Mike Hahn - hahnbytes@gmail.com

CoopleSoft

CoopleSoft is both a game engine and a tool used for teaching STEM subjects such as math and computers. The game engine is called CoopleGames and the teaching tool is called CoopleTeach. Both are implemented using Java and make use of a scripting language called JCoople. The games are partially written in Java and also written in JCoople. CoopleTeach includes a whiteboard, and the math-specific functionality of the whiteboard (as well as other subjects) is written in JCoople. CoopleSoft also supports general app development, not just games and teaching tools.

Business Model

Premium and trial mode features can only be accessed by users who convert. The premium mode of games enables 2-player mode in full color. Trial mode of games enables 2-player mode with a grayscale display. Trial mode of apps limits images to grayscale, and all other pixel colors to white or light gray.

CoopleSoft members fall into 4 categories: bronze, silver, gold, and platinum. All members can use CoopleTeach without restrictions. Silver, gold and platinum users pay $10, $20 and $30, respectively. Silver users can access games in trial mode. Gold users can access games in premium mode, and apps in trial mode. Platinum users can access games/apps in premium mode.

CoopleGames

CoopleGames is used to develop mobile/desktop games using 2D and 2.5D animation. It is also used to develop smartphone apps ("CoopleApps"). Both the games and the apps can be partially written in Java as well as JCoople. Premium mode games enable 2-player mode. Bronze members can only use 2-player mode if their opponent is a silver, gold or platinum member. Bronze and silver members can only use 2-player mode in grayscale, or color if their opponent is a gold or platinum member.

CoopleApps

CoopleApps are smartphone apps developed using CoopleGames. Premium mode apps include more advanced features than basic mode apps. Silver and gold members can only use premium mode features in monochrome mode: the display generated by JCoople code is monochrome (white and light gray) with grayscale images. Platinum members can access premium mode in full color. Developers of freemium apps are encouraged to make use of the built-in gold and platinum switches (taking into account user classes), but this is not mandatory.

CoopleTeach

CoopleTeach is a multi-user CoopleApp used for teaching STEM subjects such as math and computers. The student interacts with a whiteboard and the teacher displays the whiteboard on a smartphone. The math functionality of the whiteboard is written in JCoople. Teachers pay to CoopleSoft royalties of $0.50/month/student.

JCoople

JCoople is an open source scripting language which is implemented using Java. Its sister language is Joopletags, which is a text markup language. The games and smartphone apps are partially written in JCoople. The math functionality of the whiteboard is also written in JCoople.

User Upgrades

New users are always bronze members. Users who wish to upgrade must install the CoopleSoft Upgrade app and pay $10 for silver, $20 for gold, $30 for platinum. Upgrading from silver to gold, and from gold to platinum costs $10. Upgrading from silver to platinum costs $20. Every time a user upgrades, the cost of the upgrade is divided by quantity one plus the number of CoopleSoft games/apps installed on the user's phone. This amount, call it U, is distributed on a quarterly basis to each of the developers of the games/apps. Developers of more than one CoopleSoft game/app on the user's phone receive a multiple of the amount U.

Since unlike Android, iPhone users lack shared storage on their phones, additional work is required to handle iPhone user upgrades. The user must tap the CoopleSoft logo, which looks like a rounded rectangle containing an uppercase CS, with the lower right-hand corner of the C flowing into the lower left-hand corner of the S. Then the user enters a 6-digit token in a dialog box. This process must be repeated for each installed CoopleSoft game/app, in order for the upgrade to be recognized by that game/app. The 6-digit token is displayed in the CoopleSoft Upgrade app.

When upgrading to silver or gold, the user must confirm the upgrade within 30 days, otherwise the user is automatically downgraded to bronze or silver, respectively. The user's credit card is charged at the time of upgrade confirmation. When upgrading to gold, the user is given the choice of immediately confirming the upgrade, or letting it expire after 30 days in case no confirmation takes place.

CoopleGames Intro

CoopleGames is a tool used to make 2-player mobile games, using 2D and 2.5D animation. Android games are written in Java, and iOS games are written in Swift. Games also run on desktop operating systems: Windows, Linux and Mac. When running games in 2-player mode, at least one player must be using a smartphone. When both players are using smartphones, dual-OS games enable one player to be Android and the other player to be iOS. End-users can customize games using a scripting language called JCoople. Game connectivity is accomplished using Bluetooth. CoopleGames also supports development of general Java/Swift/JCoople applications, not just games.

Business Model

Premium mode games enable 2-player mode. Bronze members can only use 2-player mode if their opponent is a silver, gold or platinum member. Bronze and silver members can only use 2-player mode in grayscale, or color if their opponent is a gold or platinum member. Silver, gold and platinum users pay $10, $20 and $30, respectively.

