Mike Hahn - hahnbytes@gmail.com

Gramrite

Gramrite is a smartphone app and website showcasing software written in Java, Psybergram, and Psybertags. Psybergram is a new programming language and Psybertags is a new text markup language. Gramrite has a freemium business model: inner users pay $10/year, and outer users pay no fees. Outer users can browse the Gramrite.com website and the server of the Gramrite smartphone app (mini app store), but only inner users can upload code (paid apps), images, and other content.

Paid Apps

Psybergram smartphone apps which do not exceed the monthly bandwidth quota are hosted for free. Web hosting fees for apps which exceed the quota start at $5 per month. Apps which charge optional fees to users are only for inner users, for those users who elect to pay those fees. Paid apps without a free option are only for inner users. Google and Apple generally charge a commission of 30 percent for all payments between app authors and end-users, which drops to 15 percent after one year.

Self Hosting

Authors of apps have the option of hosting their own apps. They download the master app, which executes Psybergram code. In the case of Android, the master app is an APK file. App authors who opt for self hosting pay royalties of $5 per month per app to Gramrite, or $10 per month for paid apps. Once or twice a year, Gramrite connects to the app author's server and disables it in case the renewal fee remains unpaid.

Miscellaneous

Any code a given outer user writes only runs on their own computer and/or smartphone. Outer users cannot edit multi-line text edit boxes on the server having more than 255 characters. Psybergram programs run on Windows, Mac, Linux, Android and iOS devices. Clicking on PDF lets you print this website in its entirety. Boardalog is used for teaching math and computers.

Groupware

Gramrite offers a form of groupware for organizations. Organizations which serve consumer/survivors use the groupware for free (see Psyberhood). The Psyberhood material includes a detailed description of the groupware. Other organizations pay annual subscription fees of $2 per user for the first 50 users, and $0.50 per user for the 51st user and all subsequent users.

In Brief

Gramrite includes 3 components: an app store for smartphone apps which are all written in Psybergram, an online community for consumer/survivors called Psyberhood, and the Boardalog tool used by tutors to teach math. Psybergram is a new programming language I'm inventing, along with a text markup language called Psybertags. The app store has a freemium business model whereby users can pay a subscription fee to access more advanced features. Psyberhood is free for consumer/survivors, but other online communities pay a fee for each member. Math tutors pay a fee for each student. The Ministry of Health provides partial funding to support Psyberhood.

Psyberhood

Psyberhood.org is a non-profit organization which leverages the power of Gramrite in service of the international community of consumer/survivors. (A consumer/survivor is a person with mental health issues.) Partnerships with existing clubhouses and other organizations which serve consumer/survivors (members) will be nurtured, in order to help their members make online connections with other members, friends/relatives of members, and mental health professionals. Volunteer moderators will be recruited and supervised by Psyberhood, funded by the Ministry of Health. Social media applications (forums, chat rooms, Twitter-like apps) will be developed to serve the members. Each participating organization can focus on social media apps meant just for their members. Various diagnoses correspond with other social media subgroups. Gramrite users can browse the directory of organizations and make donations.

Digital Activities

Activities for the members include social media apps, CBT resources, multiplayer board and action games, game tournaments, online courses (education), schedules of bricks and mortar events, diagnosis-specific resources, workshops in: web design, coding (Java, Psybergram, and Psybertags), game design, writing/poetry, digital art, and job search; posting of recipes, guidelines for navigating the mental health system for members and relatives, and member web profiles. Members can get creative by customizing their personal web profiles using Psybertags, and even Psybergram (defaults to hide member customizations). The A4i schizophrenia smartphone app developed by CAMH can be a jumping off point for community members designing their own therapeutic apps. All activities except coding are built using both Psybergram/Psybertags and web-based HTML/JavaScript software tools.

Games

A game engine which supports multiplayer action games (and also board games) is written in Java. The games themselves are written in Psybergram. Graphics supported include 2D and 2.5D, but not 3D. A dimetric projection is used to support 2.5D graphics.

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.

Boardalog

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. The Whytergram, a specialized whiteboard, is used to teach math. 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 or smartphone, running a whiteboard called the Whytergram. 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 Whytergram, or an always on top chat window.

Whytergram

The Whytergram supports math being taught, using text in monospaced mode. Most of its functionality is written in Psybergram. 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 Psybergram programming language. Each step in the math problem being solved manipulates this Psybergram 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 Whytergram 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 Gramrite.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.

