the other user's screen
- Arrow key: move text cursor
- Letter, digit, symbol, space: insert char.
- Insert: toggle insert/overwrite mode
- Underscore: toggle underline of current char. and move right
- Vertical slash: toggle vertical grid line at cursor and move down
- Ctrl+vertical slash: insert vertical slash char.
- Home/End: beginning/end of line
- Enter: append blank line, cursor backs up to line up with beginning of word to left of original cursor position on previous non-blank line
- Backspace (on blank line): cursor backs up to line up with beginning of previous word on previous non-blank line (unindent one word at a time)
- Delete: delete/clear highlighted block (depends on insert/overwrite mode)
- Shift+Arrow: highlight multi-line block if up/down and nothing already highlighted, else highlight rectangular block
- F1: show/hide menu (Ctrl+Letters)
- Ctrl+(Z, X, C, V, Y, B, I, U, J, O, N, F, R, H, G, K, Q, S, P, T, F1): undo, cut, copy, paste, redo, bold, italics, underline, justify, overwrite toggle, notes/content view toggle, find, search/replace, heading, graphic, hyperlink, special char., shade/borders, panel/text box, table, help screen
- Ctrl+J: justify, use arrow keys: Up = cancel, Down = center
- Ctrl+Up/Down Arrow: superscript/subscript
- Ctrl+Left/Right Arrow: move left/right word at a time
- Parenthesis/square bracket/brace bracket: insert oversize open/close bracket if:
  - a narrow block of text is highlighted (one square cell by at least 3 square cells)
- Else if a wider block of text is highlighted (at least 3 cells tall and 2 cells wide):
  - enclose highlighted block in oversize open/close brackets

- Quiz mode:
  - Similar to content view
  - All lines containing content text (usually black text) are read-only
  - User presses Enter to insert a blank line
  - User presses Enter to split line containing user text (blue) into 2 separate lines
- Play Content:
  - Navigation buttons in a row at top left (content view)
  - Start Icon: Access file commands:
    - Open, Close, New, Save, Save As, Print, Exit
  - [| <] : Beginning
  - [> |] : End
  - [<<] : Previous Slide
  - [>>] : Next Slide
  - [^] : Beginning of Current Slide
  - [>] : Play
  - [||] : Pause
  - Space : Single-Step
  - [<] : Reverse Single-Step
- Chat Window:
  - Enter: transmit chat entry
  - Up/Down Arrow: display previous/next chat entry of current user
  - Shift+Up/Down Arrow: display previous/next chat entry of different user
  - Left/Right Arrow: used to edit chat entry (highlight if used with Shift)
  - Ctrl+(Z, X, C, V, Y): undo, cut, copy, paste, redo
- Mac commands (optionally disabled on non-Macs):
  - Ctrl+O: same as Insert
  - Shift+Ctrl+E: same as Home
  - Ctrl+E: same as End
  - Delete: same as Backspace

Lystagger Syntax

Screen layout definition language. Doubles as HTML replacement for Jovelyst web development framework.

Tags:

- (tag)
- (tag; body)
- (tag (fld val)*; body)

Body:

- text
- (; text)*
- ((fld val)*; text)*

Notation:

- * repeats
- a; b; c = (a)(b)(c)

Android Keyboard

1	2	3	4	5	6	7	8	9	0
Q	W	E	R	T	Y	U	I	O	P
A	S	D	F	G	H	J	K	L	;
SH	Z	X	C	V	B	N	M	&	BK
SP	(	)	-	"	\	#	:	FN	CR

SH - Shift
SP - Space
BK - Backspace
CR - Enter
FN - Functions
& - Symbols

Psyvaspace Website

Psyvaspace (click here for old version), a sister project of Jovelyst, is an online IT-oriented community of consumer/survivors. The members learn various subjects and skills, including coding/web design as well as math, literacy, computer basics, and Microsoft Office. The tutors are volunteers and teach remotely or on-site. Students and tutors can use Windows or Android devices. The tutor's on-site Android device communicates with the student's computer using Bluetooth. The tutoring software consists of a whiteboard and a lesson player, and is written in an open source language of my own invention called Jovelyst. Jovelyst runs on all computers except iPhones, as it is implemented using Java. Volunteers will be recruited to write lessons in various subjects, which are played by the lesson player. Customized versions of the lesson player will be developed by the Jovelyst developers. West Neighbourhood House may or may not be involved in beta-testing the tutoring software.

Psyvaspace Partners

In addition to skills development, Psyvaspace includes forums (moderated by volunteers), chat rooms, blogs written by members, and a database of mental health resources. When the Jovelyst Compiler is fully operational, Progress Place and Portico (the 2 prospective partner organizations) will be contacted. Progress Place is a clubhouse for consumer/survivors. The primary role of Progress Place is to assist Psyvaspace in applying for funding from the Ministry of Health and Long-Term Care. The funding will pay for 2 full-time employees: the Executive Director and the Volunteer Coordinator, plus $2000 per year for a dedicated server and 2 domain name registrations. Mike is the CEO of Jovelyst.com and the Webmaster of Psyvaspace.

