Cooperscript

Cooperscript (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). Cooperscript source files have a .COOP extension. Coopertags files (the sister language of Cooperscript, a text markup language) have a .CPTG extension. Cooperscript boasts an ultra-simple Lisp-like syntax unlike all other languages; target platforms include mobile and desktop. COOPERScript: Compact Object Oriented Programming Environment and Runtime System.

Special Characters

Core:

- ( ) grouping
- - word separator
- ; end of stmt.
- : dot operator
- " string delimiter
- \ escape char.

Operators:

- + - * / %
- = < >
- & | ^ ~ ! ?

Other:

- # comment
- {} block comment
- _ used in identifiers
- $ string prefix char.

Differences from Python

- Parentheses, not whitespace
- Operators come before their operands
- Integration with Coopertags
- 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

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/brace/double quote when the open delimiter is inserted.

Coopertags

Coopertags is a simplified markup language used to replace HTML. Mock JSON files using Coopertags syntax have a .CPJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Coopertags code can be embedded in the Cooperscript echo statement. Coopertags 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]*

Cooperscript call:

- [expr: <expr>]
- [exec: <stmt>...]
- [coop: <path>]

Grayscale Mode

By default, smartphone apps written in Cooperscript run in grayscale mode for all Cooporhood users, without color. Registered Cooporhood users pay a subscription fee of $10/year to unlock RGB mode, enabling display of full color graphics. All screen elements include an optional 8-bit grayscale color setting, which is used in grayscale mode. If the grayscale color setting is omitted, an average of the three 8-bit RGB color settings is used. All non-Cooporhood mobile users run the apps with RGB mode unlocked. All Cooporhood profits are used to subsidize Psyberhood, an online community of people with mental health issues.

Feature Chart

Free	$10/year	$2 spend	Features
X	–	–	Ads
–	X	X	No ads
–	X	X	Browse Cooporgroops
X	–	X	Grayscale Apps
–	X	–	Color Apps
X	N/A	N/A	Color Apps (non-members)

Psyberhood

The Psyberhood consumer/survivor members meet in coffeeshops using smartphones to communicate orders to the outing leader, who places the order, pays with a gift card, obtains the receipt, enters the order amounts, tax and total, and shows the receipt to the other members (who verify their order amounts). All members are expected to take turns being outing leaders or deputy leaders. Low-functioning outing leaders are assisted by deputy leaders. Members must have PayPal or a credit card, alternatively, they can use cash to pay their coffeeshop tabs which is handled by a participating mental health organization. Coffeeshop orders are subject to a pretax 5 percent surcharge payable to Psyberhood. Members who haven't gone on outings for a single calendar month or more are exposed to ads.

Social Media

The online community of members (which is smartphone-based) is similar to Facebook. Each member has one or more friends who are also members. Members submit posts and comments on other posts/comments. Posts which are friends-only are only visible to friends of the original poster. Public posts are visible to everyone. Posts are moderated by mental health organizations.

Structure of Psyberhood

Psyberhood membership is divided into subgroups. Beneath the top-level group of users, different lower-level groups of users exist for different diagnoses, such as depression, bipolar, and schizophrenia. An example of an even more specialized group of users consists of "schizophrenia" AND "family members", where the "family member" category is a sub-category of "role" located under the nonprofit category. Groups of users exist for organizations which serve consumer/survivors such as CAMH and Progress Place. Large organizations such as CAMH can have separate groups of users for departments and teams. Local groups of users consist of a category associated with a geographic region, combined with another category using the boolean AND operator.

Population Subsets

Every Psyberhood group of users is defined by a given category, or 2 or more categories combined with boolean operators (and, or, not). Categories are organized in a tree of arbitrary depth. Any given category can appear in more than one place in the category tree. What follows is a list of top-level categories and an assortment of their immediate sub-categories:

- Nonprofit (Populations, Organizations, Jobs)
- Mental Health (Job Types, Disorders)
- Geography (Countries, Regions, Cities, Neighbourhoods, Languages, Races)

Cooporhood

Cooporhood.com is a generalized version of Psyberhood. Each Cooporgroop brings together a corresponding subset of the population. Subsets include truck drivers, dentists, social workers, immigrants, ex-convicts, etc. Members of every Cooporgroop get together in coffee shops and share social media posts. Members who choose to become Patrons donate between 5 and 20 percent of their orders to nonprofits, on top of the 5 percent which all members pay. Moderators do not scan every post. Members click on the octagon icon with an exclamation point inside it to report problem posts to the moderators (other problem posts are detected using machine learning). Posts can be public or friends-only, and some Cooporgroops are private: posts only visible to members. Organizations which require private Cooporgroops pay extra fees (registered charities are exempted). Members who don't make use of get-togethers for a single calendar month or more are exposed to ads, until they finally make use of a get-together. Premium users, who pay $10/year, can browse public Cooporgroops. Basic users, who pay no subscription fees, can also browse public Cooporgroops if they spent at least $2 in the previous calendar month.

