Mike Hahn - hahnbytes@gmail.com

Cooperhub

Cooperhub is an Android/iOS app which takes as input source code written in Cooperscript and Coopertags (programming and text markup languages, respectively), for each Cooperscript app. Developers write the Cooperscript apps, and earn money from end-users. The end-users run the Cooperhub app and select the desired Cooperscript app, defaults to most recently used app. So the Cooperhub mobile app is capable of running multiple Cooperscript apps. The business model is freemium and ad-supported.

Business Model

Gold users pay $20/year and can run any Cooperscript app. Silver users pay $10/year and can only run 1 Cooperscript app in any given calendar month. Bronze users pay no fees, see ads, and can only run the browser-based versions of each Cooperscript app (a conversion utility converts Coopertags and Cooperscript code to HTML and Javascript, respectively). The developers of Cooperscript apps receive a share of Cooperhub net revenue (developer fees) in proportion to the number of user clicks generated by the mobile users (not browser-based) of their apps. Net revenue equals gross revenue minus expenses such as web hosting fees, marketing expenses, and employee wages. The founder receives 5 to 10 percent of net revenue instead of being paid a salary.

Browser-Based Incentive

Developers of Cooperscript apps in which the number of user clicks/keystrokes of the browser-based version is less than that of the Cooperhub app version are penalized. In this case the difference between Cooperhub app and browser-based user click counts is divided by half of the total user click counts, giving a number between 0 and 2. This number is subtracted from 2 and then divided by 2, yielding a number between 1 and 0, which is multiplied by the revenue which would have been received by that developer had the penalty not been applied, and the final result is the actual revenue amount. Some Cooperscript apps don't have browser-based versions. The corresponding developer fees are one-third of those apps which do have browser-based versions.

Penalty for High Bandwidth Apps

Developers of Cooperscript apps which exceed the bandwidth quota per 1000 user clicks are penalized. For every 1000 user clicks, take the ratio between bandwidth used and the quota. If it's equal to 1, no penalty. If it's equal to 5, halve the revenue of that developer. If it's equal to 25, halve the revenue twice, and so on. Use a logarithmic formula to calculate the actual revenue if the ratio is not an exact power of 5.

Paid Apps

Some Cooperscript apps enable in-app purchases, one-time fees or subscription fees payable by the user, which are only available to gold class users. After paying an approximate 30 or 15 percent commission to Google or Apple, these funds are paid to the developer of the Cooperscript app.

Sample App

Cooperhub is bundled with a sample Cooperscript app called Imajette, which is a tool used to organize and share your image collection. Gold and silver class users can run the mobile app version of Imajette, whereas bronze class users can only run the browser-based version of Imajette.

Partners

Cooperhub can be used as a tool for teaching math and fostering online communities of consumer/survivors. Organizations which teach adult literacy and numeracy will be recruited to be test beds for teaching math. Organizations which serve consumer/survivors (people with mental health issues) will be recruited to be test beds for online communites created with Cooperhub. Registered charities can use any Cooperhub app for free: all of their clients/employees register with Cooperhub and enjoy gold class status.

Imajette

Imajette is a tool used for organizing image collections, written in Cooperscript. 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. The images are stored on a desktop/laptop computer, and displayed on a phone/tablet. The software that manages the stored images is written in Cooperscript and uses Jetty as a web server. The display of the images on the mobile device is browser-based.

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 features are available in the Imajette application installed on the desktop/laptop computer, such as adding/deleting images, features and categories.

Image Sharing

If image sharing is technically feasible (and it surely is), Imajette can be made much more powerful. Only users who have converted are allowed to share their images with other users. 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 Cooperhub.com website. Instead of downloading all the images, only the file names are downloaded, each consisting of 16 hex digits. The web server on the local drive serves HTML consisting of remote image links such as the following:

https://cooperhub.com/img/6f374e80ca15a3e4.jpg

Social Media

If image sharing is technically feasible, then perhaps so is social media. Users can share posts and comments with followers (plain text or Coopertags code), but only users who have converted can share images. Imajette has a freemium business model, so users never see ads.