Two additional roles of Progress Place are to beta test the Psyvaspace website, and also the members of the Clerical Unit can be enlisted to perform data entry, keeping the database of mental health resources up to date. Progress Place can choose to play an active role in the operations of Psyvaspace, or to simply take on the role of helping to secure funding.

Portico is a network of Canadian addiction and mental health websites, and is affiliated with CAMH. Hopefully they will permit Psyvaspace to be included in that network. If Progress Place chooses not to be involved in the Psyvaspace project, then hopefully Portico or some other branch of CAMH can assist in applying for funding from the Ministry of Health and Long-Term Care. In case Jovelyst is successful in the marketplace, then the funding from the Ministry is only temporary, since most of the profits earned by the Jovelyst website can be used to fund Psyvaspace.

Business Plan (Outline)

1. Core Development:
    1. Implement Jovelyst Compiler
    2. Recruit GitHub Java Users
    3. Port Jovelyst to Android
    4. Integrate Lystagger with Jovelyst
    5. Port Lystagger to Android
    6. Develop Linux Client (always-on-top text)
    7. Port Linux Client to Windows
    8. Develop Android Tutor Client
    9. Add Bluetooth support
    10. Approach CAMH/Progress Place
2. Advanced Development:
    1. Learn WebSocket coding
    2. Implement Monospace Mode
    3. Implement Rich-Text Mode
    4. Make both modes multi-user
    5. Implement Android Whiteboard
    6. Make whiteboard multi-user
    7. Implement Converter: Lystagger-to-HTML
    8. Implement Converter: Jovelyst-to-JavaScript
3. IDE Development:
    1. Implement 2 Eclipse Plugins
    2. Use Plugin as Jovelyst IDE
    3. Use other Plugin for Lystagger IDE
    4. Implement WYSIWYG (Lystagger)
    5. Port IDEs from Linux to Windows
    6. Port IDEs to Android
4. Game Engine (optional):
    1. Implement Game Engine
    2. Make game engine multiplayer
5. Website:
    1. Design website
    2. Implement Tutor-Search
    3. Implement Session-Scheduling
    4. Handle Credit Cards/PayPal
    5. Handle Sales Tax
    6. Implement Game/Lesson Center
    7. Expand Jovelyst Library
    8. Write Lesson Apps
    9. Design Psyvaspace Website
6. Funding:
    1. Write Formal Business Plan
    2. Recruit CEO
    3. Mike is the CTO (T = Technology)
    4. Recruit Psyvaspace Board Members
    5. Apply for Psyvaspace Funding
7. Launch:
    1. Purchase Google AdWords ads
    2. Jovelyst goes live
    3. Psyvaspace goes live
    4. Consider using Web Design Pros
    5. Use SEO to increase traffic
    6. Hire Volunteers

7. Recruit Tutors
8. Recruit Lesson-Writers

8. Expansion:
    1. Expand to other Canadian cities
    2. Expand Psyvaspace to other Clubhouses
    3. Make website scalable
    4. Expand to American cities
    5. Expand to UK
    6. Make website bilingual (Quebec)
    7. Make website multilingual
    8. Expand to Europe/Asia, etc.
9. Business Matures:
    1. Share profits with GitHub Java Users
    2. Rent Office Space
    3. Hire Paid Employees

History

I am Mike, the founder, and got the idea for Jovelyst on Canada's 150th birthday. I was at my brother's cottage and used the memo feature of my phone to make a to do list. (Prior to April 2017 I had never owned a cell phone.) Realizing that a phone can actually do more than just call/text, I then got the idea of using the programming language I had previously designed and begun implementing, as it was ideal for a small screen. I then resurrected my old games program idea (make your own games), just for Android.

About Me

I am Mike Hahn, the founder of Jovelyst.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. I'm now a volunteer math/computer tutor at West Neighbourhood House. My hobbies are reading quora.com questions/answers and the news at cbc.ca. About twice a year I get together with my sister Cathy who lives in Victoria. She comes here or I go out there usually in the summer. Prior to starting Jovelyst I used to lie on the couch a lot, not being very active. Now I'm busy most of the time. I visit my brother Dave once a month or so and I also visit my friends Main and Steph once or twice a month.

Contact Info

Mike Hahn
Founder, Jovelyst.com
515-2495 Dundas St. West
Toronto, ON M6P 1X4

Country: Canada
Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: www.hahnbytes.com

Language Grammar

Jovelyst is a 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). Jovelyst code is often accompanied by Lystagger screen definition files. Lystagger is similar to HTML, except open tags begin with an open parenthesis and a keyword, and the closing tag is simply a close parenthesis. Any text enclosed in a tag is preceded by a semicolon. File extensions include .LYST (source code), .LSTA (assembler code), .LSTC (compiled code), and .LSTG (Lystagger).

