Mike Hahn - hahnbytes@gmail.com

JCoopiter

JCoopiter is a tool used to make Java and Android apps extensible. The app developers download the JCoople API, which is a JAR file enabling Android apps to talk to the app extensions. Those extensions are written in a language called JCoople. The sister language of JCoople is Joopletags, which is a text markup language. JCoople can also be used to make pure JCoople apps (j-capps) for Android which lack Java code.

Business Model

Subscribers pay $5/year to JCoopiter and at least $5/year to each app developer, and guest users pay no fees. Subscribers who are end-users can make unrestricted use of app extensions and premium features of j-capps. Subscribers who are developers pay $10/year to JCoopiter and can develop j-capps and app extensions. Guest users can also develop j-capps and app extensions, but they cannot be uploaded. JCoople is free for Windows and Linux.

Monochrome Mode

Only end-users of app extensions who are guests experience monochrome mode. All text rendered by JCoople is pale gray with a white background, and bitmaps are pre-rendered in pale gray and white. To temporarily suppress monochrome mode, the user must press down about a second, which restores colored pixels (but only while the user continues to press down). The action of pressing down only has an effect on monochrome mode one quarter of the time, at other times monochrome mode remains in effect.

Sister Projects

CooplePix is an image sharing j-capp, and CoopleTeach is a tool used for teaching math and computers. CoopleTeach is implemented in Java, and includes a whiteboard used for teaching math and other STEM subjects. The subject-specific logic of the whiteboard is written in JCoople. CoopleTeach beta testing will hopefully be performed by West Neighbourhood House and Progress Place, for math and computers, respectively.

JCoople Development

Developers of the Android apps download the JCoople API in the form of a JAR file. The developers of the JCoople-based functionality of each app download the JCoople Software Development Kit (SDK). The app developers decide which modules, classes, methods, and fields are accessible to JCoople code. JCoople code is uploaded to a folder located under jcoopiter.com/devs/xyz/, in the case of an app developer named XYZ. End-users download the JCoople code previously uploaded. Modifying a field which is not read-only triggers a public JCoople guard function, which can veto the update if required. Joopletags panel collections have a mutable switch which allows sub-panels to be added/deleted.

Joopletags Database

JCoople code can access master MySQL databases having one or more of the following fields: id, parentid, firstname, lastname, startingdate, endingdate. The parentid field is a foreign key. The detail database has the following fields: id, tableid, recordid, lineno, text. The text field is 100 chars. long and consists of Joopletags code, usually organized in a format which emulates a different language called JSON. Each record in the master MySQL databases is associated with zero or more records in the detail database.

CooplePix

CooplePix is a sample freemium JCoople app: a tool used for organizing image collections. Subscribers can share their images. Each image has an optional name consisting of one or more name parts, and zero or more features. Each feature has one or more mutually exclusive categories.

Main Menu

The main menu consists of 3 columns of buttons. The left column includes all the different commands: Grid, Search, Feature, Clear, Edit, Settings, Quit. The middle column includes all the different categories of the primary feature, plus All. The right column is for the secondary feature. An example of a primary feature is hair color. An example of a secondary feature is clothing color. The maximum number of features is 255, and the maximum number of categories per feature is 255.

Grid View

Images are displayed in rows and columns. Clicking on an image takes you to image view. Both grid view and image view include at the bottom a navigation row of 4 low-height buttons: yellow, red, green, blue. Yellow is Left, blue is Right, red is Up, and green is Mode. Left and Right display previous and next screen. Up takes you to main menu. Mode toggles between all images and all images having a given name. Images appear in random order by default. Clicking on Grid in main menu takes you to grid view.

Image View

Image size of subject is maximized. Left and Right display previous and next image. Up takes you to grid view. Mode changes from all images to all images having a given name, taking you to grid view. Mode is grayed out if already in single-name mode. Clicking on an image "likes" it. Most-liked images are more likely to appear near the beginning of random image lists, although that effect decays over time.