Population Subsets

Every Cooporgroop is defined by a given category, or 2 or more categories combined with boolean operators (and, or, not). Categories are organized in a tree of arbitrary depth. Any given category can appear in more than one place in the category tree. What follows is a list of top-level categories and an assortment of their immediate sub-categories:

1. Business (Occupation, Industry, Companies, Public Sector)
2. Science (Physics, Chemistry, Biology)
3. Humanities (Philosophy, History, Political Science, Linguistics)
4. Education (Graduate Level, Undergraduate, Secondary, Primary, Schools)
5. Mathematics (Algebra, Calculus, Geometry, Arithmetic)
6. Information Technology (Languages, Platforms, Hardware, Software Genres, Companies)
7. Art (Literature, Visual, Music, Theatre, Movies & Television)
8. Sports (Baseball, Football, Basketball, Hockey, Soccer)
9. Nonprofit (Populations, Organizations, Jobs)
10. Health (Job Types, Disorders)
11. Geography (Countries, Regions, Cities, Neighborhoods, Languages, Races)
12. Religion (Christianity, Islam, Judaism, Hinduism)

Psybergiving

Psybergiving is a feature of Cooporhood meant to raise money for the nonprofit sector. Members who choose to become Patrons donate between 5 and 20 percent of their orders to specific nonprofit organizations (who have their own Cooporgroops) or those organizations which serve specific populations, on top of the 5 percent which all members pay. This is a powerful feature of Cooporhood, which along with the face-to-face contact make it superior to Facebook. Plus members don't see ads, unless they skip the get-togethers for a single calendar month or more.

Customization and Communities

Cooporgroops can be customized by the Cooporgroop leaders using a new programming language called Cooperscript and a text markup language called Coopertags. The 2 built-in communities are Psyberhood and Cooporhood. Other, alternative communities make use of the Cooporhood social network graph: a network of Cooporhood users. All communities except Psyberhood are implemented using Cooperscript and Coopertags. Cooporhood.com receives 15 percent of revenue earned by the alternative communities.

Blogs

Blogs are free for all users, except blogs which make use of server-side Javascript code (automatically converted from Cooperscript). Blog posts and comments can contain Coopertags code, as well as client-side Cooperscript code, not just plain text. Only premium users can submit blog posts which make use of server-side code.

Private Cooporgroops

Private Cooporgroops differ from public Cooporgroops in that posts are only visible to Cooporgroop members. Organizations which need private Cooporgroops pay 10 percent of the coffee shop order amounts instead of 5 percent, splitting these private Cooporgroop fees with their members 50-50 if desired. Registered charities are exempt from paying the private Cooporgroop fees.

Profitability

Assume that Cooporhood employs 3 moderators who each screen about 1050 posts/week and each make $600/week. Each moderator spends 2 minutes screening each post, 30 posts/hour, 1050 posts/week based on 35 hour week (say 1000 posts/week to use a round figure), and the moderators earn $17.14/hour. Assume each member posts once per week, and moderators screen every post. Then ratio of members to moderators is 1000:1. Assume each member goes on one outing per week, spending $2 per outing. Then weekly revenue per member is $0.10. Assume moderators only screen 2 percent of posts. Machine learning and members flagging problem posts makes this possible. Then ratio of members to moderators is 50,000:1.

The number of members is 150,000, so weekly revenue is $15,000. Assume 3 full time employees exist (Mike does not draw a salary, unlike the other co-founder, the iOS programmer, and one other employee). They are each paid $25/hour, or $1000/week. Weekly wages are $1800 for the moderators plus $3000 equals $4800. After paying wages, the amount left over to pay expenses is $10,200/week or $530,000/year.

About Us

I am Mike Hahn, the founder of Cooperscript.org. 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.

Contact Info

Mike Hahn
Founder
Cooperscript.org
2495 Dundas St. West
Ste. 515
Toronto, ON M6P 1X4
Canada

Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: treenimation.net/hahnbytes/

Psyberhood Project Summary

The Psyberhood consumer/survivor members meet in coffeeshops using smartphones to communicate orders to the outing leader, who places the order, pays with a gift card, obtains the receipt, enters the order amounts, tax and total, and shows the receipt to the other members (who verify their order amounts). All members are expected to take turns being outing leaders or deputy leaders. Low-functioning outing leaders are assisted by deputy leaders. Members must have PayPal or a credit card, alternatively, they can use cash to pay their coffeeshop tabs which is handled by a participating mental health organization. Coffeeshop orders are subject to a pretax 5 percent surcharge payable to Psyberhood. Members who haven't gone on outings for a single calendar month or more are exposed to ads.