Implementation Steps

1. Develop foundation of Psybergram code execution - ***almost done!***
2. Approach Progress Place, clubhouse for consumer/survivors
3. Develop rest of Psybergram code execution
4. Release PRUNE 1.0 for Linux: Psybergram RUN-time Environment
5. PRUNE for Linux/Android is open source
6. Use Specialisterne to hire Android programmer on spectrum:
    - they find tech jobs for those on autism spectrum
7. Develop Linux and Android branches in parallel
8. Linux:
    1. Console-based PRUNE
    2. Monospaced/GUI version of PRUNE
    3. Write Psybertags design specs
    4. Develop Psybertags
    5. Integrate Psybergram with Psybertags
    6. Full GUI version of PRUNE
    7. Psybergram code editor
    8. WYSIWYG Psybertags screen editor
    9. Design website
    10. Perform beta testing
    11. Monetizing functionality
    12. Psyberhood
9. Android:
    1. Contract programmer, 6 months
    2. Console-based PRUNE (after 8.1 is working)
    3. Monospaced/GUI version of PRUNE
    4. Full GUI version of PRUNE
    5. Psybergram code editor
    6. WYSIWYG Psybertags screen editor
10. Make pitch to DMZ tech incubator (after 9.2)
11. Search for angel investor
12. If necessary, apply for funding from Ministry of Health:
    1. Assume revenue of $300,000 and expenses of $500,000
    2. Then annual funding of $200,000 is required
13. Psybergram
    1. Social media apps
    2. Hire moderators
    3. Handle donations
    4. Perform beta testing
    5. Launch website
    6. Purchase Google AdWords advertising
14. IOS:
    1. Convert PRUNE to Swift/iOS
    2. Hire Swift programmer if funds available
    3. Psybergram code editor
    4. WYSIWYG Psybertags screen editor
15. Implement Game Engine for Linux
16. Implement Boardalog for Linux
17. Port Game Engine to Android and iOS
18. Port Boardalog to Android and iOS
19. Exit strategy: all apps self hosted, no more royalty payments

Linux

1. Console-based PRUNE *
2. Monospaced/GUI version of PRUNE *
3. Psybertags design specs
4. Psybergram/Psybertags Integration
5. Full GUI version of PRUNE *
6. Psybergram code editor *
7. WYSIWYG Psybertags screen editor *
8. Game Engine *
9. Whytergram w/ Bluetooth *
10. Whytergram for Math
11. Whytergram Expression Handling
12. Chat Window *
13. Desktop Sharing Software

Android

1. Psybertags design specs
2. Psybergram/Psybertags Integration
3. Master App
4. Royalty Enforcement
5. Tutor Client w/ Bluetooth

iOS

1. Master App
2. Royalty Enforcement
3. Tutor Client w/ Bluetooth

Psybergram

1. Social media apps +
2. Hire moderators +
3. Browse App Collection +
4. Game Tournaments +
5. Event Scheduling +
6. Mental Health Resources +
7. Coding Workshops
8. Game Design Workshops
9. Writing/Painting Workshops +
10. Job Search +
11. Social Media Profiles
12. Therapeutic Apps
13. Whytergram

Web

1. Gramrite.com website
2. Monetizing functionality
3. Handle Donations
4. Web Design Workshops
5. Tutor Directory
6. Boardalog.com website

+ Repeated in Web list

* Repeated in both Android and iOS lists

Annual Revenue

Admittedly, these numbers are on the optimistic side somewhat, since it's not a given that Gramrite, Psyberhood or Boardalog will manage to sign up thousands of users. But at least it shows that profitability can be possible, potentially. Also, even if only one project ends up being profitable, the other 2 projects can be abandoned and financial sustainability can be achieved. If Psyberhood winds up being the only surviving project, it doesn't have to be profitable since funding from the Ministry of Health can bridge the gap between revenue and expenses.

Category	Rate	Qty	Amount
Gramrite:	$10/user	16K	$160,000
Self Hosting:	$80/app	125	10,000
Psyberhood:	$2/user	25K	50,000
Boardalog:	$5/student	20K	100,000
Tutors:	$20/tutor	2K	40,000
Subtotal:		63K	$360,000
App Store:	15%		- 54,000
Total:			$306,000

Psybergram

Psybergram (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). Psybergram source files have a .PSYG extension. Psybertags files (the sister language of Psybergram, a text markup language) have a .PSYT extension. Psybergram 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 Psybertags
- 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.

Psybertags

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

Psybergram 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 Psybergram 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