Search

Displays a keyboard: letters, space and asterisk, with or without digits, no shift key. Typing letters narrows down the list of 1st name parts, displaying matches above the keyboard. Clicking on a match or typing space displays matching 2nd name parts, and typing letters then narrows down the list of 2nd name parts (along with the 1st name part). Typing asterisk (*) matches any name part. Clicking on a partial name displays matching complete names, and user then clicks on a matching name or continues typing. Clicking on a complete name takes you to grid view. Each image has zero or more name parts.

Feature

Displays feature list in middle column. Clicking again toggles between displaying feature list in middle and right columns. Clicking on a feature list displays categories in selected feature.

Clear

Set all feature settings to All Categories.

Edit

Toggle edit mode, enabling adding/deleting images, features and categories, or reordering images.

Settings

Show/hide digits on keyboard. Change row and column counts. Toggle random/fixed order for lists of images in a given category or having a given name, for ordered lists. More advanced features include adding/deleting images, features and categories.

Image Sharing

Users can choose to share only images having a given name, and/or belonging to one or more categories. Users can also make image metadata public (names and categories), and browsable by other users. The user downloads the image database belonging to another user from the server. Instead of downloading all the images, only the file names are downloaded, each consisting of 16 hex digits. Users cannot browse images of guests, and guests cannot share email links to more than 16 images at a time.

CoopleTeach

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 an annual subscription fee of $5/student for the first 20 students in a class, and $1/student for the 21st and all subsequent students in a given class. Tutors pay $20/year to be included in the tutor directory.

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. 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).

Implementation Steps

1. Develop foundation of JCoople code execution - ***almost done!***
2. Approach Progress Place and West Neighbourhood House
3. Develop rest of JCoople code execution
4. Release JCoople as console-based compiler on GitHub
5. Implement GUI: monospaced mode
6. Release JCoople/GUI on GitHub
7. Write Joopletags design specs
8. Develop Joopletags
9. Integrate JCoople with Joopletags
10. JCoople/Joopletags: JCoople RUn-time Environment (JCRUE)
11. JCRUE for Linux is open source
12. Integrate JCRUE with Android: JCoople Library
13. Develop CooplePix for Linux
14. Develop JCoople code editor
15. Expand code editor to JCoople SDK
16. Convert CooplePix to JCoople
17. Implement CoopleTeach
18. Implement CoopleBoard using Java
19. Implement math functionality of CoopleBoard using JCoople
20. Develop monetizing functionality
21. Perform beta testing using PP and WNH
22. Launch website
23. Purchase Google AdWords advertising
24. Convert JCRUE from Java to Swift
25. Port Android system to iOS
26. Implement Keyboard Aid (bells and whistles of editor)
27. Develop WYSIWYG Joopletags screen editor

Annual Revenue

Category	Rate	Qty	Amount
JCoopiter:	$5/end-user	3K	$15,000
	$10/developer	1K	$10,000
CooplePix:	$5/user	1K	$5,000
CoopleTeach:	$5/student	10K	$50,000
Tutors:	$20/tutor	1K	$20,000
Subtotal:		16K	$100,000
App Store:	15%	$20K	- 3,000
Total:			$97,000

Free	Annual Fee			Available Features
	$5	$10	$20	
X	–	–	–	Monochrome mode: app extensions
–	X	X	X	No monochrome mode
–	X	X	X	Use premium features of j-capps
–	X	X	X	Unrestricted app extensions
X	X	X	X	Write j-capps, app extensions
–	–	X	X	Upload j-capps, app extensions
–	–	X	X	MySQL data hosting
CooplePix				
X	–	–	–	No public images
–	X	X	X	Public images: browsable
X	X	X	X	Share email links: under 16 images
–	X	X	X	Share email links: over 16 images
CoopleTeach Tutors				
–	–	–	X	Included in directory
–	–	–	X	Students can pay online

About Us

I am Mike Hahn, the founder of JCoopiter.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.

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
```