Keyboard Aid

This optional feature (Windows/Linux) 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.

Special Characters

- `()` grouping
- `-` used in identifiers
- `;` end of stmt.
- `:` dot operator
- `"` string delimiter
- `\` escape char.
- `#` comment

More Characters

- `_` used in identifiers
- `$` string prefix char.
- `*` public switch
- `{}` block comment

Differences from Python

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

Grammar Notation

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

Compiler and Assembler

The Jovelyst Compiler translates source code into compiled code, and optionally into assembler code. Assembler code is an intermediate language which is much simpler than source code, although both source code and assembler code are in the form of text files. During Jovelyst development, the developer uses the Jovelyst Assembler, which converts assembler code (hand-written by the developer) into compiled code. This is a necessary step enabling the JovelYst Runtime Environment (JYRE) to be tested prior to the development of the Jovelyst Compiler.

Jovelyst Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space, also any semicolon before a close parenthesis may be omitted):

<source file>:
- [<use>] [* <vars>] [<vars>] [do <block>] [<def>]… [<class>]… [do <block>]

<use>:
    use (<import-semi>...)

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

<import stmt>:
    import <module>
    import (<module>…)
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    (<name> as <name>)
    (**:** <name>… [as <name>])

<mod list>:
    <id as>
    (<id as>…)

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

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

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

<class>:
- ([*] <cls typ><name> [<base class>] [<does>] [* <vars>] [<vars>] <def>…)
- ([*] scool <name> [<does>] [<const list>] [<def hdr>]...)
- ([*] enum <name><elist>)

<cls typ>:
    class
    abclass

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

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

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

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

<def hdr>:
    (<defun><name> ([<parms>]) [<dec>])

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

<defun>:
    def
    abdef

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

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

<parm>:
    <id>
    (tuple <id>...)
    (<set op><id><expr>)
    (<set op> (tuple <id>...) <expr>)

<dec>:
    decor <call expr>...

<block>:
    ([<stmt-semi>]…)

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

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

<pjump stmt>:
    return <expr>
    ** <raise stmt>

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

<stmt>:
    <open stmt>
    <closed stmt>

<open stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    ** <try stmt>
    <pjump stmt>
    <pcall stmt>
    <asst stmt>
    <del stmt>

<closed stmt>:
    <jump stmt>
    <call stmt>
    <print stmt>
    <lstg tag>

<lstg tag>:
    (lstg **;** ...)

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

<call stmt>:
    (<name> [<arg list>])

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

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

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

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

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

<set op>:
    set | =

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

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

<obj expr>:
    <name>
    <call stmt>

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

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

<for stmt>:
    for <name> in <expr> do <block>

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

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

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<del stmt>:
    del <expr>

<paren stmt>:
    (<open stmt>)
    <closed stmt>

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

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>...)
    <renum expr>
    <tuple expr>
    <lyst expr>
    <dict expr>
    <bitarray expr>
    <string expr>
    <bytezero expr>
    <bytes expr>
    <target expr>
    <obj expr>
    <cast>

<quest>:
    quest | ?

<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>

<tuple expr>:
    (tuple <expr>…)

<lyst expr>:
    (lyst [<expr>]…)

<dict expr>:
    (dict [<pair>]…)

<bitarray expr>:
    (bitarray <enum name> [<elist>])
    (bitarray <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>
    <idt>
    (<id><id>)

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

<renum expr>
    (renumize <expr><ren id>...)
    (renumize <expr><ren int>...)
    (renumize <expr><ren ch>...)

<ren id>:
    (0 <id>)
    (1 <id>)
    (1 <id><id>)

<ren int>:
<ren ch>:
    // expr is <dec int> | <char lit>
    (0 <expr>)
    (0 <expr><expr>)
    (1 <expr>)
    (1 <expr><expr>)

<cast>:
    (cast <type><expr>)

<print stmt>:   *// built-in func*
    (print [<expr>]…)
    (echo [<expr>]…)

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

No white space allowed between tokens, for rest of Jovelyst 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>:
    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>:
    none
    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

<string lit>:
    [$ <str prefix>] <short long>

<str prefix>:
    r | u | R | U

<short long>:
    " [<short item>]... "
    " " " [<long item>]... " " "

<short item>:
    <short char>
    <escaped str char>

<long item>:
    <long char>
    <escaped str char>

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

<long char>:
    any source char. except "\"

<bytes lit>:
    $ <byte prefix><shortb longb>

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

<shortb longb>:
    " [<shortb item>]... "
    " " " [<longb item>]... " " "

<shortb item>:
    <shortb char>
    <escaped char>

<longb item>:
    <longb char>
    <escaped char>

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

<longb char>:
    any ASCII char. except "\"

<escaped char>:
    \newline
        ignore "\", newline chars.
    \\ *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*