Partners

Cooperhub can be used as a tool for teaching math and fostering online communities of consumer/survivors. (A consumer/survivor is someone with mental health issues.) The founder is a member of a clubhouse for consumer/survivors known as Progress Place. If that organization agrees to let me use them as a test bed for my online community, then they could act as beta testers and maybe even customers. An online community is similar to social media such as Facebook or Twitter, but without ads and fully customizable for each participating organization. Additional online communities can be created for the different classes of consumer/survivors: depression, bipolar, schizophrenia, PTSD, anxiety, etc., as well as family members, friends, and professionals related to or involved with the consumer/survivors. Organizations which teach adult literacy and numeracy will be recruited to be test beds for teaching math.

Users Without Smartphones

Most Cooperhub apps can run in browser-based mode, enabling clients of nonprofit organizations without smartphones to fully participate in the Cooperhub community. In case they don't have Internet or computers of any kind, many organizations provide public computer access, usually Windows desktop computers.

Teaching Math Locally

Students display the curriculum on a desktop/laptop computer running the Mathgrid (a whiteboard used to teach math). Teachers and tutors display a window of the student'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 Math Remotely

Teachers and tutors can teach remotely using a desktop/laptop computer instead of a smartphone. A chat window which is always on top except when hidden facilitates communication between teacher and student. WebSocket is used to facilitate communication between clients (teachers/students running a browser) and the server.

Mathgrid

Math is taught using text in monospaced mode. 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 Cooperscript programming language. Each step in the math problem being solved manipulates this Cooperscript 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 Mathgrid 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).

Projected Revenue

These 5 scenarios assume that Cooperhub makes it to the big time. If I only have a few hundred users, I obviously can't afford to hire any paid employees, and little or no profit is achieved. Assume each user generates $1 of ad revenue per year. This is slightly conservative since 5 percent of users see no ads.

1. **Minimal scenario:** Assume number of converted Cooperhub users is 1,000, conversion rate of 5 percent, all converted users are gold class users, 30 Cooperscript developers exist, annual expenses are $10,000 for Google AdWords advertising of 6K and web hosting of 4K. Then gross annual revenue equals 20,000 x (0.05 x 20 + 1) = $40,000, annual expenses equals $10,000, net annual revenue equals 40,000 - 10,000 = $30,000, founder's salary equals 30,000 x 0.05 = **$1,500** = $125/month, total annual expenses equals 10,000 + 1,500 = $11,500, total developers' fees equals 40,000 - 11,500 = 28,500, therefore each developer receives 28,500 / 30 = $950/year = $79/month on average.
2. **Conservative scenario:** Assume number of Cooperhub users is 200,000, conversion rate of 5 percent, all converted users are gold class users, 100 Cooperscript developers exist, annual expenses are $10,000 each for both Google AdWords advertising and web hosting. Then gross annual revenue equals 200,000 x (0.05 x 20 + 1) = $400,000, annual expenses equals $20,000, net annual revenue equals 400,000 - 20,000 = $380,000, founder's salary equals 380,000 x 0.05 = **$19,000**, total annual expenses equals 20,000 + 19,000 = $39,000, total developers' fees equals 400,000 - 39,000 = 361,000, therefore each developer receives 361,000 / 100 = $3,610/year = $301/month on average.
3. **Medium scenario:** Assume number of Cooperhub users is 500,000, conversion rate of 5 percent, all converted users are gold class users, 100 Cooperscript developers exist, annual expenses are $25,000 for Google AdWords advertising of 10K and web hosting of 15K, and employee wages (including founder) happen to equal developers' fees. Let x = total employee wages not including founder, and let y = founder's salary = 5 percent of net revenue. Developers' fees = x + y. Then gross annual revenue equals 500,000 x (0.05 x 20 + 1) = $1,000,000, annual expenses equals $25,000, and the following Equation (A) holds: 1,000,000 - 25,000 = 2(x + y). So net annual revenue equals 1,000,000 - 25,000 - x - y = $975,000 - x - y. Now Equation (B) holds: y = 0.05(975,000 - x - y), or 20y = 975,000 - x - y, or 21y = 975,000 - x, so Equation (B) becomes: x + 21y = 975,000, and Equation (A) is: 975,000 = 2x + 2y = x + 21y. So x = 19y, and 19y + 21y = 975,000, 40y = 975,000, then y = founders salary = **$24,375**, and employee wages = x = 19y = **$463,125**, and using Equation (A), developers' fees equal x + y = 975,000 / 2 = $487,500, so each developer receives 487,500 / 100 = $4,875/year = $406/month on average.
4. **Optimistic scenario:** Assume number of Cooperhub users is 1,000,000, conversion rate of 5 percent, all converted users are gold class users, 100 Cooperscript developers exist, annual expenses are $25,000 for Google AdWords advertising of 10K and web hosting of 15K, employee wages one full-time and one part-time add up to **$100,000**. Then gross annual revenue equals 1,000,000 x (0.05 x 20 + 1) = $2,000,000, annual expenses equals $25,000, let y = founder's salary and let z = net annual revenue equals 2,000,000 - 25,000 - 100,000 - y, or z = $1,875,000 - y. Now y = (0.05)z, and y = 0.05(1,875,000 - y), or 20y = 1,875,000 - y, and 21y = 1,875,000, so y = **$89,286**, and z = 20y = total developers' fees, so each developer receives 20y / 100 = y / 5 = $17,857/year = $1,488/month on average.
5. **Many Employees scenario:** Assume everthing in Scenario 4 is the same except each developer receives $3,600/year, several employees exist, and founder's salary equals 10 percent of net revenue instead of 5 percent. Let x = total employee wages except founder, and let y = founder's salary. Net annual revenue equals 2,000,000 - 25,000 - x - y = 1,975,000 - x - y = 360,000 = 3,600(100), founder's salary equals y = (1,975,000 - x - y) / 10, or y = 360,000 / 10 = **$36,000**, now 1,975,000 - y - x = 360,000, so x = 1,975,000 - 36,000 - 360,000 = **$1,579,000**.