Dimetric Projection

Games are 2.5D using dimetric projection. All planes are parallel or at 90 degree angles with each other, the vantage point of the user is at a 45 degree angle, and all horizontal/vertical lines in the horizontal plane are rendered such that the slope of the line is +/- 0.5 (vertical lines in vertical planes are always vertical). Only horizontal, vertical, and diagonal lines at 45 degree angles are allowed. Since all planes are angled instead of directly facing the user due to the dimetric projection, diagonal lines are not rendered using a slope of 0.5, but have some other slope. Curves are limited to circular arcs in multiples of 45 degrees. Due to the dimetric projection they are rendered as elliptical arcs. Text is monospaced and appears skewed. Labels are allowed which contain a single line of normal text, bounded by a normal rectangle. Labels are always displayed in front of/on top of the dimetric projection.

Animation

Objects can move in 8 directions in 2D mode (90 degree angles and 45 degree angles) and 6 directions in 2.5D mode (up/down, left/right, forward/backward). Objects may include discs and balls. Support for collision detection functionality is provided. The parent object of an animated 2.5D object is assumed to be located on the ground or building directly beneath it. Objects can also dynamically change shape, incrementally or all at once.

Collision Detection

Animation in 2D mode is covered here. All game objects (moving and stationary) have a bounding rectangle. Non-rectangular game objects have a transparent bitmap. A non-rectangular animation region has one or more transparent border bitmaps, which are game objects, usually stationary. Collision detection OR's each pixel in one transparent bitmap with underlying pixel in the other transparent bitmap, detecting a collision upon first OR operation yielding a zero result. Additional logic is required to prevent overshooting a border of a non-rectangular animation region.

CoopleTeach Intro

CoopleTeach is a tool used for teaching STEM subjects such as math and computers, and is implemented in Java. 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. The CoopleBoard, a specialized whiteboard, is used to teach various STEM subjects. The teacher's employer pays to CoopleSoft royalties of $0.50/month/student (waived for registered charities).

Teaching Locally

Members display the curriculum on a Windows computer running a whiteboard called the CoopleBoard. Teachers display a window of the member's screen on a smartphone held in landscape orientation. 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 CoopleBoard, or an always on top chat window.

CoopleBoard

The CoopleBoard supports math being taught, using text in monospaced mode. Most of its functionality is written in Java, but extensions used to teach STEM subjects are written in JCoople. A Java API enables the development of CoopleBoard extensions using Java. 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 JCoople programming language. Each step in the math problem being solved manipulates this JCoople 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 CoopleBoard 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).

About Us

I am Mike Hahn, the founder of CoopleSoft.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 was greatly minimized after I ramped up development of my previous Aljegrid project in early March 2021.

Implementation Steps

1. Develop foundation of JCoople code execution - ***almost done!***
2. Develop rest of JCoople code execution
3. Release JCoople as console-based compiler on GitHub
4. Implement GUI: monospaced mode
5. Release JCoople/GUI on GitHub
6. Write Joopletags design specs
7. Develop Joopletags
8. Integrate JCoople with Joopletags
9. JCoople/Joopletags: JCoople RUn-time Environment (JCRUE)
10. JCRUE for Linux is open source
11. Integrate JCRUE with Android: JCoople Library
12. Develop Java 2D game engine
13. Develop Java 2.5D game engine
14. Develop Java-JCoople interface
15. Implement user classes
16. Implement Swift concurrently:
    1. Hire Swift Programmer (SP)
    2. SP paid 60% of profits or 20K, whichever is greater
    3. Profit: revenue minus expenses
    4. Expenses include wages of Java programmers (not founder)
    5. Convert JCRUE from Java to Swift
    6. Port Android system to iOS
    7. SP receives 10K at midpoint of project
    8. SP receives another 10K (minimum) upon successful termination
    9. Note: initial offer of 8 to 10K, go as high as 20K
17. Implement CoopleTeach
18. Implement CoopleBoard (CB) using Java
19. CB: JCoople math functionality
20. CB: JCoople algebraic expression handler
21. Develop monetizing functionality
22. Perform beta testing
23. Launch website
24. Purchase Google AdWords advertising
25. Develop JCoople code editor
26. Expand code editor to JCoople SDK
27. Implement Keyboard Aid (bells and whistles of editor)
28. Develop WYSIWYG Joopletags screen editor

Annual Revenue

Category	Rate	Qty	Amount
Silver:	$5/user	1K	$5,000
Gold:	$10/user	4K	$40,000
Platinum:	$15/user	1K	$15,000
Teach:	$6/student	25K	$150,000
Total:			$210,000

JCoople

JCoople (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). JCoople source files have a .JOOP extension. Joopletags files (the sister language of JCoople, a text markup language) have a .JPTG extension. JCOOPLE (Java-Centric Object-Oriented Programming Language/Environment) 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 Joopletags
- 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.

Joopletags

Joopletags is a simplified markup language used to replace HTML. Mock JSON files using Joopletags syntax have a .JPJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Joopletags code can be embedded in the JCoople echo statement. Joopletags 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: (JCoople code)
- [expr: <expr>]
- [exec: <stmt>...]
- [joop: <path>]

JCoople 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 JCoople 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
```