The online community of members (which is smartphone-based) is similar to Facebook. Each member has one or more friends who are also members. Members submit posts and comments on other posts/comments. Posts which are friends-only are only visible to friends of the original poster. Public posts are visible to everyone. Posts are moderated by mental health organizations.

Cooporgroops

Cooporgroops are subsets of the online community of consumer/survivors. Each diagnosis, mental health organization, team, member role, and geographic region has its own cooporgroop. Roles include survivor, friend/relative, and professional. Each team is a subset of a given mental health organization. Compound cooporgroops consist of two or more cooporgroops (which themselves may be compound cooporgroops) combined with boolean operators (and, or, not).

Cooporhood is a generalized version of Psyberhood. Cooporgroops can be any conceivable category of members. Different occupations, ethnic groups, organizations, academic subjects, pop culture topics, etc. are all possible cooporgroops. Members who choose to become Patrons donate 5 to 20 percent of their coffeeshop orders to nonprofit organizations. Some cooporgroops are private: posts only visible to members. For-profit organizations which need private cooporgroops pay a 10 percent surcharge on coffeeshop orders, and may choose to split those fees 50-50 with their members.

Cooperscript and Coopertags

Cooporgroops can be customized by modifying their Cooperscript/Coopertags code (new programming/text markup languages respectively). Cooporhood subscribers, who pay $10/year, along with other users who spent at least $2 in the previous/current calendar month, can browse public cooporgroops and see no ads. Cooporhood subscribers can run third-party Cooperscript smartphone apps in full color mode, whereas Cooporhood users who are non-subscribers are limited to running those smartphone apps in grayscale mode.

The 2 built in social media communities are Psyberhood and Cooporhood. Alternative communities are written in Cooperscript and Coopertags. Cooporhood.com receives 15 percent of revenue earned by the alternative communities. Each new community imports its membership database from an existing community.

Implementation Steps

1. Develop foundation of Cooperscript code execution - ***almost done!***
2. Develop rest of Cooperscript code execution
3. Release Cooperscript as console-based compiler on GitHub
4. Learn web programming using Node.js
5. Develop web-based prototype of Psyberhood
6. Use CAMH and/or Progress Place to recruit testers
7. Hire Java programmer who is on autism spectrum, as co-founder
    - Use Specialisterne, they find IT jobs for people on spectrum
8. Co-founder to develop Psyberhood smartphone app
9. Mike to develop core of CRUNE:
    1. Cooperscript RUNtime Environment (CRUNE)
    2. Write Coopertags (monospaced) design specs
    3. Implement GUI: monospaced mode
    4. Release Cooperscript/GUI on GitHub
10. Make pitch to DMZ tech incubator at Ryerson
11. Search for angel investor
12. Co-founder to assist Mike in developing CRUNE
13. Mike to develop full version of CRUNE:
    1. Write Coopertags (full version) design specs
    2. Develop Coopertags
    3. Integrate Cooperscript with Coopertags
14. Steps 15 and 16 are simultaneous
15. Mike to lead development of Cooperscript SDK:
    1. Develop Cooperscript code editor
    2. Expand code editor to Cooperscript SDK
    3. Develop monetizing functionality
    4. Release Cooperscript SDK
16. Co-founder to integrate CRUNE with Android: Cooperscript Library
17. Upon failure of angel investor search:
    1. Co-founder is laid off
    2. Mike to develop rest of project, including iOS app
18. Step 19 begins whenever either Step 15 or Step 16 is completed
19. Develop iOS version of CRUNE
    1. Convert CRUNE to Swift
    2. Hire Swift/Java programmer to develop iOS app
20. Launch Android and iOS Psyberhood apps
21. Posts moderated by mental health organizations
22. Develop Cooporhood app using Cooperscript/Coopertags
23. Launch Cooporhood app
24. Purchase Google AdWords advertising
25. Hire moderators (or use besedo.com)
26. Develop prototype communities using Cooperscript
27. Release source code of prototype communities
28. Implement machine learning to detect problem posts
29. Nice to have, not essential:
    1. Implement Keyboard Aid (bells and whistles of editor)
    2. Develop WYSIWYG Coopertags screen editor
    3. Implement Coopertags-to-HTML converter
    4. Implement Cooperscript-to-Javascript converter

Cooperscript Grammar

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

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*

<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 Cooperscript 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