About Us

I am Mike Hahn, the founder of Cooperhub.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 Cooperscript code execution - ***almost done!***
2. Approach Progress Place: online communities
3. Approach West Neighbourhood House: teaching math
4. Develop rest of Cooperscript code execution
5. Release Cooperscript as console-based compiler on GitHub
6. Implement GUI: monospaced mode
7. Release Cooperscript/GUI on GitHub
8. Write Coopertags design specs
9. Develop Coopertags
10. Integrate Cooperscript with Coopertags
11. Cooperscript/Coopertags: Cooperscript Runtime Environment (CRE)
12. Develop Imajette for Linux
13. Use Specialisterne to hire local Android programmer on spectrum:
    - they find tech jobs for those on autism spectrum
14. Develop Imajette app
15. Implement image sharing
16. Make pitch to DMZ tech incubator
17. Develop Android Cooperhub app
18. Use Specialisterne to hire remote iOS programmer on spectrum
19. Convert CRE to Swift
20. Develop iOS Cooperhub app
21. Search for angel investor
22. Without angel investor, do not renew contracts of autistic programmers
23. Develop converter: Cooperscript/Coopertags to Javascript/HTML
24. Develop monetizing functionality
25. Launch website
26. Purchase Google AdWords advertising
27. Develop Cooperscript code editor
28. Implement Imajette social media
29. Implement online communities
30. Implement Mathgrid: teaching math
31. Implement Keyboard Aid (bells and whistles of editor)
32. Develop WYSIWYG Coopertags screen editor
33. Exit strategy: if necessary, release Java code of CRE on GitHub

Exit Strategy

In case Cooperhub is not profitable, the Java (and Swift) source code of the CRE will be released on GitHub. This can be used to create standalone Android and iOS mobile apps by bundling the CRE with the Cooperscript/tags source code of each app in the Cooperscript app store.

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, short for Compact Object-Oriented Programming Environment and Runtime System, 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 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

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.

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]*

Call: (Cooperscript code)
- [expr: <expr>]
- [exec: <stmt>...]
- [coop: <path>]

Monetizing Cooperscript

When open source Cooperscript and Coopertags are up and running, the Cooperhub mobile app will be developed. Both of those languages are embedded in the Cooperhub app. A utility converts Coopertags and Cooperscript code to HTML and Javascript, respectively. This utility is used by browser-based versions of multiple Cooperscript apps. Cooperhub executes the multiple Cooperscript apps.

Cooperscript 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 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